THE MICROSOFT JOURNAL FOR DEVELOPERS

NOVEMBER 15, 2016 VOL 31 NO 12

msdn magazine

Special Issue

Be Unstoppable

Create high-impact Windows® user experiences with DevExpress

THE MICROSOFT JOURNAL FOR DEVELOPERS

NOVEMBER 15, 2016 VOL 31 NO 12

msdn magazine

Special Issue

Connect(); Special Issue

Microsoft

WE'RE CHANGING THE WAY YOU LOOK AT REPORTING

The Text Control Reporting Framework combines powerful reporting features with an easy-to-use, MS Word compatible word processor.

Our award-winning developer libraries are completely independent from MS Word or any other third-party application and can be completely integrated into your business application.

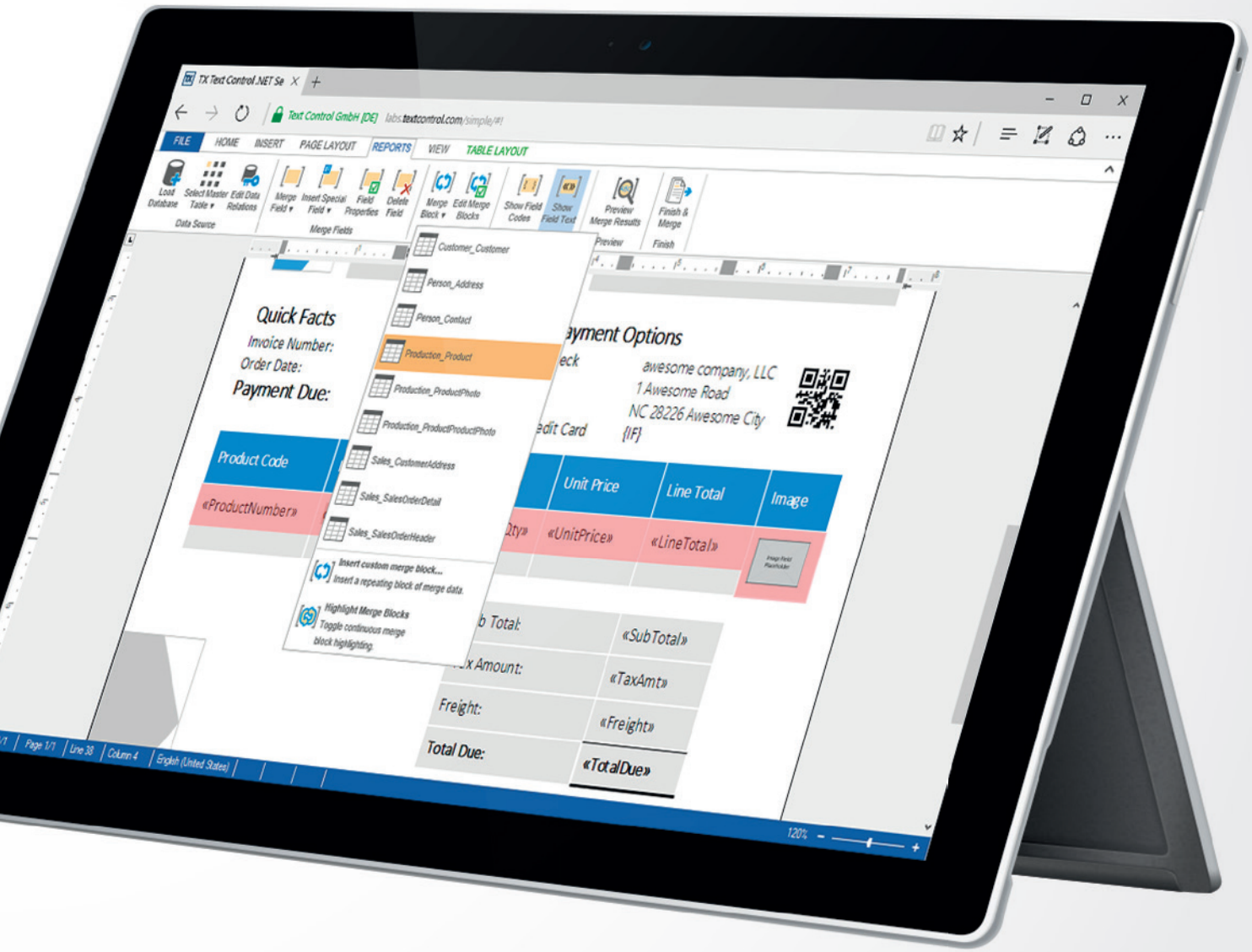
Follow the trend and switch to flow type layout reporting.

www.textcontrol.com www.reporting.cloud

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

Visual Studio Partner

REPORTING LIBRARIES FOR WINDOWS, WEB, MOBILE AND CLOUD APPLICATIONS



TEXT CONTROL

msdn magazine

NOVEMBER 15, 2016 VOLUME 31 NUMBER 12

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

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

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

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

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

Reaching the Staff

Staff may be reached via e-mail, telephone, fax, or mail. A list of editors and contact information is also available online at Redmondmag.com.

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

Irvine Office (weekdays, 9:00 a.m. – 5:00 p.m. PT)
Telephone 949-265-1520; Fax 949-265-1528
4 Venture, Suite 150, Irvine, CA 92618

Corporate Office (weekdays, 8:30 a.m. – 5:30 p.m. PT)
Telephone 818-814-5200; Fax 818-734-1522
9201 Oakdale Avenue, Suite 101, Chatsworth, CA 91311

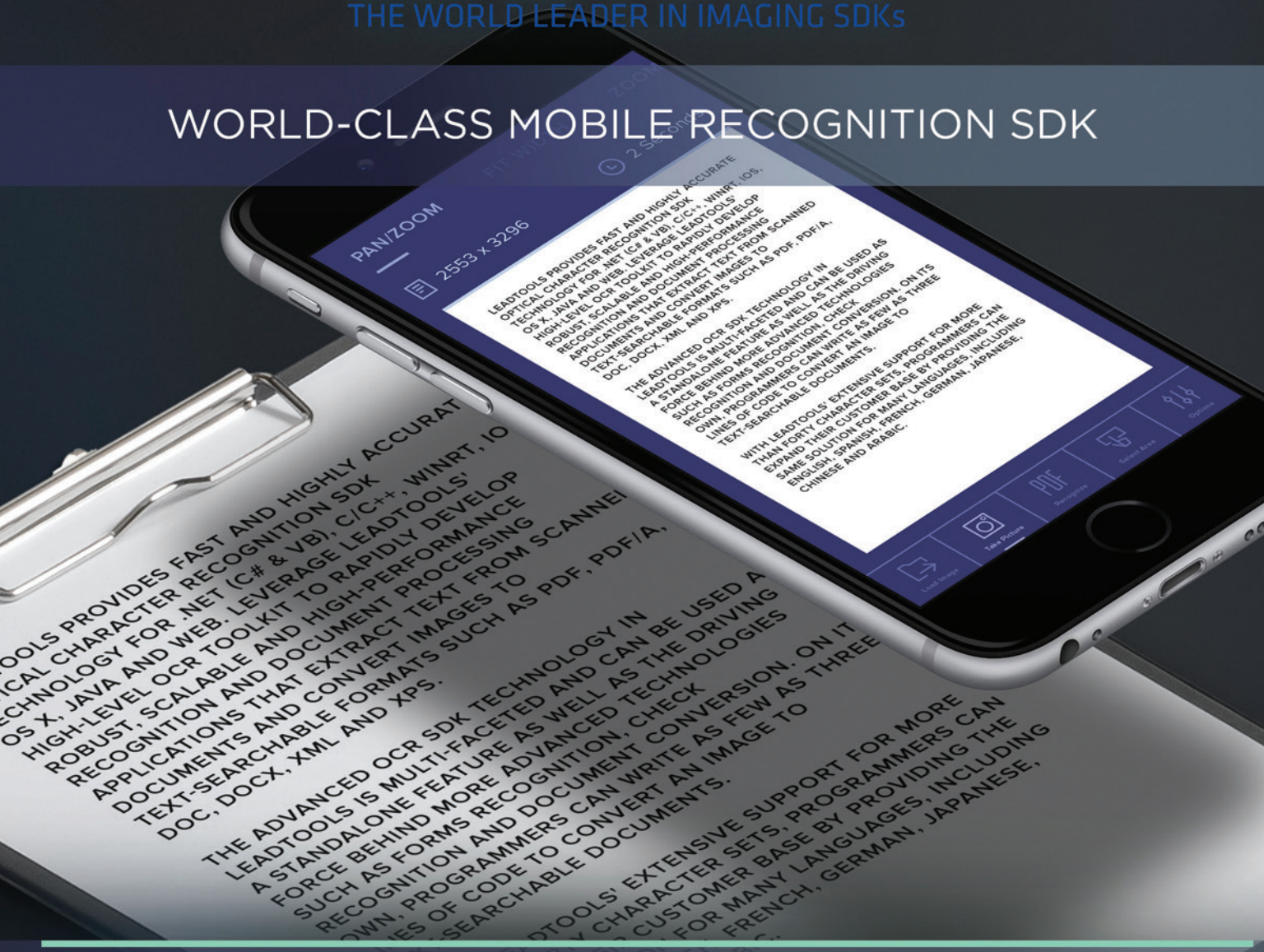
The opinions expressed within the articles and other contentsherein do not necessarily express those of the publisher.

Microsoft

LEADTOOLS®

THE WORLD LEADER IN IMAGING SDKs

WORLD-CLASS MOBILE RECOGNITION SDK



LEADTOOLS recognition engines enable developers to easily integrate fast and accurate OCR, Barcode, Driver's License, Passport, and Credit Card recognition functionality into their mobile apps.

Download fully functional sample apps built with the LEADTOOLS SDK.

Google play | Download on the App Store | Microsoft

DOWNLOAD OUR 60 DAY EVALUATION
WWW.LEADTOOLS.COM

LEAD TECHNOLOGIES INCORPORATED

SALES@LEADTOOLS.COM
800.637.1840

Editor's Note

MICHAEL DESMOND

Reconnect();

Two years ago we published a special issue of *MSDN Magazine* focused on Microsoft's tools and services efforts in the wake of the 2014 Connect(); conference in New York City. That event was remarkable in that it transformed the dialog Microsoft was having with developers. Cross-platform, open source, mobile and cloud were the points of emphasis at the 2014 conference, and the seeds Microsoft planted then have gone on to bear fruit today.

On Nov. 16 Microsoft holds its third Connect(); conference, and as was the case at the first event, the focus is on breaking down borders, reaching across platforms and maximizing developer productivity. The difference this year is that the nascent tools and platforms Microsoft touted in 2014 have matured, from innovative frameworks like .NET Core and ASP.NET Core, to cross-platform tools that leverage Xamarin technology such as Visual Studio for Mac and Visual Studio Team Services. Microsoft is paying off on the promises it made in 2014, and the impact on developers will be profound.

Our exploration of the innovations coming out of Connect(); begins with Kasey Uhlenhuth's dive into the productivity enhancements coming in Visual Studio 2017, while Mikayla Hutchinson explores the new Visual Studio for Mac IDE, which brings the first-class Visual Studio development experience to developers on the Apple flagship platform. *MSDN Magazine* columnist Mark Michaelis follows with a deep exploration of the upcoming C# 7.0 programming language, which adds compelling features like deconstructors, local variables and improved pattern matching.

The Microsoft acquisition of Xamarin this year was a game changer, and its impacts are on display both at Connect(); and in this issue. Former *MSDN Magazine* columnist Charles Petzold returns to our pages to show how native view embedding improves cross-platform mobile development by letting developers directly reference iOS, Android and Windows native controls within Xamarin.Forms XAML files. Tyler Whitney shows how Xamarin can be used to target the Universal Windows Platform alongside iOS and Android, while Craig Dunn dives into the interactive documentation and live coding features of Xamarin Workbooks that let developers learn and experiment with the entire native SDKs for Android, iOS, macOS and Windows Presentation Foundation.

The Microsoft acquisition of Xamarin this year was a game changer, and its impacts are on display both at Connect(); and in this issue.

There's a lot more, including Jean-Marc Prieur's and Sam Guckenheimer's feature on ruggedized DevOps and how it injects security into the development and release pipeline. And don't miss Omid Afnan's dive into Big Data and Yina Arenas' exploration of the API-driven capabilities of Microsoft Graph.

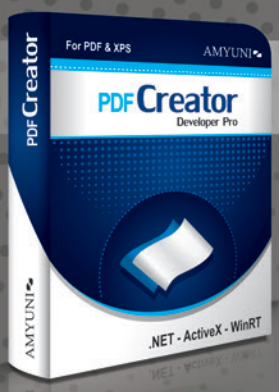
Finally, check out our additional coverage on the *MSDN Magazine* Web site. Thomas Dohmke shows how Microsoft is taking DevOps to the next level, enabling developers to connect a repository and then build, test, deliver and monitor apps from a single dashboard connected with an Azure back end (msdn.com/magazine/mt790198). Also featured is Justin Raczak's examination of Xamarin Test Cloud in mobile app development (msdn.com/magazine/mt790199), and Michael Rys' dive into U-SQL and its use in Big Data applications (msdn.com/magazine/mt790200).

Microsoft Connect(); carries forward the work that the conference first laid out two years ago. We hope this special issue of *MSDN Magazine* will help you to take full advantage of all the advances we expect to see in the years to come.

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

Switch to Amyuni PDF

AMYUNI



Create and Edit PDFs in .NET, COM/ActiveX, WinRT & UWP

NEW v5.5

- Edit, process and print PDF 1.7 documents
- Create, fill-out and annotate PDF forms
- Fast and lightweight 32- and 64-bit components for .NET and ActiveX/COM
- New Universal Apps DLLs enable publishing C#, C++, CX or Javascript apps to windows Store
- Updated Postscript/EPS to PDF conversion module

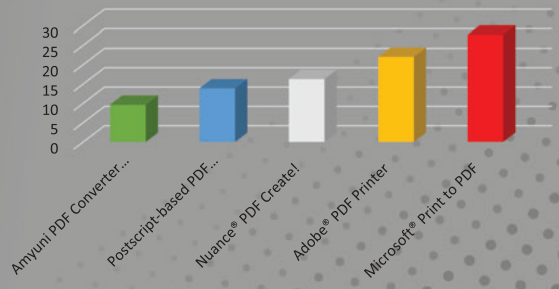
Complete Suite of Accurate PDF Components

- All your PDF processing, conversion and editing in a single package
- Combines Amyuni PDF Converter and PDF Creator for easy licensing, integration and deployment.
- Includes our Microsoft WHQL certified PDF Converter printer driver
- Export PDF documents into other formats such as Jpeg, PNG, XAML or HTML5
- Import and Export XPS files using any programming environment

High Performance PDF Printer for Desktops and Servers

- Print to PDF in a fraction of the time needed with other tools. WHQL tested for all Windows platforms. Version 5.5 updated for Windows 10 support

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



Other Developer Components from Amyuni®

- WebkitPDF: Direct conversion of HTML files into PDF and XAML without the use of a web browser or a printer driver
- PDF2HTML5: Conversion of PDF to HTML5 including dynamic forms
- Postscript to PDF Library: For document workflow applications that require processing of Postscript documents
- OCR Module: Free add-on to PDF Creator uses the Tesseract engine for character recognition
- Javascript engine: Integrate a full Javascript interpreter into your applications to process PDF files or for any other need

All development tools available at

www.amyuni.com

AMYUNI Technologies

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

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

A Quick Look at Productivity Enhancements in Visual Studio 2017 RC

Kasey Uhlenhuth

If you want to be more productive, you have to save more time or more effort on the tasks you perform every day. Visual Studio 2017 RC boosts your productivity with automated tasks and improved navigation, as well as with immediate feedback on the quality and state of your code.

The strides Visual Studio has made in refactorings, code generation, code analysis, navigation, testing, and debugging for .NET developers were made possible by project "Roslyn," a six-year undertaking to re-architect the C# and Visual Basic compilers. Visual Studio 2017 RC leverages the Visual Studio 2015 investment in Roslyn to crank out tons of cool new features.

Navigating Code

When developers drill into a bug, discover the implications of a refactoring, or work to grok an unfamiliar code base, they rely on the accuracy and ease of code navigation. This release delivers a greatly improved navigation experience—with Find All References, GoTo and Indent Guides—to get you from A to B with greater confidence and fewer distractions (even in large code bases).

Previously, Find All References would discover all usages of a symbol and then output the results as a simple, flat list in the Results Window. Now, as **Figure 1** shows, the references are colorized and arranged in a hierarchical display with custom grouping, sorting, filtering and searching to help you rapidly home in on the particular reference you desire—even when there are many references in the list. Furthermore, hovering over a reference displays a tool tip to give foresight into where you'll be navigating to before you actually go there. You can save or "lock" your Find All References results for situations when you need to run the command several times but want your original results to persist (for example, when investigating and drilling down into the impact of a refactoring).

To save even more time, learn the handy keyboard shortcuts shown in **Figure 2**.

Another feature that has undergone significant improvement in this release is GoTo (formerly Navigate To). GoTo All is a fast, complete search for any file, type, member or symbol declaration in a solution. Icons at the bottom of the search bar allow you to filter your results by group or adjust the scope of the search, as shown in **Figure 3**. (But if you prefer the keyboard over the mouse, you can also take advantage of a query syntax that lets you filter directly by typing a simple prefix). The gear icon in the bottom-right corner

This article is based on a preview version of Visual Studio 2017 RC. All information is subject to change.

This article discusses:

- Code navigation enhancements
- Help with writing correct, readable code
- Improved testing and debugging

Technologies discussed:

Visual Studio 2017 RC

DevExpress®

Touch-Optimized Hybrid Apps

DevExpress Universal provides a comprehensive collection of UX tools so you can create touch-first apps that are built for today and ready for tomorrow.

Your Next Great Hybrid App Starts Here

Build touch-first apps for any Windows device and leverage existing code investments.
devexpress.com/hybrid

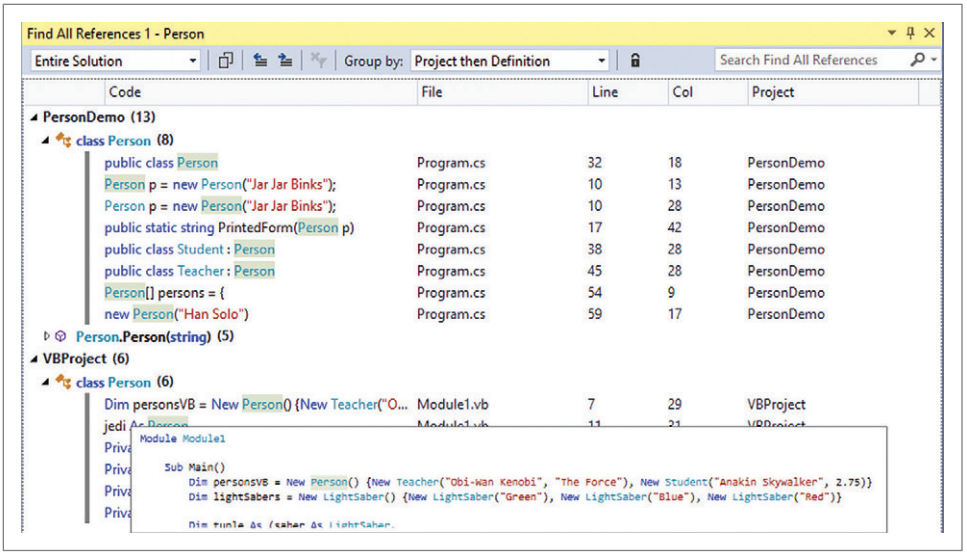


Figure 1 **Find All References Improvements Include Colorization, Custom Organization and Hover Previews**

lets you customize a handful of settings, including the placement of the search bar, a live preview of the files containing the highlighted result, and additional file information for each result.

Finally, you might recognize Indent Guides from the popular Productivity Power Tools extension. Now, as part of the core product, the dotted, gray vertical lines serve as landmarks in code to provide context within your frame of view. When hovering over a bottom brace, you get a colorized preview of the top matching brace, and when hovering over the guide itself, you get a colorized preview of the code surrounding and containing the matching top brace. These features aim to give you a better sense of your code "geography" and context without scrolling or navigating. Indent Guides also work well with the existing outlining features in Visual Studio and even offer more regions that are expandable and collapsible.

You can also enable "Map Mode" on your scroll bar. This transforms the scrollbar into a "map" of your code, allowing you to view a miniature version of the entire document—including errors, breakpoints and so forth. Hovering over any part of the "map" will display a preview of the code at that point in the document.

And, again, you can increase your efficiency by learning the useful keyboard shortcuts shown in **Figure 4**.

Writing and Reading Code

Along with navigation, developers spend a lot of time writing and reading code. Visual Studio 2017 RC focuses on facilitating the writing of correct code, as well as maintaining the readability of developer code bases. Building on features in Visual Studio 2015, this release provides a refined IntelliSense experience, more refactorings and code fixes, and customizable code-style configuration and enforcement.

Figure 2 **Keyboard Shortcuts for General Navigation**

Go To Definition	Peek Definition	Find All References	Go To Implementation	Go To All (File/Type/Member/Symbol)
F12	Alt+F12	Shift+F12	Ctrl+F12	Ctrl+T or Ctrl+,

IntelliSense The goal of Visual Studio is to assist rather than hinder the various code-writing workflows that are being used today. One of the most obvious ways Visual Studio makes you productive is with IntelliSense (see **Figure 5**). Visual Studio 2017 RC updates the IntelliSense experience with several enhancements: smart preselection, filtering and XAML support.

Smart preselection will determine the "target type" required at a position in code and will preselect items in the IntelliSense completion list matching that type. This speeds your typing flow and removes the burden of having to figure out the expected type at a given location. IntelliSense *filtering* allows you to filter the completion list by category; for example, you can filter out extension methods or view only events. This feature boosts productivity when you're working in a large code base where there are many items in the completion list or when dealing with unfamiliar code.

The goal of Visual Studio is to assist rather than hinder the various code-writing workflows that are being used today.

Finally, this release delivers a whole new experience for *XAML IntelliSense* to help developers bind quickly and correctly and see only relevant information. This smarter completion experience includes completion when binding events, paths and functions with x:Bind; camelCase matching support (for example, "RTB" will complete as "RichTextBox"); and namespace prefix autocompletion.

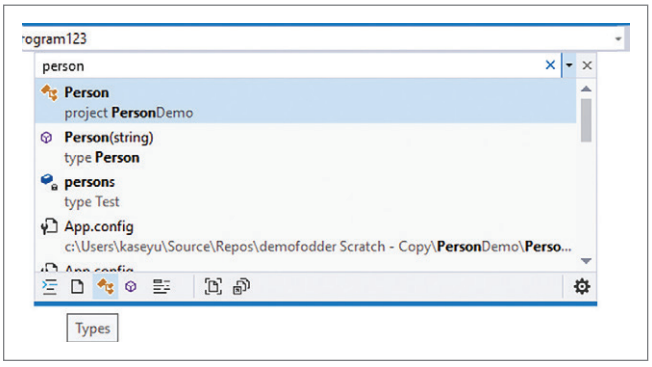


Figure 3 **Use Icons or Query Syntax to Filter Results by Files, Types, Members or Symbols**

Figure 4 **Keyboard Shortcuts for GoTo**

	Go To All	Go To Line	Go To File	Go To Type	Go To Member	Go To Symbol
Shortcut	Ctrl+T or Ctrl+,	Ctrl+G	Ctrl+1, F	Ctrl+1, T	Ctrl+1, M	Ctrl+1, S
Query Prefix	No prefix	:	f	t	m	#

Code Analysis Visual Studio 2015 introduced the live code analysis feature, which enables "as-you-type" feedback on your code. This lets you learn about issues early, before they build, rather than accumulating a set of problems you might never get around to fixing. To resolve the errors identified in live code analysis, you use the lightbulb menu or the shortcut "Ctrl+." to access code fixes and refactorings. Visual Studio 2017 RC takes live analysis and code fixes a step further by amplifying the set of refactorings and code fixes available, and by introducing code style analyzers that identify style issues in code as soon as they're typed.

> Maintaining a consistent, readable code base is a challenging endeavor.

Visual Studio 2015 included some core refactorings: extract method or interface, change method signature, inline temporary variable, introduce local variable, and remove unnecessary usings and imports. Visual Studio 2017 RC expands the set of refactorings and fixes to help you maintain a readable code base and catalyze your development workflows. For example, a significant number of developers initially write all their classes, interfaces, and other types in a single file and then extract each type into a file with the matching name later. Visual Studio 2017 RC expedites this process with the refactoring option "Move Type To Matching File." Other refactorings you can look forward to include:

- Sync file and type name
- Convert property to method
- Use object initializer
- Convert null-check + throw to use ?? + throw
- Convert string.Format to interpolated string
- Make method synchronous
- Add missing case
- Add braces

Additionally, this release introduces some basic code analysis and fixes for XAML. Using the same lightbulb mechanism in C# and Visual Basic, you can sort and remove unnecessary namespaces and add missing namespaces in your XAML files.

Maintaining a consistent, readable code base is a challenging endeavor. There are good reasons to aspire to a readable code base: If all code looks consistent, it's easier to onboard new developers; and if all code looks consistent, code reviews can focus on logic rather than on formatting and style minutiae. Visual Studio 2017 RC establishes a way to configure code style with built-in style rules and custom naming conventions.

In the RC, you can go to Tools | Options | Text Editor | [C#/Basic] | Code Style | General to see the built-in configurable style rules, which include preferring var over explicit type, predefined over framework type, this over nothing. Each rule comes with a description, a configurable preference and a configurable severity. The different severities work as follows:

- **Error** displays in the editor as a red squiggle, appears in the Error List and will break the build.
- **Warning** displays in the editor as a green squiggle, appears in the Error List and will only break the build if the configuration "treat warning as errors" is enabled.
- **Suggestion** displays in the editor as gray dots, appears in the Error List and will not break the build (see **Figure 6**).
- **None** has no display in the editor, does not appear in the Error List and will not break the build.

Because preferences for naming styles vary widely across the .NET developer community, you can create your own custom naming conventions to enforce on your team. Visual Studio 2017 RC provides a set of defaults that can be configured to enforce team conventions, including that members (except fields) should be PascalCase, types should be PascalCase, interfaces start with "I," and async methods end with "Async." You can configure and add to these defaults by going to Tools | Options | Text Editor | [C#/Basic] | Code Style | Naming. Naming rules are placed in a grid for configuration just like the standard style rules (note that order matters here—the first matching style rule is applied).

To add a custom naming rule, you click the green plus ("+") button at the bottom of the rule grid to open the rule creation dialog. You can then use default specifications (for example, "all classes") and styles (such as, "use prefix _") to craft a custom naming rule. If the built-in specifications and styles aren't sufficient to piece together your desired naming convention, you can write your own.

```
public MainWindow()
{
    InitializeComponent();
    ViewModel = new MainViewModel();
    this.WhenAnyValue(x => x.ViewModel).BindTo(this, x => x.DataContext);
    this.MC
}
```

ContextMenuClosing
GotMouseCapture
IsMouseCapturedChanged
IsMouseCaptureWithinChanged
IsMouseDirectlyOverChanged
LostMouseCapture
ManipulationCompleted
MouseDoubleClick
PreviewMouseDoubleClick

Figure 5 **IntelliSense Now Has Highlighting and Filtering**

```
int answer = 42;
```

use 'var' instead of explicit type
IDE0007 use 'var' instead of explicit type
...
int answer = 42;
var answer = 42;
...
Change style option
Preview changes
Fix all occurrences in: Document | Project | Solution

Figure 6 **Enforce Team Style and Naming Conventions in the Editor so You Get Live Diagnostics When You Violate Rules**

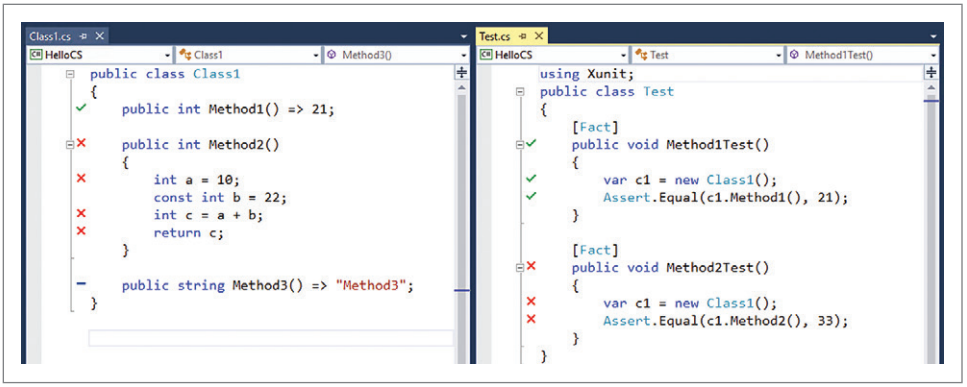


Figure 7 **Live Unit Testing Provides Instant Feedback on Whether Code Is Touched by Passing, Failing or No Tests**

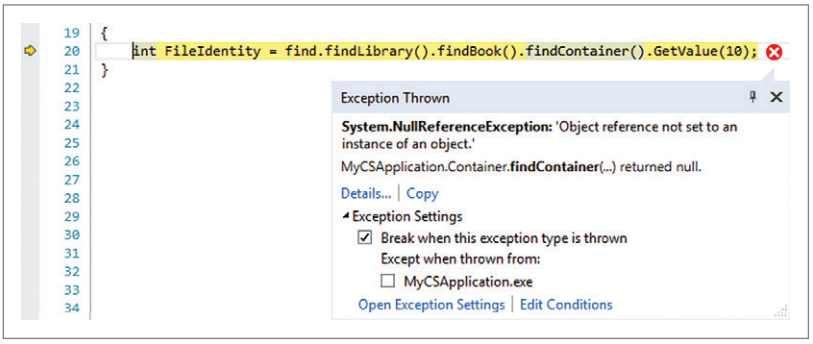


Figure 8 **The New Exception Helper Removes the Need to Drill into Dropdowns to Find the Most Important Information and Actions**

You can also adjust your style rules from the editor by pressing "Ctrl+." to trigger the lightbulb menu, selecting the rule fix in the menu and then clicking the gear icon in the live code preview.

Testing Code

Supporting unit tests creates an interesting tension between selecting enough tests to ensure code is correct while still running as few tests as possible so you don't have to wait so long for test results. This tradeoff between correctness and time has often left developers feeling unproductive and frustrated. To mitigate this stress, Visual Studio 2017 RC introduces Live Unit Testing for C# and Visual Basic (see **Figure 7**).

Live Unit Testing analyzes data generated at run time to run only impacted tests after an edit and provides immediate feedback on the status of the tests in the editor. These inline visualizations appear on a line-by-line basis:

- If a line of executable code is hit by at least one failing test, it's decorated with a red "x."
- If a line of executable code is hit by all passing tests, it's decorated with a green checkmark.
- If a line of executable code is hit by no tests, it's decorated with a blue dash.

The live code coverage and test result information provided by Live Unit Testing removes the burden of manually selecting and running tests. The live feedback also serves to notify you instantly if your change has broken the program—if inline visualizations shift from green checkmarks to red x's, you know you broke a test. At any point in time you can hover over the check or x to see what tests are hitting the given line. Additionally, you can navigate directly to that test by clicking on it in the hover tool tip.

If you're part of a team that practices test-driven development, Live Unit Testing gamifies the workflow; in other words, all tests will be red and failing at first, and as you implement each method, you'll see them turn green if they succeed. For all other developers, Live Unit Testing provides visual feedback for when they've broken their code.

Debugging Code

When all else fails, developers rely on debugging to help them identify the source of an issue. Visual Studio 2017 RC saves you time by reducing the number of actions required to step through a program and to drill into exception information.

Run To Click does exactly what it sounds like; it executes a program until it reaches the target line of code and breaks in debug mode. Essentially, it removes the need for developers to constantly add, hit and remove temporary breakpoints by combining all these actions into one click. To use this feature, you simply need to press the green "run to here" icon that appears to the left of the code line when you hover over it in debug mode.

Perhaps the biggest productivity improvement in the debugger is the new Exception Helper. The redesigned dialog displays the most important information from an exception at the top level, like inner exception details and the expression that returns null, as shown in **Figure 8**. Visual Studio 2017 RC also enables you to prohibit breaking on exception types in specific cases—allowing you to disregard exceptions thrown from third-party libraries or certain .dlls while debugging.

Wrapping Up

Visual Studio 2017 RC focuses on making you more productive by saving you time and effort. I'm thrilled to be able to share improvements in navigation with GoTo and Find All References, enhancements to IntelliSense both in C#/Visual Basic and XAML, an expansion of live code analysis with more refactorings and fixes and the addition of code style, an interactive way of testing with Live Unit Testing, and an efficient debugging experience with the new exception helper. I look forward to hearing your feedback and hope you have a productive experience with Visual Studio 2017 RC. ■

Kasey Uhlenhuth *is a program manager on the .NET and Visual Studio team at Microsoft and is currently working on modernizing the C# developer experience. Previously, she worked on C# Interactive and Node.js Tools for Visual Studio. Reach her at kaseyu@microsoft.com or on Twitter: @kuhlenhuth.*

Thanks to the following Microsoft technical experts for reviewing this article: *David Carmona, Kevin Pilch-Bisson and Mark Wilson-Thomas*

Introducing Visual Studio for Mac

Mikayla Hutchinson

At Connect(); in November, Microsoft is launching a preview of Visual Studio for Mac. This is an exciting development, evolving the mobile-centric Xamarin Studio IDE into a true mobile-first, cloud-first development tool for .NET and C#, and bringing the Visual Studio development experience to the Mac.

A New Member of the Visual Studio Family

At its heart, Visual Studio for Mac is a macOS counterpart of the Windows version of Visual Studio. If you enjoy the Visual Studio development experience, but need or want to use macOS, you should feel right at home. Its UX is inspired by Visual Studio, yet designed to look and feel like a native citizen of macOS. And like Visual Studio for Windows, it's complemented by Visual Studio Code for times when you don't need a full IDE, but want a lightweight yet rich standalone source editor.

Below the surface, Visual Studio for Mac also has a lot in common with its siblings in the Visual Studio family. Its IntelliSense and refactoring use the Roslyn Compiler Platform; its project system and build engine use MSBuild; and its source editor supports TextMate bundles. It uses the same debugger engines for Xamarin and .NET Core apps, and the same designers for Xamarin.iOS and Xamarin.Android.

Compatibility is a key focus of Visual Studio for Mac. Although it's a new product and doesn't support all of the Visual Studio project types, for those it does have in common it uses the same MSBuild solution and project format. If you have team members on macOS and Windows, or switch between the two OSes yourself, you can seamlessly share your projects across platforms. There's no need for any conversion or migration.

Mobile-First, Cloud-First Development

The primary workloads supported by Visual Studio for Mac are native iOS, Android and Mac development via Xamarin, and server development via .NET Core with Azure integration. It gives you all the tools you need to develop the rich, native mobile app experiences that users expect today, and the cloud-based server back ends to power them.

It's all powered by the C# language you know and love, with the latest C# 7 productivity enhancements. You get the performance of

This article uses a pre-release version of Visual Studio for Mac. All information is subject to change.

This article discusses:

- A preview of a new member of the Visual Studio family
- Mobile-first, cloud-first C# development with Xamarin and ASP.NET Core
- Core features and capabilities of Visual Studio for Mac
- Developing an ASP.NET Core back end for a Xamarin mobile app

Technologies discussed:

Visual Studio for Mac

compiled code, the productivity of a modern type-safe language, access to the unique features of each platform, and a rich ecosystem of libraries and tools. You can use your existing experience across the mobile and cloud domains, sharing code between client and server. And with all your projects in one solution, you can take advantage of solution-wide cross-project refactoring and code navigation.

C# isn't the only language supported in the Visual Studio for Mac preview. For the functional programmers among you, it includes excellent F# support, powered by the same F# compiler used in Visual Studio.

iOS, Android and Mac

With the fragmented mobile market today it's important to be able to target a wide range of devices. Because it's based on Xamarin Studio, Visual Studio for Mac has mature support for C#-based iOS, Android and Mac development with the Xamarin Platform. You can take advantage of your existing C# experience and libraries, and share common code across platforms, with full access to the native APIs so you can build a fast, polished native app experience.

For even greater code sharing, you can use the cross-platform Xamarin.Forms UI library, which provides a familiar XAML-based development environment that can target multiple platforms, including iOS, Android, macOS and the Universal Windows Platform (UWP)—though UWP development is currently only supported in Visual Studio—and maps to the native UI on each platform. When you need more control, you can mix and match Xamarin.Forms with direct access to the native toolkits. There's a huge ecosystem of libraries available for Xamarin via NuGet, too, including platform-specific libraries, bindings to native code and portable .NET Standard libraries.

Like Visual Studio, Visual Studio for Mac has drag-and-drop designers for iOS and Android development that let you rapidly assemble and fine-tune your UI. For Xamarin.Forms, it has rich XAML IntelliSense and a side-by-side live preview, as **Figure 1** shows. Both the designer and the live preview use a simulator to render your app exactly how it will appear on the device, and this even works for your custom controls.

Cutting-Edge Cloud

Almost every mobile app is backed by a service, and Visual Studio for Mac makes it easy to develop your app's service with its support for the latest ASP.NET Core Web development platform. ASP.NET Core runs on .NET Core, the latest evolution of the .NET Framework and runtime. It's been tuned for blazingly fast performance, factored for small install sizes, and reimagined to run on Linux and macOS, as well as Windows.

.NET Core gives you a huge degree of flexibility in how and where you develop and deploy your server application, whether in your own datacenter or on a cloud platform such as Microsoft Azure. Because both .NET Core and Xamarin Platform are open source, you won't have to worry about vendor lock-in.

The Visual Studio for Mac support for .NET Core projects also allows you to write .NET Standard libraries, the new way to share code across .NET platforms going forward. .NET Standard libraries replace Portable Class Libraries (PCLs) and offer a much broader API surface area. Because .NET Core and Xamarin Platform are .NET Standard-compliant, they're a great way to share code, both within your solution and via the NuGet Package Manager.

A Familiar Workspace

The Visual Studio for Mac workspace should be familiar to existing Visual Studio developers. When you first open it, you see a Welcome Page with a list of recently opened solutions, a feed of developer news and other information to help you get started.

To create a new solution, go to the File menu and select New Project, and you'll see the workspace containing your new solution. As you can see in **Figure 2**, there's a central tabbed source editor with a number of other docked windows or "pads" around it, such as Solution, Output, Properties, Document Outline and Toolbox. Like Visual Studio, this layout is highly customizable and switches automatically, depending on whether you're coding, debugging or using the drag-and-drop designer.

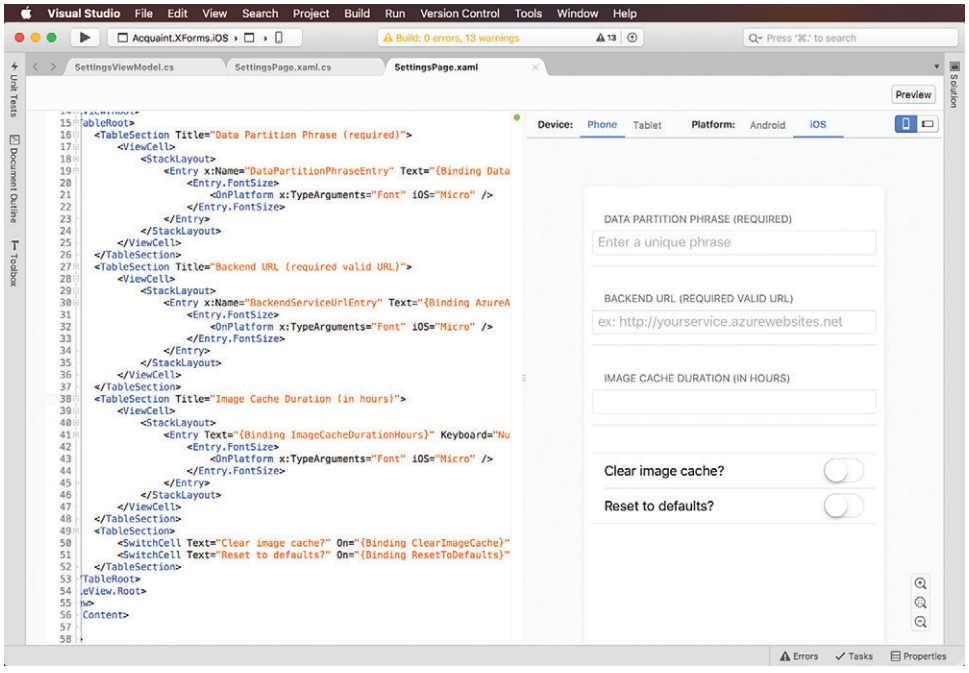


Figure 1 **The Xamarin.Forms XAML Live Preview**

The toolbar is familiar, too, but has a few notable differences:

On the left is the Run button, a dropdown to select the Active Configuration, as well as dropdowns to select the Run Configuration and Target Device. For cross-platform mobile development, it's important to be able to easily switch the device or simulator on which you're testing or debugging your app. The Run Configuration is like the startup

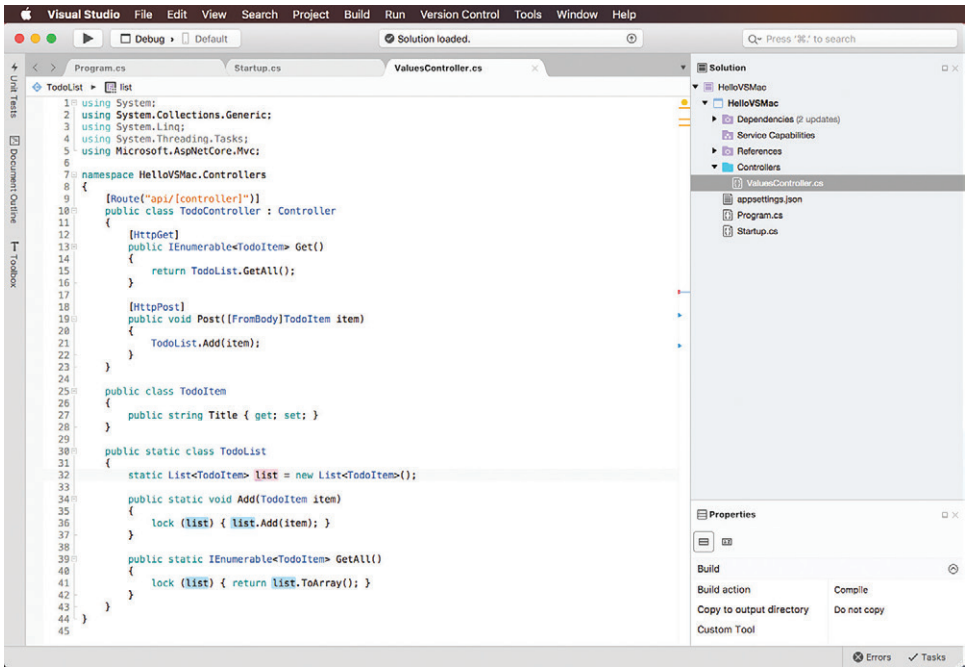


Figure 2 **The Visual Studio for Mac Workspace**

project in Visual Studio, except that in addition to switching which project runs, you can also create custom-named sets of run options.

In the center of the toolbar is a notification area, which shows messages about various operations, such as building or restoring NuGet packages. When there's a running operation, a cancel button shows up in the notification area. This is also where notifications about software updates are displayed. You can click on some notifications, such as build errors, and they'll bring up a pad with more information.

At the right of the toolbar is the global search. In addition to helping you find things like commands and files in your solution, its camelCase filtering system makes it an excellent way to quickly activate commands, or jump to files or types in your solution. It can even kick off a Find in Files search in your solution, or open the NuGet Package Manager to search for a package.

The Solution pad works much the same as the Solution Explorer in Visual Studio, letting you explore and manage the structure of your solution, your project and the files in it. The context menu gives you a range of context-specific commands on the items in the solution tree, such as adding or removing files from projects, editing project references, opening Terminal windows in folders, and building or debugging specific projects.

The Errors pad shows any build warnings and errors, and is also where you can find the build log output in a split view. Unlike Visual Studio, there isn't a single unified pad for all kinds of output. For example, an Application Output pad shows the output from your app when you run or debug it, and logs from NuGet operations are shown in a NuGet Console pad. The Properties pad contextually shows properties of whatever is currently focused and selected, and can be used to view and change the build action of files in the solution pad.

In the center is the heart of the IDE, the source editor, which has all the features you'd expect from a member of the Visual Studio family. **Figure 3** shows C# IntelliSense and syntax highlighting in a .NET Core project. There's also code folding, live underlining of errors and suggestions as you type, configurable automatic formatting, code navigation commands and an array of powerful refactoring tools.

Not all of the editor's functionality is enabled by default. You can tweak the Visual Studio for Mac settings in the Preferences dialog, which is accessible from its Mac application menu. This is equivalent to the Options dialog in the Visual Studio Tools menu, and contains plenty of options to help you customize the IDE to work the way you want.

Unit testing is supported using NUnit, and other test runners can be plugged in via extensions. The tests discovered in your assembly are shown in a Unit Tests pad that can be accessed from the View | Pads menu. There's also git version control integrated right into the source editor, with a row of tabs along the bottom of the editor to access the current file's log, diff and blame view.

If you'd like to get up to speed quickly with some more tips and tricks, I encourage you to watch my "Become a Xamarin Studio Expert" session from Xamarin Evolve 2016 (xmn.io/xs-expert) as its content applies directly to Visual Studio for Mac.

Figure 3 **IntelliSense in a .NET Core Project**

Open Source Core

Like Xamarin Studio, Visual Studio for Mac is based on the open source MonoDevelop IDE, which is actively developed by Microsoft. It's written entirely in C#, and has a rich extensibility model that you can use to add functionality ranging from simple editor commands to entirely new languages

BUSINESS FILE APIS

TRY RISK FREE - 30 Day Trial

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

Aspose.Total

Aspose.Cells
XLS, CSV, PDF, SVG, HTML, PNG
BMP, XPS, JPG, SpreadsheetML...

Aspose.Words
DOC, RTF, PDF, HTML, PNG
ePUB, XML, XPS, JPG...

Aspose.Pdf
PDF, XML, XSL-FO, HTML, BMP
JPG, PNG, ePUB...

Aspose.Slides
PPT, POT, ODP, XPS, HTML
PNG, PDF...

Aspose.BarCode
JPG, PNG, BMP, GIF, TIF, WMF
ICON...

Aspose.Tasks
XML, MPP, SVG, PDF, TIFF
PNG...

Aspose.Email
MSG, EML, PST, MHT
OST, OFT...

Aspose.Imaging
PDF, BMP, JPG, GIF, TIFF
PNG...

+ more!

.NET | Java | Cloud

Download FREE trial at www.aspose.com.

Contact Us:
US: +1 903 306 1676
EU: +44 141 628 8900
AU: +61 2 8006 6987
sales@aspose.com

ASPOSE
File Format APIs

Figure 4 **The Controller and Its Simple Shared To-Do List Storage**

```
[Route("api/[controller]")]
public class ToDoController : Controller
{
  [HttpGet]
  public IEnumerable<ToDoItem> Get()
  {
    return ToDoList.GetAll();
  }

  [HttpPost]
  public void Post([FromBody]ToDoItem item)
  {
    ToDoList.Add(item);
  }
}

public class ToDoItem
{
  public string Title { get; set; }
}

public static class ToDoList
{
  static List<ToDoItem> list = new List<ToDoItem>();

  public static void Add(ToDoItem item)
  {
    lock (list) { list.Add(item); }
  }

  public static IEnumerable<ToDoItem> GetAll()
  {
    lock (list) { return list.ToArray(); }
  }
}
```

and project types. Even core features such as C# editing, Xamarin.iOS, Xamarin.Android and ASP.NET Core are implemented as extensions.

Like Visual Studio and Visual Studio Code, the C# support in Visual Studio for Mac is powered by the open source Roslyn Compiler Platform. You get the exact same IntelliSense experience you're familiar with from Visual Studio, as well as support for in-editor live Analyzers and Code Fixes. Visual Studio for Mac even includes the Refactoring Essentials collection of Analyzers and Code Fixes by default.

Visual Studio for Mac supports editing a wide range of languages though the use of TextMate bundles, which provide syntax highlighting and simple IntelliSense. It includes a number of open source TextMate bundles from Visual Studio Code.

Creating an ASP.NET Core App

To show you how easy it is to get up to speed with Visual Studio for Mac, I'm going to walk though creating a simple ASP.NET Core back end. It's for a hypothetical "Shared To-do List" mobile app, which allows multiple users to add items, and all users see the items that any of them post.

Please note that I'm writing this article using a pre-release version of Visual Studio for Mac, and some details of the UI may change in the release. However, the approaches and concepts discussed in this article will still apply.

After installing and opening Visual Studio for Mac, I start by clicking on the New Solution button on the welcome page, which opens the New Project dialog. I navigate into the Cloud section, choose the ASP.NET Core Web Application template, and click Next, then choose the Web API template. The Web API template creates a RESTful Web service which is perfect for a mobile back end, though you can add views to the project later to create a Web front end.

Finally, I name my project HelloVSMac and click Create. Visual Studio for Mac creates the projects using the dotnet templating engine, opens it and starts restoring the NuGet packages on which it depends. If you open the project file in the editor using the Tools | Edit File context menu on the project in the solution pad, you can see that it's a minimalistic MSBuild-based project file that's intended to be easy to understand. If you edit it directly and save it, the IDE will automatically reload your modified version.

Looking at the project in the solution pad, the key items are:

Packages: Your project's NuGet package dependencies. ASP.NET Core, the .NET Core framework and the MSBuild targets that build the project are all installed via NuGet packages.

Program.cs: The entry point of your Web app. ASP.NET Core apps are programs, so there's a Main method entry point that creates, builds and runs the WebHost at the heart of your app.

Startup.cs: Which defines a Startup class that was passed to the WebHost. This class contains your application's initialization methods.

appsettings.json: Your app's configuration settings. This is the ASP.NET Core equivalent of the ASP.NET web.config.

For the purposes of this walk-through, I'll leave these all as is, and look at the ValuesController.cs file in the Views folder. This

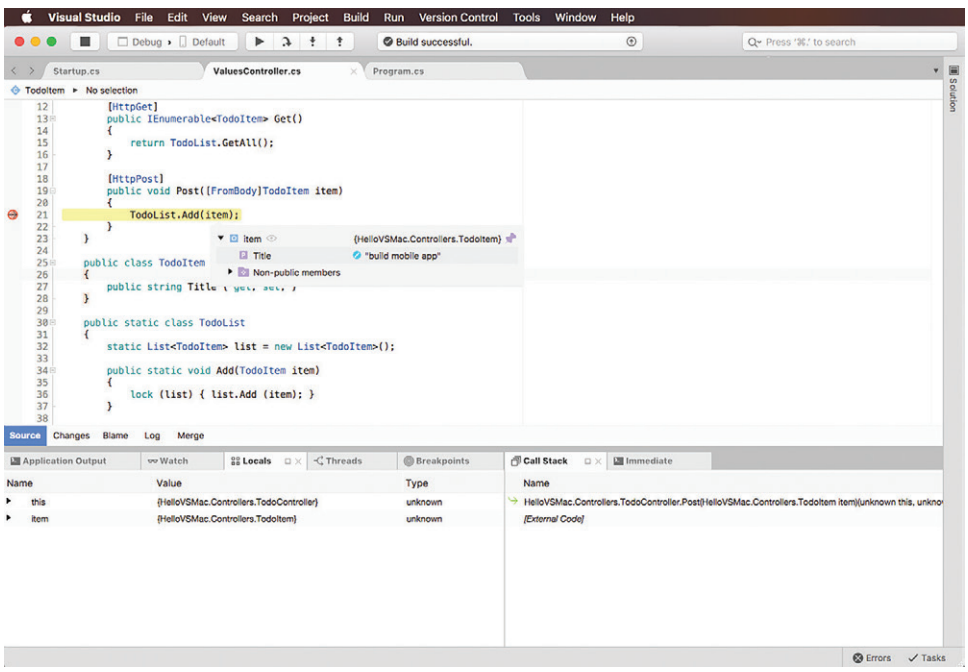


Figure 5 **Debugging a .NET Core Project**

contains a ValuesController class registered on the [Route("api/[controller]")] route. The [controller] is a placeholder for the class name, so this is really the api/values route.

I'll start by defining a very simple ToDoItem class and a ToDoList storage class. ToDoList is static so it can be shared among requests. In a real app you'd use a database for this, but it will do for now. I also rename the controller class to ToDoController (which makes the route api/todo), connect the Get and Post methods to the store, and clear out the other unused controller methods. The result can be seen in **Figure 4**.

This is now a complete, but very small, RESTful Web service. Let's try it out.

I place a breakpoint in the Post method, and start debugging the app. The Output pad starts to show the output from the ASP.NET Core built-in kestrel Web server as the app starts up, by default on port 5000, but it won't do anything else until it receives a request. You can open your Web browser and check 127.0.0.1:5000/api/todo, but it'll just be an empty array.

Because there isn't a mobile client for this service yet, it's time to open the macOS Terminal app and use curl to send a POST request to the app:

```
$ curl -H "Content-type: application/json" -X POST -d '{ title: "build
  mobile app" }' 127.0.0.1:5000/api/todo
```

This triggers the breakpoint in the debugger. You can inspect the value that has automatically been parsed from the JSON body of the request and converted into the ToDoItem object. You can see that Visual Studio for Mac automatically entered the debugging layout, and has all the debugger pads you'd expect: Stack, Locals, Threads, Breakpoints and so on.

Now, go back to the terminal and use curl to access the Get method, and you'll see the JSON array containing the item that was added:

```
$ curl 127.0.0.1:5000/api/todo
[{"title":"build mobile app"}]
```

The next step is to build the mobile app, but I'll let you explore that yourself. For more in-depth information on ASP.NET Core, I recommend checking out asp.net/get-started, and if you'd like to learn more about Xamarin development, there's plenty of great material at developer.xamarin.com. Although there isn't much documentation on Visual Studio for Mac yet, the Xamarin Studio documentation applies directly in most cases, and Visual Studio documentation is often applicable, too.

Wrapping Up

I hope this brief overview has whetted your appetite to try Visual Studio for Mac and make it your macOS IDE of choice for cloud and mobile development! If you have a Mac I encourage you to download the preview from VisualStudio.com, give it a spin, and let us know how you like it. We're excited to hear your feedback to help guide it through the preview and beyond. ■

Mikayla Hutchinson *is a senior program manager on Xamarin Platform at Microsoft. Previously she developed the mobile and Web tooling for Xamarin Studio and was a core developer on MonoDevelop. You can follow her on Twitter: @mjhutchinson.*

Thanks to the following Microsoft technical experts for reviewing this article: *Larry O'Brien and Lluis Sanchez*

Programming C# 7.0

Mark Michaelis

Back in December 2015, I discussed the designing of C# 7.0 (msdn.com/magazine/mt595758). A lot has changed over the last year, but the team is now buttoning down C# 7.0 development, and Visual Studio 15 Preview 5 is expected to implement virtually all of the new features. (I say virtually because until Visual Studio 15 actually ships, there's always a chance for further change.) For a brief overview, you can check out the summary table at itl.tc/CSharp7FeatureSummary. In this article I'm going to explore each of the new features in detail.

Deconstructors

Since C# 1.0, it's been possible to call a function—the constructor—that combines parameters and encapsulates them into a class. However, there's never been a convenient way to deconstruct the object back into its constituent parts. For example, imagine a PathInfo class that takes each element of a filename—directory name, file name, extension—and combines them into an object, with support for then manipulating the object's various elements. Now imagine you wish to extract (deconstruct) the object back into its parts.

In C# 7.0 this becomes trivial via the deconstructor, which returns the specifically identified components of the object. Be careful not to confuse a deconstructor with a destructor (deterministic object deallocation and cleanup) or a finalizer (itl.tc/CSharpFinalizers).

Take a look at the PathInfo class in **Figure 1**.

Obviously, you can call the Deconstruct method as you would have in C# 1.0. However, C# 7.0 provides syntactic sugar that significantly simplifies the invocation. Given the declaration of a deconstructor, you can invoke it using a new C# 7.0 "tuple-like" syntax (see **Figure 2**).

Notice how, for the first time, C# is allowing simultaneous assignment to multiple variables of *different* values. This is *not* the same as the null assigning declaration in which all variables are initialized to the same value (null):

```
string directoryName, filename, extension = null;
```

Instead, with the new tuple-like syntax, each variable is assigned a different value corresponding not to its name, but to the *order* in which it appears in the declaration and the deconstruct statement.

As you'd expect, the type of the out parameters must match the type of the variables being assigned, and var is allowed because the type can be inferred from Deconstruct parameter types. Notice, however, that while you can put a single var outside the parentheses as shown in Example 3 in **Figure 2**, at this time it's not possible to pull out a string, even though all the variables are of the same type.

Note that at this time, the C# 7.0 tuple-like syntax requires that at least two variables appear within the parentheses. For example, (FileInfo path) = pathInfo; is not allowed even if a deconstructor exists for:

```
public void Deconstruct(out FileInfo file)
```

This article uses pre-release versions of C# 7.0 and Visual Studio 15. All information is subject to change.

This article discusses:

- Deconstructors
- Pattern matching with is expressions and with the switch statement
- Local functions
- Return by reference
- More expression-bodied members
- More C# 7.0 features

Technologies discussed:

C# 7.0 and Visual Studio 15 Preview 5

Code download available at:

bit.ly/2dip18Y

Figure 1 **PathInfo Class with a Deconstructor with Associated Tests**

```
public class PathInfo
{
  public string DirectoryName { get; }
  public string FileName { get; }
  public string Extension { get; }
  public string Path
  {
    get
    {
      return System.IO.Path.Combine(
        DirectoryName, FileName, Extension);
    }
  }

  public PathInfo(string path)
  {
    DirectoryName = System.IO.Path.GetDirectoryName(path);
    FileName = System.IO.Path.GetFileNameWithoutExtension(path);
    Extension = System.IO.Path.GetExtension(path);
  }

  public void Deconstruct(
    out string directoryName, out string fileName, out string extension)
  {
    directoryName = DirectoryName;
    fileName = FileName;
    extension = Extension;
  }

  // ...
}
```

In other words, you can't use the C# 7.0 deconstructor syntax for Deconstruct methods with only one out parameter.

Working with Tuples

As I mentioned, each of the preceding examples leveraged the C# 7.0 tuple-like syntax. The syntax is characterized by the parentheses that surround the multiple variables (or properties) that are assigned. I use the term "tuple-like" because, in fact, none of these deconstructor examples actually leverage any tuple type internally. (In fact, assignment of tuples via a deconstructor syntax isn't allowed and arguably would be somewhat unnecessary because the object assigned already is an instance representing the encapsulated constituent parts.)

With C# 7.0 there's now a special streamlined syntax for working with tuples, as shown in **Figure 3**. This syntax can be used whenever a type specifier is allowed, including declarations, cast operators and type parameters.

For those not familiar with tuples, it's a way of combining multiple types into a single containing type in a lightweight syntax that's available outside of the method in which it's instantiated. It's lightweight because, unlike defining a class/struct, tuples can be "declared" inline and on the fly. But, unlike dynamic types that also support inline declaration and instantiation, tuples can be accessed outside of their containing member and, in fact, they can be included as part of an API. In spite of the external API support, tuples don't have any means of version-compatible extension (unless the type parameters themselves happen to support derivation), thus, they should be used with caution in public APIs. Therefore, a preferable approach might be to use a standard class for the return on a public API.

Prior to C# 7.0, the framework already had a tuple class, System.Tuple<…> (introduced in the Microsoft .NET Framework 4).

Figure 2 **Deconstructor Invocation and Assignment**

```
PathInfo pathInfo = new PathInfo(@"\\test\unc\path\to\something.ext");

{
  // Example 1: Deconstructing declaration and assignment.
  (string directoryName, string fileName, string extension) = pathInfo;
  VerifyExpectedValue(directoryName, fileName, extension);
}

{
  string directoryName, fileName, extension = null;
  // Example 2: Deconstructing assignment.
  (directoryName, fileName, extension) = pathInfo;
  VerifyExpectedValue(directoryName, fileName, extension);
}

{
  // Example 3: Deconstructing declaration and assignment with var.
  var (directoryName, fileName, extension) = pathInfo;
  VerifyExpectedValue(directoryName, fileName, extension);
}
```

Figure 3 **Declaring, Instantiating and Using the C# 7.0 Tuple Syntax**

```
[TestMethod]
public void Constructor_CreateTuple()
{
  (string DirectoryName, string FileName, string Extension) pathData =
    (DirectoryName: @"\\test\unc\path\to",
    FileName: "something",
    Extension: ".ext");

  Assert.AreEqual<string>(
    @"\\test\unc\path\to", pathData.DirectoryName);
  Assert.AreEqual<string>(
    "something", pathData.FileName);
  Assert.AreEqual<string>(
    ".ext", pathData.Extension);

  Assert.AreEqual<(string DirectoryName, string FileName, string Extension)>(
    (DirectoryName: @"\\test\unc\path\to",
      FileName: "something", Extension: ".ext"),
    (pathData));

  Assert.AreEqual<(string DirectoryName, string FileName, string Extension)>(
    (@"\\test\unc\path\to", "something", ".ext"),
    (pathData));

  Assert.AreEqual<(string, string, string)>(
    (@"\\test\unc\path\to", "something", ".ext"), (pathData));

  Assert.AreEqual<Type>(
    typeof(ValueTuple<string, string, string>), pathData.GetType());
}

[TestMethod]
public void ValueTuple_GivenNamedTuple_ItemXHasSameValuesAsNames()
{
  var normalizedPath =
    (DirectoryName: @"\\test\unc\path\to", FileName: "something",
    Extension: ".ext");

  Assert.AreEqual<string>(normalizedPath.Item1, normalizedPath.DirectoryName);
  Assert.AreEqual<string>(normalizedPath.Item2, normalizedPath.FileName);
  Assert.AreEqual<string>(normalizedPath.Item3, normalizedPath.Extension);
}

static public (string DirectoryName, string FileName, string Extension)
  SplitPath(string path)
{
  // See http://bit.ly/2dmJIMm Normalize method for full implementation.

  return (
    System.IO.Path.GetDirectoryName(path),
    System.IO.Path.GetFileNameWithoutExtension(path),
    System.IO.Path.GetExtension(path)
    );
}
```

C# 7.0 differs from the earlier solution, however, because it embeds the semantic intent into declaration and it introduces a tuple value type: System.ValueTuple<…>.

Let's take a look at the semantic intent. Notice in **Figure 3** that the C# 7.0 tuple syntax allows you to declare alias names for each ItemX element the tuple contains. The pathData tuple instance in **Figure 3**, for example, has strongly typed DirectoryName: string, FileName: string, and Extension: string properties defined, thus allowing calls to pathData.DirectoryName, for example. This is a significant enhancement because prior to C# 7.0, the only names available were ItemX names, where the X incremented for each element.

Now, while the elements for a C# 7.0 tuple are strongly typed, the names themselves aren't distinguishing in the type definition. Therefore, you can assign two tuples with disparate name aliases and all you'll get is a warning that informs you the name on the right-hand side will be ignored:

```
// Warning: The tuple element name 'AltDirectoryName1' is ignored
// because a different name is specified by the target type...
(string DirectoryName, string FileName, string Extension) pathData =
  (AltDirectoryName1: @"\\test\unc\path\to",
  FileName: "something", Extension: ".ext");
```

Similarly, you can assign tuples to other tuples that may not have all alias element names defined:

```
// Warning: The tuple element name 'directoryName', 'FileNAme' and 'Extension'
// are ignored because a different name is specified by the target type...
(string, string, string) pathData =
  (DirectoryName: @"\\test\unc\path\to", FileName: "something", Extension: ".ext");
```

To be clear, the type and order of each element does define type compatibility. Only the element names are ignored. However, even though ignored when they're different, they still provide IntelliSense within the IDE.

Note that, whether or not an element name alias is defined, all tuples have ItemX names where X corresponds to the number of the element. The ItemX names are important because they make the tuples available from C# 6.0, even though the alias element names are not.

Another important point to be aware of is that the underlying C# 7.0 tuple type is a System.ValueTuple. If no such type is available in the framework version you're compiling against, you can access it via a NuGet package.

For details about the internals of tuples, check see intellitect.com/csharp7tupleiinternals.

Figure 4 **Type Casting Without Pattern Matching**

```
// Eject without pattern matching.
public void Eject(Storage storage)
{
  if (storage == null)
  {
    throw new ArgumentNullException();
  }
  if (storage is UsbKey)
  {
    UsbKey usbKey = (UsbKey)storage;
    if (usbKey.IsPluggedIn)
    {
      usbKey.Unload();
      Console.WriteLine("USB Drive Unloaded.");
    }
    else throw new NotImplementedException();    }
  else if(storage is DVD)
  // ...
  else throw new NotImplementedException();
}
```

Pattern Matching with Is Expressions

On occasion you have a base class, Storage for example, and a series of derived classes, DVD, UsbKey, HardDrive, FloppyDrive (remember those?) and so on. To implement an Eject method for each you have several options:

- As Operator
  - Cast and assign using the as operator
  - Check the result for null
  - Perform the eject operation
- Is Operator
  - Check the type using the is operator
  - Cast and assign the type
  - Perform the eject operation
- Cast
  - Explicit cast and assign
  - Catch possible exception
  - Perform operation
  - Yuck!

There's a fourth, far-better approach using polymorphism in which you dispatch using virtual functions. However, this is available only if you have the source code for the Storage class and can add the Eject method. That's an option I'm assuming is unavailable for this discussion, hence the need for pattern matching.

The problem with each of these approaches is that the syntax is fairly verbose and always requires multiple statements for each class to which you want to cast. C# 7.0 provides pattern matching as a means of combining the test and the assignment into a single operation. As a result, the code in **Figure 4** simplifies down to what's shown in **Figure 5**.

The difference between the two isn't anything radical, but when performed frequently (for each of the derived types, for example) the former syntax is a burdensome C# idiosyncrasy. The C# 7.0 improvement—combining the type test, declaration and assignment into a single operation—renders the earlier syntax all but deprecated. In the former syntax, checking the type without assigning an identifier makes falling through to the "default" else cumbersome at best. In contrast, the assignment allows for the additional conditionals beyond just the type check.

Note that the code in **Figure 5** starts out with a pattern-matching is operator with support for a null comparison operator, as well:

```
if (storage is null) { ... }
```

Pattern Matching with the Switch Statement

While supporting pattern matching with the is operator provides an improvement, pattern-matching support for a switch statement is arguably even more significant, at least when there are multiple compatible types to which to convert. This is because C# 7.0 includes case statements with pattern matching and, furthermore, if the type pattern is satisfied in the case statement, an identifier can be provided, assigned, and accessed all within the case statement. **Figure 6** provides an example.

Notice in the example how local variables like usbKey and dvd are declared and assigned automatically within the case statement. And, as you'd expect, the scope is limited to within the case statement.

Perhaps just as important as the variable declaration and assignment, however, is the additional conditional that can be appended

Get Connected Today.
TAKE A FREE TEST DRIVE!
www.MelissaData.com/kyc

Know Your Customer

Customers share tons of personal data through an increasing number of channels and applications. We supply the full spectrum of data quality solutions to solve your KYC challenges – to collect, verify, enrich and consolidate clean contact data for a true and complete view of the customer. Microsoft®, Oracle®, Pentaho®, Salesforce® and more.

Solutions for 240+ Countries | **10,000+ Customers Worldwide** | **30+ Years Strong** (1985) | **Data Quality & Mailing Solutions** | **Cloud • On-Premise • Services**

Germany
www.MelissaData.de

United Kingdom
www.MelissaData.co.uk

India
www.MelissaData.in

Australia
www.MelissaData.com.au

MELISSA DATA®
Your Partner in Global Data Quality

www.MelissaData.com | 1-800-MELISSA

Figure 5 **Type Casting with Pattern Matching**

```
// Eject with pattern matching.
public void Eject(Storage storage)
{
  if (storage is null)
  {
    throw new ArgumentNullException();
  }
  if (((storage is UsbKey usbDrive) && usbDrive.IsPluggedIn)
  {
    usbKey.Unload();
    Console.WriteLine("USB Drive Unloaded.");
  }
  else if (storage is DVD dvd && dvd.IsInserted)
  // ...
  else throw new NotImplementedException();  // Default
}
```

Figure 6 **Pattern Matching in a Switch Statement**

```
public void Eject(Storage storage)
{
  switch(storage)
  {
    case UsbKey usbKey when usbKey.IsPluggedIn:
      usbKey.Unload();
      Console.WriteLine("USB Drive Unloaded.");
      break;
    case DVD dvd when dvd.IsInserted:
      dvd.Eject();
      break;
    case HardDrive hardDrive:
      throw new InvalidOperationException();
    case null:
    default:
      throw new ArgumentNullException();
  }
}
```

Figure 7 **A Local Function Example**

```
bool IsPalindrome(string text)
{
  if (string.IsNullOrWhiteSpace(text)) return false;

  bool LocalIsPalindrome(string target)
  {
    target = target.Trim();  // Start by removing any surrounding whitespace.
    if (target.Length <= 1) return true;
    else
    {
      return char.ToLower(target[0]) ==
        char.ToLower(target[target.Length - 1]) &&
        LocalIsPalindrome(
          target.Substring(1, target.Length - 2));
    }
  }
  return LocalIsPalindrome(text);
}
```

Figure 8 **Unit Testing Often Uses Local Functions**

```
[TestMethod]
public void IsPalindrome_GivenPalindrome_ReturnsTrue()
{
  void AssertIsPalindrome(string text)
  {
    Assert.IsTrue(IsPalindrome(text),
      $"'{text}' was not a Palindrome.");
  }
  AssertIsPalindrome("7");
  AssertIsPalindrome("4X4");
  AssertIsPalindrome("   tnt");
  AssertIsPalindrome("Was it a car or a cat I saw");
  AssertIsPalindrome("Never odd or even");
}
```

to the case statement with a when clause. The result is that a case statement can completely filter out an invalid scenario without an additional filter inside the case statement. This has the added advantage of allowing evaluation of the next case statement if, in fact, the former case statement is not fully met. It also means that case statements are no longer limited to constants and, furthermore, a switch expression can be any type—it's no longer limited to bool, char, string, integral and enum.

Another important characteristic the new C# 7.0 pattern-matching switch statement capability introduces is that case statement order is significant and validated at compile time. (This is in contrast with earlier versions of the language, in which, without pattern matching, case statement order was not significant.) For example, if I introduced a case statement for Storage prior to a pattern-matching case statement that derives from Storage (UsbKey, DVD and HardDrive), then the case Storage would eclipse all other type pattern matching (that derives from Storage). A case statement from a base type that eclipses other derived type case statements from evaluation will result in a compile error on the eclipsed case statement. In this way, case statement order requirements are similar to catch statements.

Readers will recall that an is operator on a null value will return false. Therefore, no type pattern-matching case statement will match for a null-valued switch expression. For this reason, order of the null case statement won't matter; this behavior matches switch statements prior to pattern matching.

Also, in support of compatibility with switch statements prior to C# 7.0, the default case is always evaluated last regardless of where it appears in the case statement order. (That said, readability would generally favor putting it at the end, because it's always evaluated last.) Also, goto case statements still work only for constant case labels—not for pattern matching.

Local Functions

While it's already possible to declare a delegate and assign it an expression, C# 7.0 takes this one step further by allowing the full declaration of a local function inline within another member. Consider the IsPalindrome function in **Figure 7**.

In this implementation, I first check that the argument passed to IsPalindrome isn't null or only whitespace. (I could've used pattern matching with "text is null" for the null check.) Next, I declare a function LocalIsPalindrome in which I compare the first and last characters recursively. The advantage of this approach is that I don't declare the LocalIsPalindrome within the scope of the class where it can potentially be called mistakenly, thus circumventing the IsNullOrWhiteSpace check. In other words, local functions provide an additional scope restriction, but only inside the surrounding function.

The parameter validation scenario in **Figure 7** is one of the common local function use cases. Another one I encounter frequently occurs within unit tests, such as when testing the IsPalindrome function (see **Figure 8**).

Iterator functions that return IEnumerable<T> and yield return elements are another common local function use case.

To wrap up the topic, here are a few more points to be aware of for local functions:

Enterprise-Proven Distributed Caching

Trusted for over a decade, the easiest most powerful in-memory data grid to scale your .NET applications

- **Fast and linearly scalable**
- **Enterprise-grade availability**
- **Industry-leading ease of use**
- **Integrated in-memory computing**

Replacing AppFabric Caching?
Try our source-code compatible drop-in.
www.scaleoutsoftware.com/appfabric

Brought to you by the scalability architects

Step up to a battle-tested in-memory data grid that has been hardened by over 425 enterprise customer deployments. ScaleOut's technology makes advanced features such as parallel LINQ query and integrated MapReduce accessible to any .NET developer. Automatic configuration and turnkey global data replication deliver legendary ease-of-use. ScaleOut's world-class support meets the needs of mission-critical applications. Run on premises or in the cloud on Microsoft Azure or Amazon AWS.

ScaleOut Software

Download your free trial today!
www.scaleoutsoftware.com/trial

Please visit scaleoutsoftware.com to learn how ScaleOut's technology can help accelerate your business or call us at +1.503.643.3422

Figure 9 **Ref Return and Ref Local Declaration**

```
public ref byte FindFirstRedEyePixel(byte[] image)
{

  //// Do fancy image detection perhaps with machine learning.
  for (int counter = 0; counter < image.Length; counter++)
  {
    if(image[counter] == (byte)ConsoleColor.Red)
    {
      return ref image[counter];
    }
  }
  throw new InvalidOperationException("No pixels are red.");
}

[TestMethod]
public void FindFirstRedEyePixel_GivenRedPixels_ReturnFirst()
{
  byte[] image;
  // Load image.
  // ...

    // Obtain a reference to the first red pixel.
  ref byte redPixel = ref FindFirstRedEyePixel(image);
  // Update it to be Black.
  redPixel = (byte)ConsoleColor.Black;
  Assert.AreEqual<byte>((byte)ConsoleColor.Black, image[redItems[0]]);
}
```

- Local functions don't allow use of an accessibility modifier (public, private, protected).
- Local functions don't support overloading. You can't have two local functions in the same method with the same name even if the signatures don't overlap.
- The compiler will issue a warning for local functions that are never invoked.
- Local functions can access all variables in the enclosing scope, including local variables. This behavior is the same with locally defined lambda expressions except that local functions don't allocate an object that represents the closure, as locally defined lambda expressions do.
- Local functions are in scope for the entire method, regardless of whether they're invoked before or after their declaration.

Return by Reference

Since C# 1.0 it has been possible to pass arguments into a function by reference (ref). The result is that any change to the parameter itself will get passed back to the caller. Consider the following Swap function:

```
static void Swap(ref string x, ref string y)
```

In this scenario, the called method can update the original caller's variables with new values, thereby swapping what's stored in the first and second arguments.

Starting in C# 7.0, you're also able to pass back a reference via the function return—not just a ref parameter. Consider, for example, a function that returns the first pixel in an image that's associated with red-eye, as shown in **Figure 9**.

By returning a reference to the image, the caller is then able to update the pixel to a different color. Checking for the update via the array shows that the value is now black. The alternative of using a by reference parameter is, one might argue, less obvious and less readable:

```
public bool FindFirstRedEyePixel(ref byte pixel);
```

There are two important restrictions on return by reference—both due to object lifetime. These are that object references shouldn't be garbage collected while they're still referenced, and they shouldn't consume memory when they no longer have any references. First, you can only return references to fields, other reference-returning properties or functions, or objects that were passed in as parameters to the by reference-returning function. For example, FindFirstRedEyePixel returns a reference to an item in the image array, which was a parameter to the function. Similarly, if the image was stored as a field within a class, you could return the field by reference:

```
byte[] _Image;
public ref byte[] Image { get {  return ref _Image; } }
```

Second, ref locals are initialized to a certain storage location in memory, and can't be modified to point to a different location. (You can't have a pointer to a reference and modify the reference—a pointer to a pointer for those of you with a C++ background.)

There are several return-by-reference characteristics of which to be cognizant:

- If you're returning a reference you obviously have to return it. This means, therefore, that in the example in **Figure 9**, even if no red-eye pixel exists, you still need to return a ref byte. The only workaround would be to throw an exception. In contrast, the by reference parameter approach allows you to leave the parameter unchanged and return a bool indicating success. In many cases, this might be preferable.
- When declaring a reference local variable, initialization is required. This involves assigning it a ref return from a function or a reference to a variable:

```
ref string text;  // Error
```

- Although it's possible in C# 7.0 to declare a reference local variable, declaring a field of type ref isn't allowed:

```
class Thing { ref string _Text;  /* Error */ }
```

- You can't declare a by reference type for an auto-implemented property:

```
class Thing { ref string Text { get;set; }  /* Error */ }
```

- Properties that return a reference are allowed:

```
class Thing { string _Text = "Inigo Montoya";
  ref string Text { get { return ref _Text; } } }
```

- A reference local variable can't be initialized with a value (such as null or a constant). It must be assigned from a by reference returning member or a local variable/field:

```
ref int number = null; ref int number = 42;  // ERROR
```

Out Variables

Since the first release of C#, the invocation of methods containing out parameters always required the pre-declaration of the out argument

Figure 10 **Inline Declaration of Out Arguments**

```
public long DivideWithRemainder(
  long numerator, long denominator, out long remainder)
{
  remainder = numerator % denominator;
  return (numerator / denominator);
}


[TestMethod]
public void DivideTest()
{
  Assert.AreEqual<long>(21,
    DivideWithRemainder(42, 2, out long remainder));
  Assert.AreEqual<long>(0, remainder);
}
```

identifier before invocation of the method. C# 7.0 removes this idiosyncrasy, however, and allows the declaration of the out argument inline with the method invocation. **Figure 10** shows an example.

Notice how in the DivideTest method, the call to DivideWithRemainder from within the test includes a type specifier after the out modifier. Furthermore, see how remainder continues to be in scope of the method automatically, as evidenced by the second Assert.AreEqual invocation. Nice!

Literal improvements

Unlike previous versions, C# 7.0 includes a numeric binary literal format, as the following example demonstrates:

```
long LargestSquareNumberUsingAllDigits =
  0b0010_0100_1000_1111_0110_1101_1100_0010_0100;  // 9,814,072,356
long MaxInt64 { get; } =
  9_223_372_036_854_775_807;  // Equivalent to long.MaxValue
```

Notice also the support for the underscore "_" as a digit separator. It's used simply to improve readability and can be placed anywhere between the digits of the number—binary, decimal or hexadecimal.

Generalized Async Return Types

On occasion when implementing an async method, you're able to return the result synchronously, short-circuiting a long-running operation because the result is virtually instantaneous or even already known. Consider, for example, an async method that determines the total size of files within a directory (bit.ly/2dExeDG). If, in fact, there are no files in the directory, the method can return immediately without ever executing a long-running operation. Until C# 7.0, the requirements of async syntax dictated that the return from such a method should be a Task<long> and, therefore, that a Task be instantiated even if no such Task instance is required. (To achieve this, the general pattern is to return the result from Task.FromResult<T>.)

In C# 7.0, the compiler no longer limits async method returns to void, Task or Task<T>. You can now define custom types, such as the .NET Core Framework-provided System.Threading.Tasks.ValueTask<T> struct, which are compatible with an async method return. See itl.tc/GeneralizedAsyncReturnTypes for more information.

More Expression-Bodied Members

C# 6.0 introduced expression-bodied members for functions and properties, enabling a streamlined syntax for implementing trivial methods and properties. In C# 7.0, expression-bodied implementations are added to constructors, accessors (get and set property implementations) and even finalizers (see **Figure 11**).

Figure 11 **Using Expression-Bodied Members in Accessors and Constructors**

```
class TemporaryFile  // Full IDisposible implementation
                     // left off for elucidation.
{
  public TemporaryFile(string fileName) =>
    File = new FileInfo(fileName);

  ~TemporaryFile() => Dispose();

  Fileinfo _File;
  public FileInfo File
  {
    get => _File;
    private set => _File = value;
  }

  void Dispose() => File?.Delete();
}
```

I expect the use of expression-bodied members to be particularly common for finalizers because the most common implementation is to call the Dispose method, as shown.

I'm pleased to point out that the additional support for expression-bodied members was implemented by the C# community rather than the Microsoft C# team. Yay for open source!

Caution: This feature is not implemented in Visual Studio 15 Preview 5.

Throw Expressions

The Temporary class in **Figure 11** can be enhanced to include parameter validation within the expression-bodied members; therefore, I can update the constructor to be:

```
public TemporaryFile(string fileName) =>
  File = new FileInfo(filename ?? throw new ArgumentNullException());
```

Without throw expressions, C# support for expression-bodied members couldn't do any parameter validation. However, with C# 7.0 support for throw as an expression, not just a statement, the reporting of errors inline within larger containing expressions becomes possible.

Caution: This feature is not implemented in Visual Studio 15 Preview 5.

Wrapping Up

I confess that when I started writing this article, I expected it to be much shorter. However, as I spent more time programming and testing the features, I discovered there was way more to C# 7.0 than I realized from reading the feature titles and following the language development. In many cases—declaring out variables, binary literals, throw expressions and such—there isn't much involved in understanding and using the features. However, several cases—return by reference, deconstructors and tuples, for example—require much more to learn the feature than one might expect initially. In these latter cases, it isn't just the syntax, but also knowing when the feature is relevant.

C# 7.0 continues to whittle away at the quickly decreasing list of idiosyncrasies (pre-declared out identifiers and lack of throw expressions), while at the same time broadening to include support for features previously not seen at the language level (tuples and pattern matching).

Hopefully, this introduction helps you jump into programming C# 7.0 immediately. For more information on C# 7.0 developments following this writing, check out my blog at intellitect.com/csharp7, as well as an update to my book, "Essential C# 7.0" (which is expected to come out shortly after Visual Studio 15 is released to manufacturing). ■

Mark Michaelis *is founder of IntelliTect, where he serves as its chief technical architect and trainer. For nearly two decades he has been a Microsoft MVP and a Microsoft Regional Director since 2007. Michaelis serves on several Microsoft software design review teams, including C#, Microsoft Azure, SharePoint and Visual Studio ALM. He speaks at developer conferences and has written numerous books including his most recent, "Essential C# 6.0 (5th Edition)" (itl.tc/EssentialCSharp). Contact him on Facebook at facebook.com/Mark.Michaelis, on his blog at IntelliTect.com/Mark, on Twitter: @markmichaelis or via e-mail at mark@IntelliTect.com.*

Thanks to the following technical experts for reviewing this article: *Kevin Bost (IntelliTect), Mads Torgersen (Microsoft) and Bill Wagner (Microsoft)*

Increase App Engagement with Xamarin and the Universal Windows Platform

Tyler Whitney

If you use Xamarin to target iOS and Android, you know about the benefits of developing for multiple platforms using one programming language and shared code. You also know about the benefits of using Xamarin.Forms to reuse your UI across Android and iOS while retaining the freedom to take advantage of unique platform capabilities.

What you might not be aware of is that you can use these same tools to target the Universal Windows Platform (UWP), reach more than 400 million devices, and increase downloads and engagement across iOS and Android devices at the same time.

Why Target the UWP?

Why should you target your app for the UWP? App engagement—that's why. As developers, we want reach. We want eyes on our content. We want our apps to be used—and used regularly—to make the most of in-app purchases, advertising, content contributions to build the app's ecosystem and so on.

This article discusses:

- How the Universal Windows Platform (UWP) extends app reach and engagement
- What the UWP offers to iOS and Android developers
- Adding a UWP project to an existing Xamarin.Forms app

Technologies discussed:

Xamarin, Universal Windows Platform

The UWP allows you to write apps that run on desktops, tablets, phones, Xbox, HoloLens and Internet of Things (IoT) devices. That's more than 400 million potential sockets for your app. But it isn't just about the number of Windows-based devices. It's also about boosting engagement across all the devices you support. Consider NPR's experience with its app, NPR One (one.npr.org).

NPR One delivers a curated stream of public radio news, stories and podcasts to its users. Targeting Windows dramatically increased engagement with NPR's content. A Microsoft blog post (bit.ly/2e30plQ) quoted Ben Schein, product manager at NPR, as saying, "Seventy percent of NPR listeners use Windows devices, and we've seen a 50 percent increase in their listening time just since Windows 10 came out. And that was before we upgraded our app." As an unexpected bonus, NPR also found when "... we released the Windows app, we had an unanticipated spike in iOS and Android downloads, as well."

This speaks to the fact that we live in a multi-device world. We know people have iOS and Android devices, and we know they have PCs at work or at home. It makes sense that they switch between these devices during the day depending on where they are and what they're doing. Having your app available on all the devices people use makes it more likely they'll stay engaged with your app and your content.

Consider a typical day where someone rides the bus to work. On the bus, she starts working on a task on her phone, such as reading a report. Once she gets to work, she might want to pick that task up on a desktop by exporting the report to Microsoft Excel for more

Figure 1 **Adaptive UI**

analysis. On the bus ride home, she might start watching a video. When she arrives home, she might transition to a PC—or to a large-screen living room experience with an Xbox—and want to pick up watching where she left off.

In each of these cases, reach isn't simply about the devices your app can run on, though that's clearly important. Reach is also about how well your app lets users transition between devices as they try to accomplish their tasks on the device that makes the most sense at the time.

What Is the UWP?

The UWP provides a common app platform that's available on every device that runs Windows 10.

In addition to a common set of APIs available on all Windows 10-based devices, device-specific APIs allow you to make the most of a device. I'll now discuss some of the main benefits the UWP provides to developers.

Adaptive and Beautiful UI The UWP is designed to help your app adapt to different screen sizes and types of input. It provides UI controls and layout controls that adapt to the device on which they're running whether it's a small phone screen or a large entertainment center screen.

Figure 1 shows an example of adaptive UI. Notice the location of the call button and picture-in-picture control on the mobile device. When the app runs on the phone, the size of the call button changes and its position adjusts, making the UI easier to use with one hand. The location of the picture-in-picture is adjusted on the phone to accommodate the smaller screen.

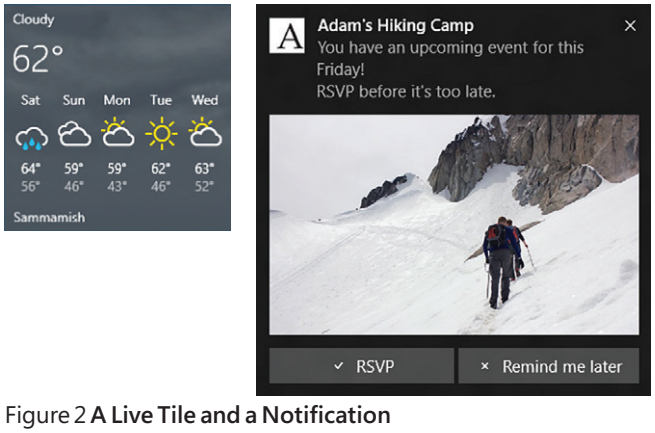


Figure 2 **A Live Tile and a Notification**

The built-in controls accept the type of inputs available on the device, whether they're touch input, pen, keyboard, mouse or Xbox controller.

You can write beautiful apps with new composition APIs. You can also create animations (including keyframe animations) and apply effects such as drop shadows, thumbnail lighting, blurs, opacity, scaling animation effects, hue rotation, parallaxing ListView items, Z-order scrolling, sepia, contrast and more. See these effects for yourself by cloning and running the WindowsUIDevLabs project on GitHub (bit.ly/2e3PDqo).

UWP apps can provide Live Tiles and notifications, as shown in **Figure 2**, to present at-a-glance information for your app. Live Tiles and notifications can boost app engagement by up to 20 percent because they bring users back to your app.

Adaptive Code Takes Advantage of the Strengths of the Device The UWP lets you tailor the experience to take advantage of the strengths of various device types. You can write adaptive code that takes advantage of the capabilities of a specific device only when the app is running on that type of device, or target your app to a specific type of device. Visual Studio filters the available APIs to those associated with the device category you target. The Windows Store scopes available apps to the type of device being used. UWP apps are available on all devices.

Cortana APIs provide the ability to add voice commands to your app. Register app actions on the Cortana portal and Cortana will suggest actions that involve your app at the right time and place.

To help you write engaging apps for the UWP, Visual Studio provides exceptional coding and debugging tooling. Resources are available to help you write your app, such as code samples (bit.ly/1RhG46l), task snippets (bit.ly/2dINSo9), a vibrant community of developers from which you can get help, and libraries such as the UWP Community Toolkit (bit.ly/2b1PAJY), which provides animations, custom controls, and services for Twitter and Facebook. You can then produce one package that can be installed on all UWP-based devices.

Get Your App into Customer Hands with Less Friction The Windows Store reduces your cost per install by making it easier to distribute to a wider audience. It handles sales across geographic boundaries, which frees you from having to understand the banking infrastructure and tax laws in other countries and reduces the friction of reaching regions outside of your own. The Store also handles licensing for apps that share content between devices so you don't have to build out infrastructure to handle those concerns.

The Store provides the experiences to which you're accustomed from an app store, such as automatic updates, licensing, trials and so on. It casts a large net by making your app available to hundreds of millions of Windows 10 users.

The UWP provides the functionality to create rich apps, run them across a wide variety of devices and make them available to your customers. As you'll see, it isn't difficult to target the UWP from Xamarin.

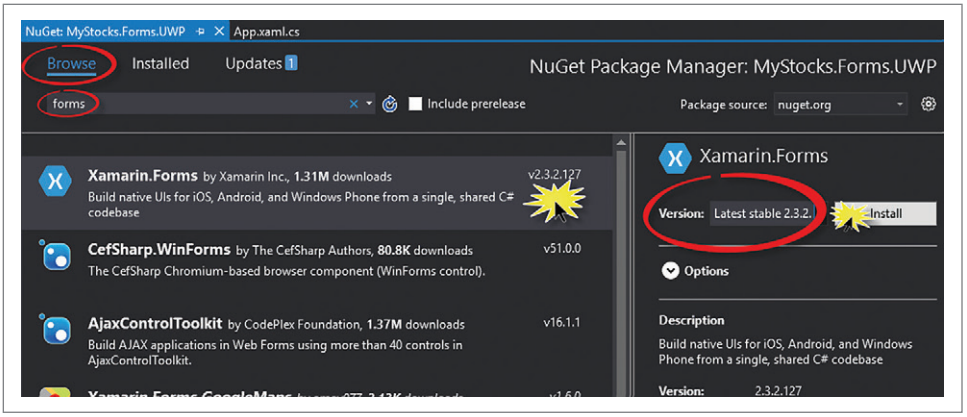


Figure 3 **The NuGet Package Manager**

Add a UWP Project to Your Existing Xamarin.Forms App

I'll show you how a UWP project was added to James Montemagno's stock ticker code sample on GitHub (bit.ly/2dYHEvs), which is a Xamarin.Forms solution that targets iOS, Android and the UWP.

I'm going to assume that you have Visual Studio 2015 Update 3 running on Windows 10 and that Xamarin is installed and up-to- date.

If you want to follow along with the steps, clone the code sample, remove the MyStocks.Forms.UWP project and add it back with the following steps:

1. Add a UWP project to the existing Xamarin.Forms solution.
2. Add a reference from the UWP project to Xamarin.Forms.
3. Add a reference to the Portable Class Library (PCL) that contains the shared forms.
4. Modify the code in the UWP project to use Xamarin.Forms, and load the app from the shared PCL project.

Let's work through these one step at a time.

Add a Blank UWP Project to Your Existing Xamarin.Forms Solution Once you've loaded the MyStocks.Forms solution into Visual Studio, right-click the solution node and choose Add | New Project. In the Add New Project dialog, navigate to Visual C# | Windows | Universal and select Blank App (Universal Windows). Name the project MyStocks.Forms.UWP and click OK.

Next, you'll see the New Universal Windows Project dialog, which asks you to choose the minimum platform versions that your UWP app will support. Click OK to select the defaults.

Now that you've added a UWP project to your Xamarin solution, right-click the new UWP project's References node and select Manage NuGet Packages.

When the NuGet package window appears, select Browse, type "forms" into the search box to narrow the list and then select Xamarin.Forms from the list. Select the latest stable version from the dropdown on the right, note which version it is and click Install (see **Figure 3**).

Ensure Your Projects Target the Same Version If you have other Xamarin.Forms projects in your solution, ensure they're using the same version of Xamarin.Forms that you added to your UWP project. To see what version of Xamarin.Forms the other projects are using, select a project (for example, MyStocks.Forms.Android), right-click its References node and choose Manage NuGet Packages. Ensure Installed is selected, type "forms" into the search box to narrow the list and then find Xamarin.Forms in the list of installed NuGet packages. Verify that the version matches the version of Xamarin.Forms you're using in your MyStocks.UWP project. Update it if it's an earlier version (see **Figure 4**).

Add a Reference to the PCL that Contains the Shared Forms You want the UI for the UWP project to use the shared UI in the Xamarin.Forms project. To accomplish that, the new UWP project needs to reference the PCL that contains the shared forms. In the My.Stocks.Forms.UWP project, right-click the References node and choose Add Reference. In the Reference Manager that appears, ensure that Projects | Solution is selected, and then select MyStocks.Forms to add the reference (see **Figure 5**).

Modify the Code in the UWP Project to Use Xamarin.Forms and Load the App from the Shared PCL Project Now I need to override code in the app template, which was added as part of the new UWP project, so it will use Xamarin.Forms to load the app from the shared PCL project. Insert the following highlighted code into the start of App::OnLaunched, which is in MyStocks.Forms.UWP | App.xaml | App.xaml.cs:

```
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
  // Initialize Xamarin.Forms here
  Xamarin.Forms.Forms.Init(e);

  #if DEBUG
  ...
}
```

In the MainPage constructor, which is in MyStocks.Forms.UWP | MainPage.xaml | MainPage.xaml.cs, add the following highlighted code to load the Xamarin.Forms project:

```
public MainPage()
{
  this.InitializeComponent();
  this.LoadApplication(new MyStocks.Forms.App());
}
```

In the same file, remove MainPage's inheritance from Page:

```
public sealed partial class MainPage : Page
```

Then make the following changes to MainPage.xaml (MyStocks.Forms.UWP | MainPage.xaml):

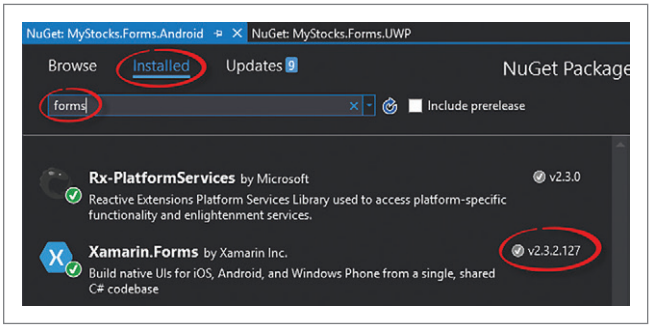
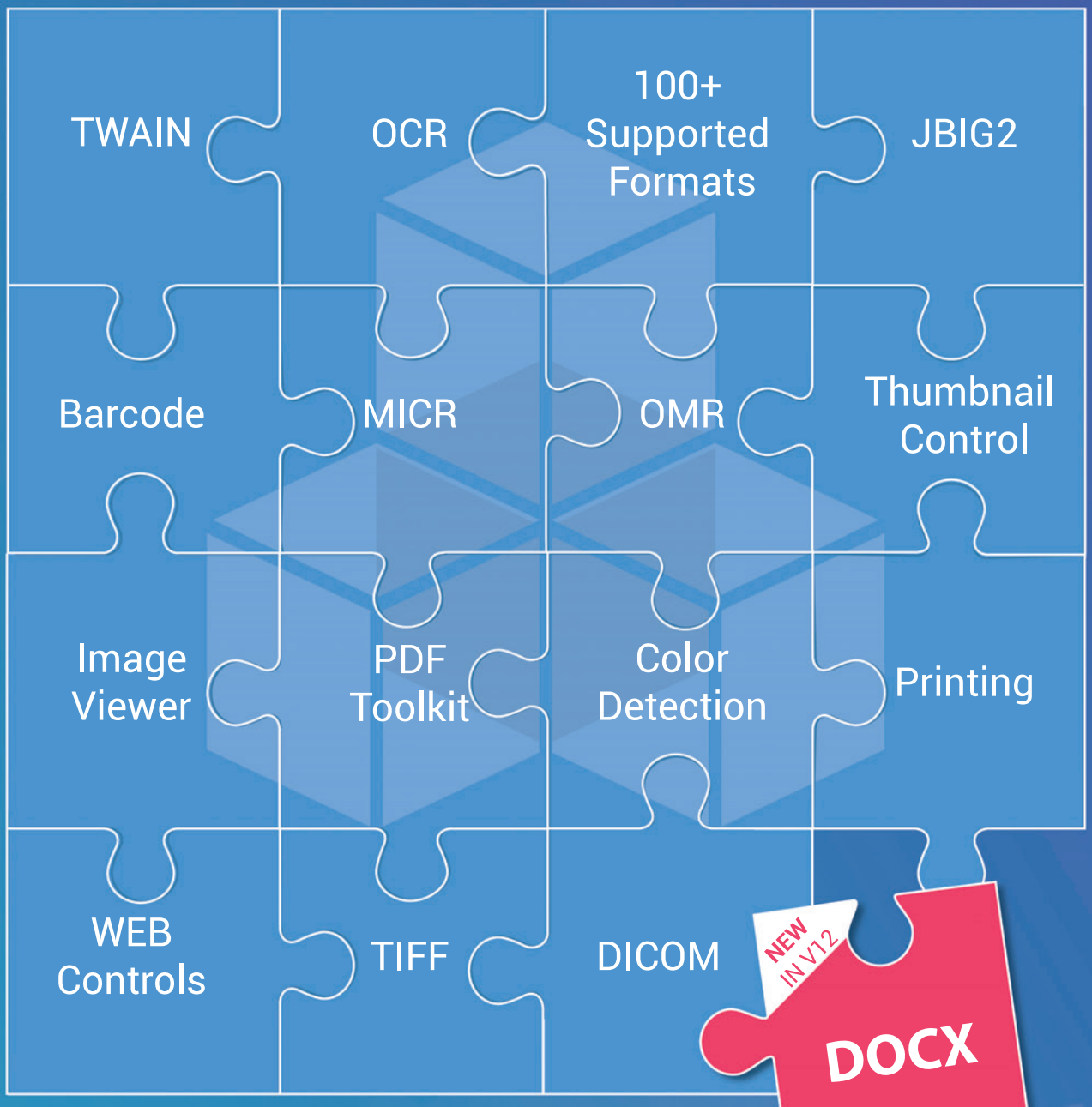


Figure 4 **Verifying the Versions of Xamarin.Forms Match**

GdPicture.NET 12

100% ROYALTY FREE

Imaging SDK For **WinForms**, **WPF** And **Web** Development



Try **GdPicture.NET V12** for **Free** for 30 days

www.gdpicture.com

GdPicture.NET is an ORPALIS imaging technologies product

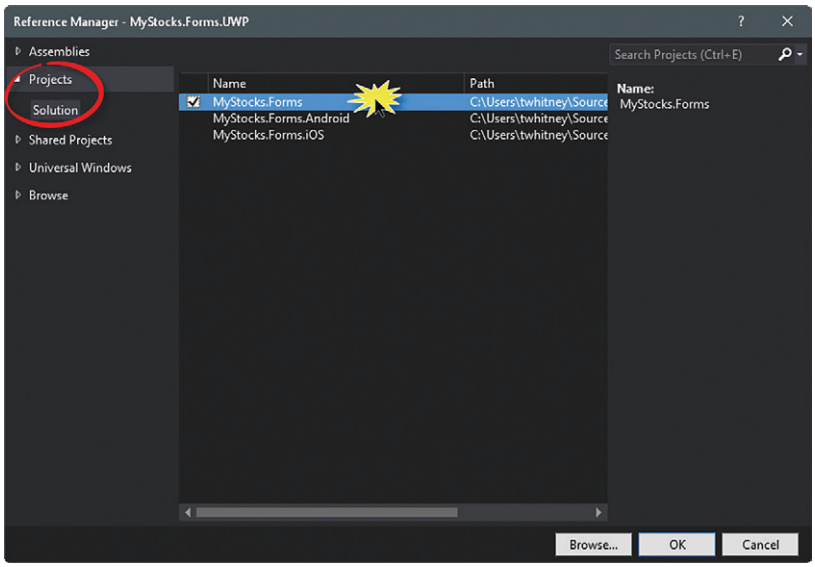


Figure 5 **Adding a Reference to the Portable Class Library**

- Inside the <Page> tag, add: xmlns:forms="using:Xamarin.Forms.Platform.UWP":

```
<Page
  x:Class="MyStocks.Forms.UWP.MainPage"
  xmlns:forms="using:Xamarin.Forms.Platform.UWP"
  ...
```

- Change the opening <Page> tag to <forms:WindowsPage and ensure that the closing tag changes from </Page> to </forms:WindowsPage>>

Now the page will inherit from Xamarin.Forms.Platform.UWP.WindowsPage.

Add the MyStocks.Forms.UWP Project to the Build Configuration After adding a UWP project to your Xamarin.Forms project, ensure it's configured to build. First, right-click on the MyStocks.Forms.UWP project node and select Set as StartUp project.

Then right-click on the solution node and select Configuration Manager. In the Configuration Manager dialog, ensure the MyStocks.Forms.UWP checkboxes for Build and Deploy are checked.

If you've been following these steps, you initially removed the MyStocks.Forms.UWP project. The new MyStocks.Forms.UWP project doesn't have the background image or the NuGet packages that were added to the original project to get the value of a stock symbol, access the Twitter API and so on. To view the result of the steps taken to add a UWP project to James Montemagno's sample app, clone a fresh copy of his GitHub project at the link provided earlier, or install the following NuGet packages to your MyStocks.Forms.UWP project: linqtotwitter, Microsoft.Bcl, Microsoft.BCL.Build, Microsoft.Bcl.Compression, Microsoft.Net.Http, Newtonsoft.Json and Xam.Plugins.TextToSpeech.

You've now added a UWP project to your Xamarin solution. The UWP project will load and run the Xamarin.Forms app from the UWP Blank App template. **Figure 6** shows the app running on Android, iOS and Windows 10.

Xamarin.Forms does the work of mapping controls from the shared forms project to the UWP project that was added. Xamarin.Forms also provides a way to introduce platform specificity, if you need to, while still sharing other parts of the UI. Check out Charles Petzold's article, "Embedding Native Views in Your Xamarin.Forms Apps," in this issue and Kevin Ashley's September 2016 article, "Cross-Platform Productivity with Xamarin" (bit.ly/2dYKr8a), for more details.

Functionality specific to your UWP app, such as Live Tiles, custom icons, notifications and so on, will go in the UWP project.

Wrapping Up

If you aren't using Xamarin.Forms for your UI, you can still add a UWP project to your solution. You can share code using a Shared Asset Project, PCL or .NET Standard Library. Then you can build out the UIs for each project—including the UWP project—by using XAML. See "Building Cross-Platform Applications" on the Xamarin developer site (bit.ly/2e3bVOC) for information about best practices.

Microsoft bought Xamarin and continues to invest in the open source project (bit.ly/1MZsCFE). We encourage your feedback and contributions! ■

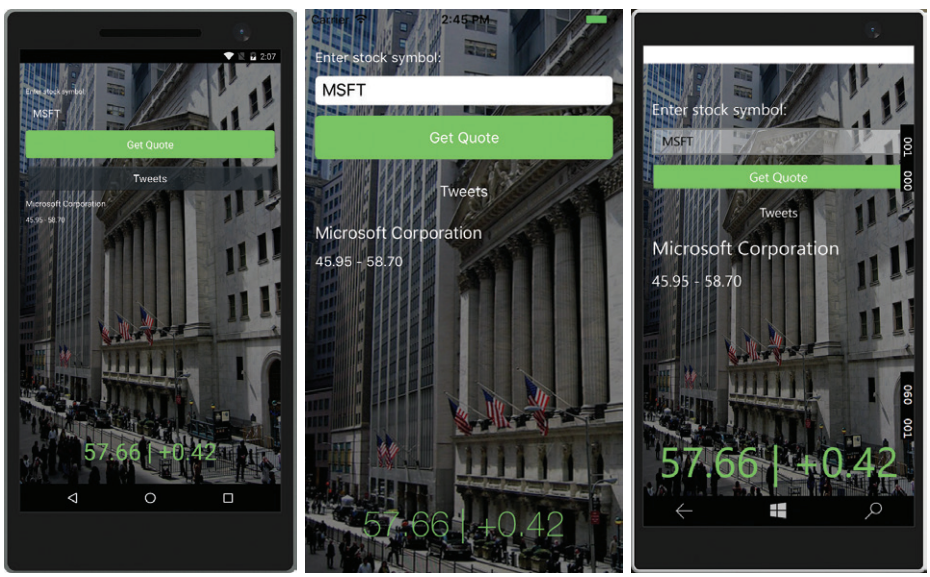


Figure 6 **App Running on Various OSes**

Tyler Whitney *is a senior content developer at Microsoft. He's written about Windows Embedded Compact and now writes about Windows 10 development.*

Thanks to the following Microsoft technical experts for reviewing this article:
Jim Cox, Norm Estabrook, James Montemagno and Jason Short

msdn magazine VP

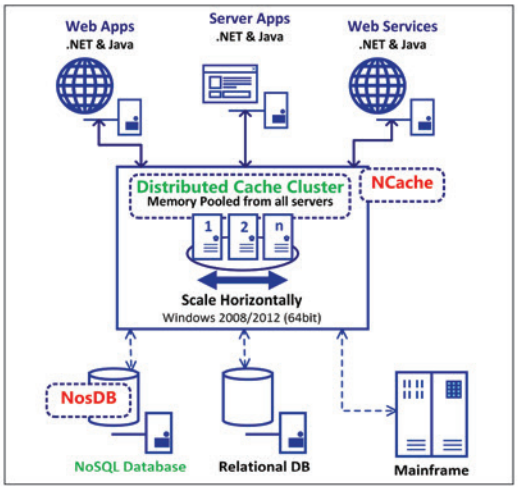
MSDN MAGAZINE VENDOR PROFILE

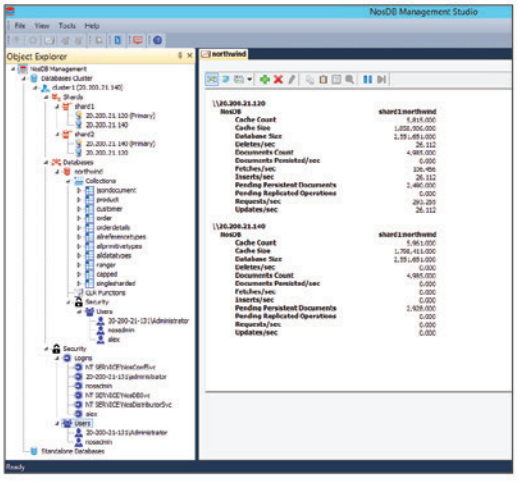
Alachisoft®

NCache and NosDB
Extreme Performance Linear Scalability

Industry leader for over 10 years. Powerful Open Source solutions (Apache 2.0)

President and Technology Evangelist Iqbal Khan co-founded Alachisoft to provide world class performance and scalability solutions for .NET applications. Thanks to our customers, Alachisoft achieved its goal with NCache, the .NET caching market leader for 10 years running. Now the company offers the first fully featured NoSQL Database for .NET, called NosDB. Look to NCache and NosDB for your database scalability and performance needs.





NCache

Distributed Cache for .NET (Open Source)

NCache is an Open Source .NET distributed cache. It caches app data and linearly scales applications to easily accommodate extreme transaction processing (XTP). Use NCache for:

- In-Memory App Data Caching
- ASP.NET Sessions State
- Runtime Data Sharing with Events

NCache is the most powerful distributed cache in the market. See why with "NCache vs. Redis" comparisons on our website.

NosDB

NoSQL Database for .NET (Open Source)

NosDB is an Open Source NoSQL Database for .NET. NosDB provides schema-free JSON documents to reduce application development time and offers extreme performance and scalability. And, NosDB provides SQL support for data access and data definition, coupled with the NosDB Management Studio to get you up and running in no time and with a low cost of ownership.

Some NosDB characteristics are:

- Multiple Shards and Data Replication
- Powerful SQL, LINQ, and ADO.NET
- Powerful Admin & Monitoring Tools
- NET, Java, REST, Node.js support

NosDB is the most feature rich NoSQL Database for .NET available. See more in the "NosDB vs. MongoDB" comparison on our website.

To learn more, please visit our website → **www.alachisoft.com**

Embedding Native Views in Your Xamarin.Forms Apps

Charles Petzold

When Xamarin.Forms debuted less than three years ago, application programmers immediately recognized it as a powerful and versatile solution for cross-platform mobile development. You can create a Xamarin.Forms solution in Visual Studio and write an entire mobile app in C#, with or without XAML, that you can compile for iOS, Android and the Universal Windows Platform (UWP). On the macOS you can use Xamarin Studio to target iOS and Android.

Xamarin.Forms includes some 20 controls, such as Button, Slider and Entry, that are often referred to as views because they derive from the Xamarin.Forms.View class. These are rendered on the various platforms using native views, or widgets, or controls, or elements as they're called in various platforms. For example, the Xamarin.Forms Entryview is mapped to an iOS UITextField, an Android EditText and a UWP TextBox.

This article discusses:

- Platform-specific extension methods
- Embedding native views in Xamarin.Forms XAML files
- Data binding and the MVVM pattern
- Shared Asset Programs vs. Portable Class Libraries

Technologies discussed:

Xamarin.Forms

Code download available at:

msdn.com/magazine/1116Connect_magcode

What makes this possible is an extensible infrastructure of platform renderers that encapsulate the native view and expose a uniform collection of properties and events that correspond to the API of the corresponding Xamarin.Forms view. You can define your own custom views and support them with your own renderers, but it's not a trivial job. For that reason, Xamarin.Forms has been enhanced recently to introduce various extensibility shortcuts that avoid the hassle of writing renderers.

> This is important: You can't enable XAML compilation when using XAML native views.

One of the most compelling of these shortcuts is called native views, a feature that lets you instantiate native iOS, Android and UWP views alongside the normal Xamarin.Forms views. That's what this article is all about. The story of native views begins with code, but becomes much more interesting when XAML gets involved.

Platform-Specific Extension Methods

Xamarin.Forms supports native views with several platform-specific classes. Each of the Xamarin.Forms platforms contains a LayoutExtensions class with an extension method named ToView that you can call on descendants of the following native types:

Figure 1 **The HelloNativeViews Class**

```
using System;
using Xamarin.Forms;

#if __IOS__
using Xamarin.Forms.Platform.iOS;
using UIKit;

#elif __ANDROID__
using Xamarin.Forms.Platform.Android;
using Android.Graphics;
using Android.Widget;

#elif WINDOWS_UWP
using Xamarin.Forms.Platform.UWP;
using Windows.UI.Text;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Media;
#endif

namespace HelloNativeViews
{
  public class HelloNativeViewsPage : ContentPage
  {
    public HelloNativeViewsPage()
    {
      View view = null;

#if __IOS__
      UILabel label = new UILabel
      {
        Text = "Hello iOS Native!",
        Font = UIFont.FromName("Papyrus", 32f),
      };
      view = label.ToView();

#elif __ANDROID__
      TextView textView = new TextView(Forms.Context)
      {
        Text = "Hello Android Native!",
        Typeface = Typeface.Create("cursive", TypefaceStyle.Normal),
        TextSize = 32f
      };
      view = textView.ToView();

#elif WINDOWS_UWP
      TextBlock textBlock = new TextBlock
      {
        Text = "Hello Windows Native!",
        FontFamily = new FontFamily("Georgia"),
        FontStyle = FontStyle.Italic,
        FontSize = 32
      };
      view = textBlock.ToView();
#endif

      view.HorizontalOptions = LayoutOptions.Center;
      view.VerticalOptions = LayoutOptions.Center;
      Content = view;
    }
  }
}
```

- iOS: UIKit.UIView
- Android: Android.Views.View
- UWP: Windows.UI.Xaml.FrameworkElement

Each version of the ToView method returns a platform-specific instance of NativeViewWrapper, which derives from Xamarin.Forms.View. NativeViewWrapper inherits all the public and protected members of Xamarin.Forms.View, and despite being platform-specific, is treated within Xamarin.Forms like a normal View instance. A second extension method is Add, which performs the ToView operation while adding the View object to a layout such as StackLayout.

Each platform's version of NativeViewWrapper has a corresponding renderer: a class named NativeViewWrapperRenderer that's simpler than most renderers because it doesn't need to support any properties, methods or events of the underlying native control. (Xamarin.Forms is open source, so you can examine these and related classes at github.com/xamarin/Xamarin.Forms.)

Let's see how this works.

Normally a Xamarin.Forms solution contains tiny stub application projects for each platform and a common Portable Class Library (PCL) that contains the bulk of your Xamarin.Forms application. However, when using native views in code, you can't use a PCL. Instead, you'll need to put your Xamarin.Forms code in a shared project, which at Xamarin is often called a Shared Asset Project or SAP. In the New Project dialog of Visual Studio select Blank App (Xamarin.Forms Shared) rather than the usual Blank App (Xamarin.Forms Portable). (In Xamarin Studio you select between Portable Class Library or Shared Library with radio buttons.) The code in this shared project is effectively an extension of each application, which means you can use C# conditional compilation directives (#if, #elif and #endif) to delimit the platform-specific code.

Figure 2 **The HelloNativeViews Program on iOS, Android and Windows 10 Mobile**

Among the downloadable code for this article is the HelloNativeViews program, with an SAP that contains the page class shown in **Figure 1**. (Note that normal code indentation practices have been altered in some code samples to fit in available space.) This class creates a label for each platform: a UILabel for iOS, a TextView for Android and a TextBlock for the UWP. It then calls ToView to convert each of these objects to a Xamarin.Forms.View object, but which is actually a NativeViewWrapper object. The page can then apply

Xamarin.Forms properties such as VerticalOptions and HorizontalOptions to the View and finally set it to the Content property of the page. **Figure 2** shows the program running on all three platforms, each with a font distinctive to that platform.

Of course, you can get the same effect entirely in standard Xamarin.Forms by setting the FontFamily property of a Label to a Device.OnPlatform method call that references the three font family names. But I think you can see how you can expand this technique in a much more sophisticated manner by taking advantage of the specific APIs that each platform supports.

Sometimes you might need to apply custom sizing methods to these views so they behave properly as children of a Xamarin.Forms layout object. Check out the article on the Xamarin developer site at bit.ly/2dhBxDk for more details.

While this is an interesting technique, of course it sure would be much nicer to instantiate these native views directly in XAML.

XAML Native Views

As of Xamarin.Forms 2.3.3 (which is in pre-release state as I write this article) you can indeed embed native views in your Xamarin.Forms XAML files. You can set properties and event handlers on these views. You can include views from multiple platforms side-by-side in the same XAML file, and they can interact with all the other Xamarin.Forms views.

One key to this feature is an extension to the XML namespace (xmlns) declaration for XAML files. Custom XML namespaces in Xamarin.Forms commonly use clr-namespace to denote the Common Language Runtime (CLR) namespace, and assembly for the assembly. The new item is targetPlatform, which indicates to which platform this particular XML namespace applies. You set this item to a member of the Xamarin.Forms TargetPlatform enumeration: iOS, Android or Windows for the UWP.

For example, you can define the following XML namespace:

```
xmlns:ios="clr-namespace:UIKit;assembly=Xamarin.iOS;targetPlatform=iOS"
```

You can use this prefix within the XAML file to reference any class or structure in the iOS UIKit namespace, for example:

```
<ios:UILabel Text="Hello iOS Native!"
             TextColor="{x:Static ios:UIColor.Red}"
             View.VerticalOptions="Center"
             View.HorizontalOptions="Center" \>
```

Text and TextColor are properties of UILabel, and TextColor is set to a static read-only property of UIColor. However, notice that the VerticalOptions and HorizontalOptions attributes are prefaced by View, and they are indeed properties of the Xamarin.Forms View class. This syntax—a class, dot and property name—is commonly used for attached bindable properties, and here it indicates that these properties are later applied to the View object that results from the conversion of UILabel to a NativeViewWrapper object. You can use this syntax only for properties that are backed by bindable properties.

To reference Android widgets you'll need an XML namespace something like this (which I've shown on three lines here, but which in the XAML file must be on one line without spaces):

```
xmlns:androidWidget="clr-namespace:Android.Widget;
                     assembly=Mono.Android;
                       targetPlatform=Android"
```

Figure 3 **The XAML File for the XamlNativeViewDemo Program**

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:XamlNativeViewDemo"
             xmlns:ios="clr-namespace:UIKit; ... "
             xmlns:androidWidget="clr-namespace:Android.Widget; ... "
             xmlns:androidGraphics="clr-namespace:Android.Graphics; ... "
             xmlns:formsAndroid="clr-namespace:Xamarin.Forms; ... "
             xmlns:winui="clr-namespace:Windows.UI; ... "
             xmlns:winText="clr-namespace:Windows.UI.Text; ... "
             xmlns:winControls="clr-namespace:Windows.UI.Xaml.Controls; ... "
             xmlns:winMedia="clr-namespace:Windows.UI.Xaml.Media; ... "
             x:Class="XamlNativeViewDemo.XamlNativeViewDemoPage">

  <Grid x:Name="grid">
    <ContentView x:Name="textToRotateParent"
                 Grid.Row="0"
                 VerticalOptions="Center"
                 HorizontalOptions="Center">

      <ios:UILabel Text="Text to Rotate"
                   TextColor="{x:Static ios:UIColor.Red}">
        <ios:UILabel.Font>
          <ios:UIFont x:FactoryMethod="FromName">
            <x:Arguments>
              <x:String>Papyrus</x:String>
              <x:Single>32</x:Single>
            </x:Arguments>
          </ios:UIFont>
        </ios:UILabel.Font>
      </ios:UILabel>

      <androidWidget:TextView x:Arguments=
                          "{x:Static formsAndroid:Forms.Context}"
                            Text="Text to Rotate"
                            TextSize="32">
        <androidWidget:TextView.Typeface>
          <androidGraphics:Typeface x:FactoryMethod="Create">
            <x:Arguments>
              <x:String>cursive</x:String>
              <androidGraphics:
                TypefaceStyle>Normal</androidGraphics:TypefaceStyle>
            </x:Arguments>
          </androidGraphics:Typeface>
        </androidWidget:TextView.Typeface>
      </androidWidget:TextView>

      <winControls:TextBlock Text="Text to Rotate"
                             FontSize="32"
                             FontStyle="{x:Static winText:FontStyle.Italic}">
        <winControls:TextBlock.FontFamily>
          <winMedia:FontFamily>
            <x:Arguments>
              <x:String>Georgia</x:String>
            </x:Arguments>
          </winMedia:FontFamily>
        </winControls:TextBlock.FontFamily>

        <winControls:TextBlock.Foreground>
          <winMedia:SolidColorBrush Color="{x:Static winui:Colors.Red}" />
        </winControls:TextBlock.Foreground>
      </winControls:TextBlock>
    </ContentView>

    <ContentView x:Name="rotateButtonParent"
                 Grid.Row="1"
                 VerticalOptions="Center"
                 HorizontalOptions="Center">

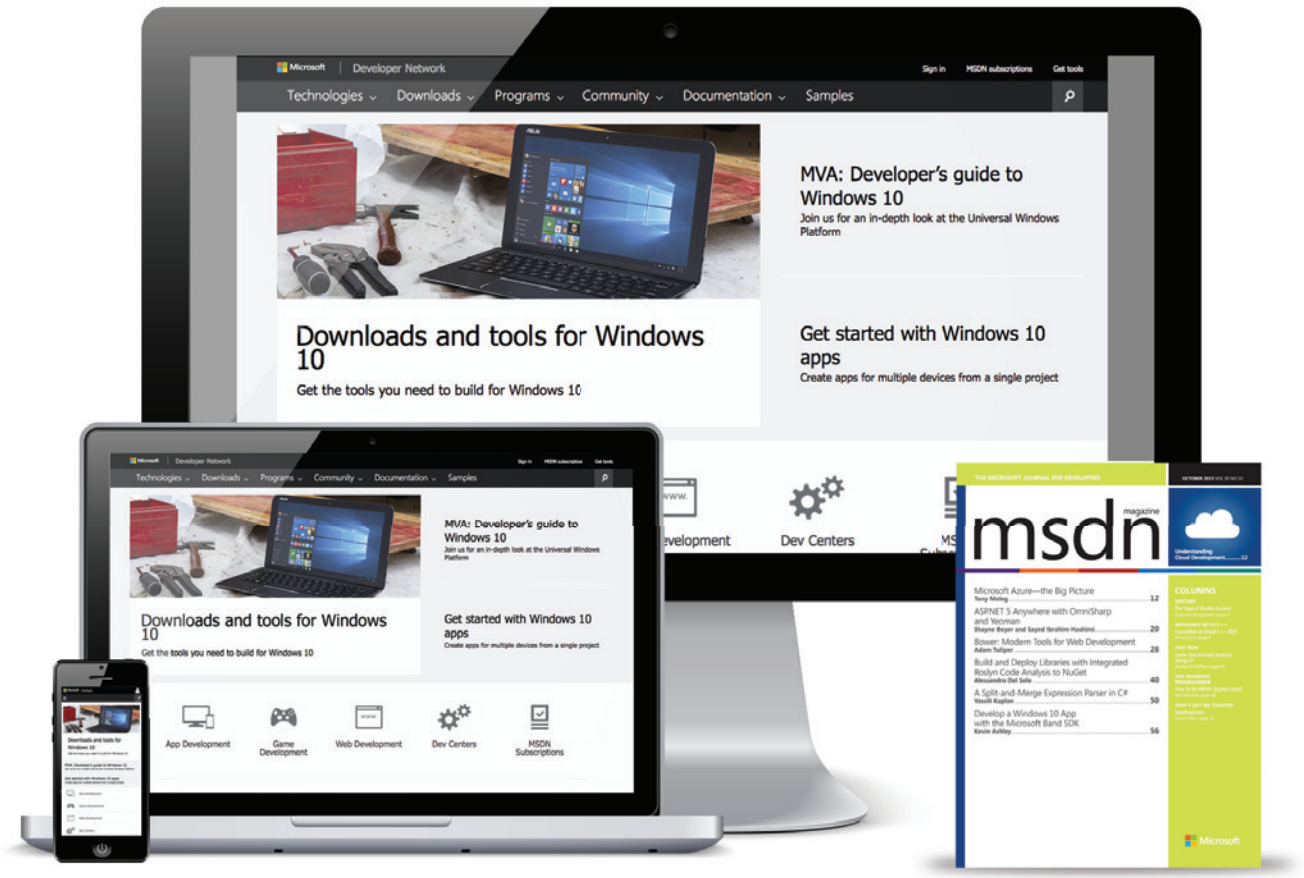
      <ios:UIButton TouchUpInside="OnButtonTap" />

      <androidWidget:Button x:Arguments="{x:Static formsAndroid:Forms.Context}"
                            Text="Rotate the Text"
                            Click="OnButtonTap" />

      <winControls:Button Content="Rotate the Text" />
    </ContentView>
  </Grid>
</ContentPage>
```

msdn magazine

Where you need us most.



Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS

LIVE! 360
TECH EVENTS WITH PERSPECTIVE

MSDN.microsoft.com

You can use any name for this namespace, of course, but I avoided using just android because Android is a little trickier than iOS: Widget constructors generally require the Android Context object as an argument. This Context object is available as a public static property of the Forms class in the Xamarin.Forms namespace in the Xamarin.Forms.Platform.Android assembly. To obtain this Context object, you'll need this XML namespace (which also must be on one line in the XAML file):

```
xmlns:formsAndroid="clr-namespace:Xamarin.Forms;
                    assembly=Xamarin.Forms.Platform.Android;
                      targetPlatform=Android"
```

You can then instantiate an Android widget in XAML by passing an argument to the constructor using the x:Arguments attribute with an x:Static markup extension:

```
<androidWidget:TextView x:Arguments="{x:Static formsAndroid:Forms.Context}"
                        Text="Hello Android Native!" />
```

The assembly name for the UWP is quite long, specifically: Windows, Version=255.255.255.255, Culture=neutral, PublicKeyToken=null, ContentType=WindowsRuntime. For Android and the UWP, you'll probably need multiple XML namespace specifications for the various CLR namespaces used for the various classes, structures and enumerations that tend to be involved in UI markup.

Keep in mind that when the XAML parser encounters one of these native views, it doesn't have access to type converters commonly used to convert XAML text strings into objects, so the markup tends to be a little more extensive to match property and object types. Often you'll need to create objects explicitly in XAML using constructors or factory methods, which means that if you're not familiar with the x:Arguments tag and the x:FactoryMethod element, now is a good time to learn.

> I optimistically began XamlNativeViewDemo as a PCL project, but it soon became apparent that the XAML needed a little help.

This is important: You can't enable XAML compilation when using XAML native views. The compile-time XAML parser doesn't have references to these native types. The parsing must be delayed until runtime, and at that point the XAML parser simply ignores anything with an XML namespace prefix that has a targetPlatform that doesn't match the platform on which the program is running. (I'm told the Xamarin.Forms developers are working on removing this restriction.)

You can't use styles with native views. Styles can target only properties that are backed by BindableProperty objects, and native views don't have such properties.

Because these XAML native views are instantiated by the runtime XAML parser, you can include them in a XAML file in either a PCL project or an SAP. However, if you need to refer to a native view from the codebehind file, you must use an SAP and delimit the platform-specific code with C# conditional compilation directives.

Here's another restriction: With either a PCL or an SAP, you can't use x:Name on a XAML native view. The problem is that the compile-time XAML parser generates a code file containing these named objects as fields, but if these fields are based on platform-specific types, the generated code can't be compiled for all the platforms.

The XamlNativeViewDemo program contains the XAML file shown in **Figure 3** with three platform-specific red-text strings and three platform-specific buttons. Pressing a button invokes an event handler in the codebehind file that rotates the text in a circle.

I optimistically began XamlNativeViewDemo as a PCL project, but it soon became apparent that the XAML needed a little help. You can't even set the text on an iOS UIButton from XAML. You need to call a method, and that requires code. Similarly, you can't set the text color on an Android TextView with a property, and the Click handler for the UWP Button is of type RoutedEventHandler, which involves a RoutedEventArgs object that doesn't derive from EventArgs and, hence, requires a platform-specific handler.

Figure 4 **The Codebehind File for the XamlNativeViewDemo Program**

```
using System;
using Xamarin.Forms;

namespace XamlNativeViewDemo
{
  public partial class XamlNativeViewDemoPage : ContentPage
  {
    View viewTextToRotate;

    public XamlNativeViewDemoPage()
    {
      InitializeComponent();

      viewTextToRotate = textToRotateParent.Content;
      View rotateButton = rotateButtonParent.Content;

#if __ANDROID__
      // Set Android text color
      var wrapper =
        (Xamarin.Forms.Platform.Android.NativeViewWrapper)
          viewTextToRotate;
      var textView = (Android.Widget.TextView)wrapper.NativeView;
      textView.SetTextColor(Android.Graphics.Color.Red);
#endif

#if __IOS__
      // Set iOS button text and color
      var wrapper = (Xamarin.Forms.Platform.iOS.NativeViewWrapper)rotateButton;
      var button = (UIKit.UIButton)wrapper.NativeView;
      button.SetTitle("Rotate the Text", UIKit.UIControlState.Normal);
      button.SetTitleColor(UIKit.UIColor.Black, UIKit.UIControlState.Normal);
#endif

#if WINDOWS_UWP
      // Set UWP button Click handler
      var wrapper = (Xamarin.Forms.Platform.UWP.NativeViewWrapper)rotateButton;
      var button = (Windows.UI.Xaml.Controls.Button)wrapper.NativeElement;
      button.Click += (sender, args) => OnButtonTap(sender, EventArgs.Empty);
#endif
    }

    void OnButtonTap(object sender, EventArgs args)
    {
      viewTextToRotate.RelRotateTo(360);
    }
  }
}
```

Figure 5 **The XAML File for the PlatformRgbSliders Program**

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:PlatformRgbSliders"
             xmlns:ios="clr-namespace:UIKit; ... "
             xmlns:androidWidget="clr-namespace:Android.Widget; ... "
             xmlns:formsAndroid="clr-namespace:Xamarin.Forms; ... "
             xmlns:winControls="clr-namespace:Windows.UI.Xaml.Controls; ... "
             xmlns:winMedia="clr-namespace:Windows.UI.Xaml.Media; ... "
             xmlns:winui="clr-namespace:Windows.UI; ... "
             x:Class="PlatformRgbSliders.PlatformRgbSlidersPage">
  <ContentPage.Padding>
    <OnPlatform x:TypeArguments="Thickness"
                iOS="0, 20, 0, 0" />
  </ContentPage.Padding>

  <ContentPage.Resources>
    <ResourceDictionary>
      <local:DoubleToSingleConverter x:Key="doubleToSingle" />
      <local:DoubleToIntConverter x:Key="doubleToInt"
                                  Multiplier="256" />
    </ResourceDictionary>
  </ContentPage.Resources>

  <ContentPage.BindingContext>
    <local:RgbColorViewModel Color="Gray" />
  </ContentPage.BindingContext>

  <Grid>
    <Grid.RowDefinitions>
      <RowDefinition Height="3*" />
      <RowDefinition Height="*" />
      <RowDefinition Height="*" />
      <RowDefinition Height="*" />
    </Grid.RowDefinitions>

    <BoxView Grid.Row="0"
             Color="{Binding Color}" />

    <ios:UISlider Grid.Row="1"
                  ThumbTintColor="{x:Static ios:UIColor.Red}"
                  MinimumTrackTintColor="{x:Static ios:UIColor.Black}"
                  MaximumTrackTintColor="{x:Static ios:UIColor.Red}"
                  Value="{Binding Red,
                          Mode=TwoWay,
                          UpdateSourceEventName=ValueChanged,
                          Converter={StaticResource doubleToSingle}}"/>
    ...

    <androidWidget:SeekBar x:Arguments=
                           "{x:Static formsAndroid:Forms.Context}"
                           Grid.Row="2"
                           Max="256"
                           Progress="{Binding Green,
                             Mode=TwoWay,
                             UpdateSourceEventName=ProgressChanged,
                             Converter={StaticResource doubleToInt}}" />
    ...

    <winControls:Slider Grid.Row="3"
                        Maximum="1"
                        StepFrequency="0.01"
                        IsThumbToolTipEnabled="True"
                        Value="{Binding Blue,
                                Mode=TwoWay,
                                UpdateSourceEventName=ValueChanged}">
      <winControls:Slider.Foreground>
        <winMedia:SolidColorBrush Color=
          "{x:Static winui:Colors.Blue}" />
      </winControls:Slider.Foreground>
    </winControls:Slider>
  </Grid>
</ContentPage>
```

These problems implied that the codebehind file needed to compensate for the limitations of the XAML, which further implied that I needed to abandon the PCL and use an SAP instead. Even with an SAP, you can't use x:Name on the native views, so I put the native views in a ContentView with an x:Name attribute to get access to them in the codebehind file, which is shown in **Figure 4**. The ContentView is also handy for setting some layout properties (such as VerticalOptions and HorizontalOptions) to avoid a lot of repetition on the native views.

I've fully qualified all the platform-specific types in the codebehind file for clarity and to avoid a bunch of platform using directives. The key to getting the underlying native view from the NativeViewWrapper is a property named NativeView (for iOS and Android) or NativeElement (for UWP).

The iOS and Android buttons can share the same event handler in the codebehind file because it's defined as an EventHandler delegate. But the UWP Button must use a separate event handler of type RoutedEventHandler, which is implemented by simply calling the handler used for iOS and Android.

Another approach to getting access to the native views in the codebehind file involves enumerating children of layout objects and searching for various types or ids. All three platforms define a Tag property—integer on iOS and object on Android and UWP—that you can use for this purpose.

I find the XamlNativeViewDemo program unsatisfactory because I don't like using SAP for my Xamarin.Forms apps. I don't know if you're as passionate as I am about preferring PCL to SAP, but if you are, you'll be happy to know that the final two programs in this article are pure PCL.

Data Bindings and MVVM

One of the best ways to avoid code in the codebehind file is to structure your application around the Model-View-ViewModel architecture (MVVM). All the interactions among the views on

Figure 6 **The PlatformRgbSliders Program Running on the Three Platforms**

the page occur within the platform-independent ViewModel. The ViewModel connects to the View (the UI) using Xamarin.Forms data bindings. The data-binding sources are properties of the ViewModel while the data-binding targets are properties of a view.

But wait a minute. Earlier I mentioned that you can't use a Style for native views because styled properties must be backed by bindable properties. Data bindings have the same restriction: The target property of a data binding—and with MVVM these targets are always views on the page—must also be a property backed by a BindableProperty object. So how can you set bindings on native views?

Here's the good news: To support data bindings on XAML native views, each platform contains SetBinding extension methods that automatically generate ad hoc BindableProperty objects on the fly. These BindableProperty objects allow the native property values to be changed from the ViewModel.

Figure 7 **The PlatformSpinners XAML File**

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:PlatformSpinners"
             xmlns:iosLocal="clr-namespace:PlatformSpinners.iOS; ... "
             xmlns:androidLocal="clr-namespace:PlatformSpinners.Droid; ... "
             xmlns:formsAndroid="clr-namespace:Xamarin.Forms; ... "
             xmlns:winControls="clr-namespace:Windows.UI.Xaml.Controls; ... "
             x:Class="PlatformSpinners.PlatformSpinnersPage">

  <ContentPage.Padding>
    <OnPlatform x:TypeArguments="Thickness"
                iOS="0, 20, 0, 0" />
  </ContentPage.Padding>

  <ContentPage.BindingContext>
    <local:ColorNameViewModel SelectedColorName="Black" />
  </ContentPage.BindingContext>

  <Grid>
    <Grid.RowDefinitions>
      <RowDefinition Height="*" />
      <RowDefinition Height="*" />
    </Grid.RowDefinitions>

    <BoxView Grid.Row="0"
             Color="{Binding SelectedColor}" />


    <iosLocal:PropertiedUIPickerView Grid.Row="1"
                                     Items="{Binding ColorNames}"
                                     SelectedItem=
                                       "{Binding SelectedColorName,
                                        Mode=TwoWay,
                                        UpdateSourceEventName=ValueChanged}"/>

    <androidLocal:PropertiedSpinner x:Arguments=
                                      "{x:Static formsAndroid:Forms.Context}"
                                    Grid.Row="1"
                                    View.VerticalOptions="Center"
                                    Items="{Binding ColorNames}"
                                    SelectedObject=
                                      "{Binding SelectedColorName,
                                      Mode=TwoWay,
                                      UpdateSourceEventName=ItemSelected}" />
    <winControls:ComboBox Grid.Row="1"
                          View.VerticalOptions="Center"
                          ItemsSource="{Binding ColorNames}"
                          SelectedItem=
                            "{Binding SelectedColorName,
                            Mode=TwoWay,
                            UpdateSourceEventName=SelectionChanged}"/>
  </Grid>
</ContentPage>
```

Now you might be considering another problem: In many cases these data bindings need to go both ways—not only from source to target but from target to source. Changes in the UI view must be reflected in ViewModel properties. The Xamarin.Forms BindableProperty infrastructure supports notifications of property changes through the INotifyPropertyChanged interface, but native views don't support this interface, so how can the Binding object know when a property of a native view changes value?

The Binding class now has a new property: UpdateSourceEventName. In the binding markup extension in XAML you can set this property to the name of an event in the native view that signals when the target property has changed.

The PlatformRgbSliders program shown in **Figure 5** is a simple example. (Some repetitious markup has been replaced with ellipses.) The XAML file contains three sets of platform-specific sliders, which I've enhanced by giving them a color corresponding to their function in the program. (It wasn't possible to do this for the Android SeekBar.) The RgbColorViewModel set to the BindingContext of the page defines Red, Green and Blue properties that it uses to construct a Color property.

The binding on the iOS UISlider requires a value converter to convert the double values in the ViewModel to float values, and the Android SeekBar requires a value converter to convert the double values to integer values. You can see that all the data bindings use the UpdateSourceEventName property so the Binding class can be notified when the user has changed the slider value. The result is shown in **Figure 6**.

> Many iOS and Android native views are just not conducive to instantiating in XAML, and there's really no reason why they should be.

Here's an interesting experiment: Remove the UpdateSourceEventName items from the Windows Slider bindings. The program still works. This is because Xamarin.Forms is able to use the notification mechanism built into the UWP dependency properties. Also, work is being done to allow the UpdateSourceEventName to be eliminated on iOS views if the view implements key-value observing.

PlatformRgbSliders has no problem-specific code in the code-behind file, so there's no problem with using a PCL. But, admittedly, PlatformRgbSliders is simple. Will you be able to use PCLs in larger programs?

At first, it doesn't look promising: Many iOS and Android native views are just not conducive to instantiating in XAML, and there's really no reason why they should be. The problem can be summarized: There aren't always enough properties in iOS and Android views for the important options that need to be set and accessed. Instead, there are too many methods.

Figure 8 **PlatformSpinners Running on the Three Platforms**

Too Many Methods

To make iOS and Android more amenable to XAML, you need to replace some of the methods with properties. Obviously the UWP is much better in this regard because it's designed for XAML, but as you saw, UWP event handlers are often based on a platform-specific delegates.

The obvious solution to this problem is to subclass the platform-specific views in wrappers that define a more XAML-friendly API that consists of properties and platform-independent events. You might also be starting to see that the *real* power of embedding native views in XAML comes when you create (or consume) custom iOS, Android and UWP views to use in your Xamarin.Forms application.

But where do you put these classes?

> It might seem weird and unnatural at first for a XAML file in a PCL to reference classes in application assemblies, but it works just fine.

If you're using an SAP, you could put them in the SAP and surround them with C# conditional compilation directives. But if you want to use a PCL—and you usually want to use a PCL—then you can't do that. A PCL can only reference another PCL, and by definition a PCL can't contain any iOS, Android or UWP code. It's a bit of a puzzle.

At least initially.

Keep in mind that you're not compiling the XAML. The XAML is parsed at compile time but that's mostly to generate a code file containing fields corresponding to the x:Name attributes. You've already seen that uncompiled XAML can contain references to iOS, Android and UWP classes. These references are resolved at run time rather than compile time, and that makes all the difference in the world. At run time, the XAML parser has access to all the assemblies that comprise the application, and that includes the individual platform start-up projects.

This means that you can put platform-specific classes in the platform application projects, or you can put the code in platform-specific libraries referenced by these application projects. These classes can be referenced from XAML. It might seem weird and unnatural at first for a XAML file in a PCL to reference classes in application assemblies, but it works just fine.

The PlatformSpinners solution demonstrates this technique. The idea is to use the iOS UIPickerView, the Android Spinner and the UWP ComboBox for selecting something from a list, but expectedly the UIPickerView and Spinner have some methods that need to be exposed as properties. In addition, the UIPickerView requires a data model that must be implemented in code.

For this reason, the PlatformSpinners.iOS application project contains a PickerDataModel and a PropertiedUIPickerView that derives from UIPickerView, so called because it adds essential properties to UIPickerView. The PlatformSpinners.Droid project contains a PropertiedSpinner that derives from Spinner.

The PlatformSpinners PCL contains a simple view model that exposes a collection of the Xamarin.Forms color names, and converts these color names to the actual colors. **Figure 7** shows the complete XAML file except for the long XML namespaces, and **Figure 8** shows it running on the three platforms with the Android Spinner and UWP ComboBox opened to show the options.

This technique of providing additional properties is something you'll likely want to do when consuming third-party iOS and Android custom views. Subclass the view to make it conducive to XAML and MVVM data bindings, and in general you'll be home free.

Is It Really Easier?

You've seen how Xamarin.Forms now allows you to reference native views—or classes derived from native views—directly in XAML rather than hiding the platform-specific code away in a renderer and defining a platform-independent interface to it.

So, you might ask: Is this really easier than creating renderers?

Yes, it is. ■

Charles Petzold *has written numerous articles for* MSDN Magazine *and its predecessor,* Microsoft Systems Journal, *over the past 30 years. He now works in the Xamarin documentation group at Microsoft and is the author of "Creating Mobile Apps with Xamarin.Forms," a free book available for downloading from the Xamarin Web site.*

Thanks to the following Microsoft technical experts for reviewing this article: *David Britch, Stephane Delcroix and Rui Marinho*

Orlando 2016

ROYAL PACIFIC RESORT AT UNIVERSAL ORLANDO

DEC 5-9

Visual Studio® LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

Journey into Code

Join us as we journey into real-world, practical education and training on the Microsoft Platform. Visual Studio Live! (VSLive!™) returns to warm, sunny Orlando for the conference more developers rely on to expand their .NET skills and the ability to build better applications.

twitter.com/live360
@live360

facebook.com
Search "Live 360"

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

EVENT PARTNERS

Microsoft Magenic

PLATINUM SPONSORS

Alachisoft HYPERGRID SMARTBEAR Syncfusion

GOLD SPONSORS

GOVERLAN IDERA QuickBase

SUPPORTED BY

Visual Studio

LIVE! 360

TECH EVENTS WITH PERSPECTIVE

6 Great Conferences 1 Great Price

Visual Studio Live! Orlando is part of Live! 360, the Ultimate Education Destination. This means you'll have access to five (5) other co-located events at no additional cost:

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

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

NEW! APPDEV TRENDS
ENTERPRISE FOCUSED. CODE DRIVEN.

Six (6) events and hundreds of sessions to choose from – mix and match sessions to create your own, custom event line-up – it's like no other conference available today!

TURN THE PAGE FOR MORE EVENT DETAILS

Whether you are an

- Engineer
- Developer
- Programmer
- Software Architect
- Software Designer

You will walk away from this event having expanded your .NET skills and the ability to build better applications.

REGISTER WITH DISCOUNT CODE L360NOV AND **SAVE $300!**

Must use discount code L360NOV

Scan the QR code to register or for more event details.

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

ROYAL PACIFIC RESORT AT UNIVERSAL ORLANDO

DEC 5-9

Check Out the Additional Sessions for Devs, IT Pros, & DBAs at Live! 360

LIVE! 360

Office & SharePoint LIVE!

ON-PREMISE, CLOUD & CROSS-PLATFORM TRAINING

Office & SharePoint Live! features 12+ developer sessions, including:

- Workshop: A Beginner's Guide to Client Side Development in SharePoint - *Mark Rackley*
- Become a Developer Hero by Building Office Add-ins - *Bill Ayres*
- Utilizing jQuery in SharePoint - Get More Done Faster - *Mark Rackley*
- Using the Office UI Fabric - *Paul Schaeflein*
- Enterprise JavaScript Development Patterns - *Rob Windsor*
- Leveraging Angular2 to Build Office Add-ins - *Andrew Connell*
- Webhooks in Office 365 - *Paul Schaeflein*

SQL Server LIVE!

TRAINING FOR DBAs AND IT PROS

SQL Server Live! features 15+ developer sessions, including:

- What's New in SQL Server 2016 - *Leonard Lobel*
- Powerful T-SQL Improvements that Reduce Query Complexity - *Hugo Kornelius*
- Implementing Data Protection and Security in SQL Server 2016 - *Steve Jones*
- Welcome To The 2016 Query Store! - *Janis Griffin*
- Workshop: Big Data, BI and Analytics on The Microsoft Stack - *Andrew Brust*

TECHMENTOR

IN-DEPTH TRAINING FOR IT PROS

TechMentor features IT Pro and DBA sessions, including:

- Workshop: 67 VMware vSphere Tricks That'll Pay for This Conference! - *Greg Shields*
- Secure Access Everywhere! Implementing DirectAccess in Windows Server 2016 - *Richard Hicks*
- Getting Started with Nano Server - *Jeffery Hicks*
- Creating Class-Based PowerShell Tools - *Jeffery Hicks*
- Harvesting the Web: Using PowerShell to Scrape Screens, Exploit Web Services, and Save Time - *Mark Minasi*
- PowerShell Unplugged: Stump Don - *Don Jones*
- Facing Increasing Malware Threats and a Growing Trend of BYO with a New Approach of PC Security - *Yung Chou*

ALM / DevOps	Cloud Computing	Mobile Client	Software Practices	Visual Studio / .NET Framework

START TIME	END TIME		
5:00 PM	8:00 PM	Pre-Conference Registration - Royal Pacific Resort Conference Center	
6:00 PM	9:00 PM	Dine-A-Round Dinner @ Universal CityWalk	
START TIME	**END TIME**		
8:00 AM	5:00 PM	**VSM01** Workshop: Distributed Cross-Platform Application Architecture - *Rockford Lhotka & Jason Bock*	
5:00 PM	6:00 PM	EXPO Preview	
6:00 PM	7:00 PM	**Live! 360 Keynote: Digital Transformation** - *David Foote, Co-founder,*	
START TIME	**END TIME**		
8:00 AM	9:00 AM	**Visual Studio Live! / Modern Apps Live!** - *Tim Sneath, Principal*	
9:00 AM	9:30 AM	Networking Break • Visit the EXPO	
9:30 AM	10:45 AM	**VST01** Building Applications with ASP.NET Core - *Scott Allen*	**VST02** Busy .NET Developer's Guide to Swift - *Ted Neward*
11:00 AM	12:15 PM	**VST05** Richer MVC Sites with Knockout JS - *Miguel Castro*	**VST06** Busy .NET Developer's Guide to Native iOS - *Ted Neward*
12:15 PM	2:00 PM	Lunch • Visit the EXPO	
2:00 PM	3:15 PM	**VST09** WCF & Web API: Can We All Just Get Along?!? - *Miguel Castro*	**VST10** Creating Great Looking Android Applications Using Material Design - *Kevin Ford*
3:15 PM	4:15 PM	Networking Break • Visit the EXPO	
4:15 PM	5:30 PM	**VST13** Busy Developer's Guide to Chrome Development - *Ted Neward*	**VST14** Creating Cordova Apps using Ionic and Angular 2 - *Kevin Ford*
5:30 PM	7:30 PM	Exhibitor Reception	
START TIME	**END TIME**		
8:00 AM	9:15 AM	**VSW01** Moving from Angular 1 to Angular 2 - *Ben Dewey*	**VSW02** The Future of Mobile Application Search - *James Montemagno*
9:30 AM	10:45 AM	**VSW05** Getting Started with Aurelia - *Brian Noyes*	**VSW06** Building Connected and Disconnected Mobile Applications - *James Montemagno*
10:45 AM	11:15 AM	Networking Break • Visit the EXPO	
11:15 AM	12:15 PM	**Live! 360 Keynote:** To Be Announced	
12:15 PM	1:45 PM	Birds-of-a-Feather Lunch • Visit the EXPO	
1:45 PM	3:00 PM	**VSW09** Living in a Command Line Web Development World (NPM, Bower, Gulp, and More) - *Ben Dewey*	**VSW10** Understanding the Windows Desktop App Development Landscape - *Brian Noyes*
3:00 PM	4:00 PM	Networking Break • Visit the EXPO • Expo Raffle @ 3:30 p.m.	
4:00 PM	5:15 PM	**VSW13** Securing Client JavaScript Apps - *Brian Noyes*	**VSW14** Let's Write a Windows 10 App: A Basic Introduction to Universal Apps - *Billy Hollis*
8:00 PM	10:00 PM	Live! 360 Dessert Luau - *Wantilan Pavilion*	
START TIME	**END TIME**		
8:00 AM	9:15 AM	**VSH01** Build Real-Time Websites and Apps with SignalR - *Rachel Appel*	**VSH02** Cognitive Services: Building Smart Applications with Computer Vision - *Nick Landry*
9:30 AM	10:45 AM	**VSH05** HTTP/2: What You Need to Know - *Robert Boedigheimer*	**VSH06** Building Business Apps on the Universal Windows Platform - *Billy Hollis*
11:00 AM	12:15 PM	**VSH09** TypeScript and ES2015 JumpStart - *John Papa*	**VSH10** A Developers Introduction to HoloLens - *Billy Hollis & Brian Randell*
12:15 PM	1:30 PM	Lunch on the Lanai	
1:30 PM	2:45 PM	**VSH13** All Your Tests Are Belong To Us - *Rachel Appel*	**VSH14** Developing Awesome 3D Apps with Unity and C# - *Adam Tuliper*
3:00 PM	4:15 PM	**VSH17** SASS and CSS for Developers - *Robert Boedigheimer*	**VSH18** From Oculus to HoloLens: Building Virtual & Mixed Reality Apps & Games - *Nick Landry*
4:30 PM	5:30 PM	Live! 360 Conference Wrap-Up - *Pacifica 6 - Andrew Brust (Moderator),*	
START TIME	**END TIME**		
8:00 AM	5:00 PM	**VSF01** Workshop: Angular 2 Bootcamp - *John Papa*	
12:00 PM	1:00 PM	Lunch	
1:00 PM	5:00 PM	**VSF01** Workshop Continues	

Speakers and sessions subject to change

AGENDAS AT-A-GLANCE

Visual Studio LIVE! — EXPERT SOLUTIONS FOR .NET DEVELOPERS

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

Presented in Partnership with Magenic

APPDEV TRENDS — ENTERPRISE FOCUSED. CODE DRIVEN.

Web Client	Web Server	Windows Client	Modern Apps Live!	Agile	Containerization	Continuous Integration	Java	Mobile	Cloud

Pre-Conference: Sunday, December 4, 2016

Pre-Conference Workshops: Monday, December 5, 2016

VSM02 Workshop: Service Oriented Technologies - Designing, Developing, & Implementing WCF and the Web API - *Miguel Castro*	**VSM03** Workshop: DevOps in a Day - *Brian Randell*	**MAM01** Workshop: Building Modern Mobile Apps - *Brent Edwards & Kevin Ford*	**ADM01** Workshop: Building Teams - *Steve Green*	**ADM02** Workshop: One Codebase to Rule Them All: Xamarin - *Fabian Williams*

Is Your IT Career on Track as Businesses Look to Become More Agile?
Chief Analyst and Chief Research Officer, Foote Partners

Day 1: Tuesday, December 6, 2016

Keynote: Faster, Leaner, More Productive: The Next Generation of Visual Studio
Lead Program Manager, Visual Studio Platform, Microsoft

App Dev Trends Keynote: You Are the Future of Enterprise Java!
- Reza Rahman, Speaker, Author, Consultant

VST03 What's New in Azure v2 - *Eric D. Boyd*	**VST04** Real World Scrum with Team Foundation Server 2015 & Visual Studio Team Services - *Benjamin Day*	**MAT01** Modern App Development: Transform How You Build Web and Mobile Software - *Rockford Lhotka*	**ADT01** Hacking Technical Debt - *Steve Green*	**ADT02** Java 8 Lambdas and the Streaming API - *Michael Remijan*
VST07 Overview of Power Apps - *Nick Pinheiro*	**VST08** Get Good at DevOps: Feature Flag Deployments with ASP.NET, WebAPI, & JavaScript - *Benjamin Day*	**MAT02** Architecture: The Key to Modern App Success - *Brent Edwards*	**ADT03** Are You A SOLID Coder? - *Steve Green*	**ADT04** PrimeFaces 5: Modern UI Widgets for Java EE - *Kito Mann*
VST11 Introduction to Next Generation of Azure PaaS – Service Fabric and Containers - *Vishwas Lele*	**VST12** To Be Announced	**MAT03** Manage Distributed Teams with Visual Studio Team Services and Git - *Brian Randell*	**ADT05** Agile Architecture - *Steve Green*	**ADT06** Full Stack Java with JSweet, Angular 2, PrimeNG, and JAX-RS - *Kito Mann*
VST15 Cloud Oriented Programming - *Vishwas Lele*	**VST16** Bringing DevOps to the Database - *Steve Jones*	**MAT04** Focus on the User Experience #FTW - *Anthony Handley*	**ADT07** Crafting Innovation - *Steve Green*	**ADT08** Who's Taking Out the Garbage? How Garbage Collection Works in the VM - *Kito Mann*

Day 2: Wednesday, December 7, 2016

VSW03 Managing Enterprise and Consumer Identity with Azure Active Directory - *Nick Pinheiro*	**VSW04** Improving Performance in .NET Applications - *Jason Bock*	**MAW01** DevOps, Continuous Integration, the Cloud, and Docker - *Dan Nordquist*	**ADW01** Stop Killing Requirements! - *Melissa Green*	**ADW02** Migrating Customers to Microsoft Azure: Lessons Learned From the Field - *Ido Flatow*
VSW07 Practical Internet of Things for the Microsoft Developer - *Eric D. Boyd*	**VSW08** I'll Get Back to You: Understanding Task, Await, and Asynchronous Methods - *Jeremy Clark*	**MAW02** Mobile Panel - *Kevin Ford, Rockford Lhotka, James Montemagno, & Jordan Matthiesen*	**ADW03** Meeting-Free Software Development in Distributed Teams - *Yegor Bugayenko*	**ADW04** The Essentials of Building Cloud-Based Web Apps with Azure - *Ido Flatow*
VSW11 How to Scale .NET Apps with Distributed Caching - *Iqbal Khan*	**VSW12** Learn to Love Lambdas (and LINQ, Too) - *Jeremy Clark*	**MAW03** C# Everywhere: How CSLA .NET Enables Amazing Cross-Platform Code Reuse - *Rockford Lhotka*	**ADW05** Introduction to Microsoft Office Graph - *Fabian Williams*	**ADW06** Building IoT and Big Data Solutions on Azure - *Ido Flatow*
VSW15 ARM Yourself for Azure Success - *Esteban Garcia*	**VSW16** Continuous Delivery on Azure: A/B Testing, Canary Releases, and Dark Launching - *Marcel de Vries*	**MAW04** Coding for Quality and Maintainability - *Jason Bock*	**ADW07** As You Think About Azure Databases, Think About DocumentDb - *Fabian Williams*	**ADW08** Where Does JavaScript Belong in the App Store? - *Jordan Matthiesen*

Day 3: Thursday, December 8, 2016

VSH03 C# Best Practices - *Scott Allen*	**VSH04** Application Insights: Measure Your Way to Success - *Esteban Garcia*	**MAH01** Modern Mobile Development: Build a Single App For iOS & Android with Xamarin Forms - *Kevin Ford*	**ADH01** From VMs to Containers: Introducing Docker Containers for Linux and Windows Server - *Ido Flatow*	**ADH02** Continuous Testing in a DevOps World - *Wayne Ariola*
VSH07 Debugging Your Way Through .NET with Visual Studio 2015 - *Ido Flatow*	**VSH08** The Ultimate Intro to Docker for Developers - *Adam Tuliper*	**MAH02** Universal Windows Development: UWP for PC, Tablet & Phone - *Brent Edwards*	**ADH03** CQRS 2.0 - Commands, Actors, and Events...Oh My! - *David Hoerster*	**ADH04** Microservices as Chat Bots Are the Future - *Yegor Bugayenko*
VSH11 Exploring Microservices in a Microsoft Landscape - *Marcel de Vries*	**VSH12** Automated UI Testing for iOS and Android Mobile Apps - *James Montemagno*	**MAH03** Modern Web Development: Building Server Side using .NET Core, MVC, Web API, and Azure - *Allen Conway*	**ADH05** The Curious Case for the Immutable Object - *David Hoerster*	**ADH06** Continuous Integration May Have Negative Effects - *Yegor Bugayenko*
VSH15 Unit Testing Makes Me Faster: Convincing Your Boss, Your Co-Workers, and Yourself - *Jeremy Clark*	**VSH16** Writing Maintainable, X-Browser Automated Tests - *Marcel de Vries*	**MAH04** Modern Web Development: Building Client Side using TypeScript and Angular2 - *Allen Conway*	**ADH07** To Be Announced	**ADH08** Mobile DevOps Demystified with Xamarin, VSTS and HockeyApp - *Roy Cornelissen*
VSH19 User Experience Case Studies - Good and Bad - *Billy Hollis*	**VSH20** Debugging the Web with Fiddler - *Ido Flatow*	**MAH05** Using All That Data: Power BI to the Rescue - *Scott Diehl*	**ADH09** Get Started with Microsoft PowerApps - *Fabian Williams*	**ADH10** Overcoming the Challenges of Mobile Development in the Enterprise - *Roy Cornelissen*

Andrew Connell, Don Jones, Rockford Lhotka, Matthew McDermott, Brian Randell, & John K. Waters

Post-Conference Workshops: Friday, December 9, 2016

VSF02 Workshop: Building Modern Web Apps with Azure - *Eric D. Boyd & Brian Randell*	**MAF01** Workshop: Modern App Deep Dive: Xamarin, Responsive Web, UWP, CSLA .NET - *Jason Bock, Allen Conway, Brent Edwards & Kevin Ford*	**ADF01** Workshop: Applied Agile - *Philip Japikse*
VSF02 Workshop Continues	**MAF01** Workshop Continues	**ADF01** Workshop Continues

The (Interactive) Future of Technical Docs

Craig Dunn

Once upon a time, printed technical books were the primary approach to learning new programming languages and SDKs. Today, you'll find a myriad of content online, from product documentation to developer blogs, from Stack Overflow to GitHub, and from podcasts to YouTube—even Xamarin University online classes.

There can still be barriers to learning, though: Configuring the new IDE you want to start programming in, understanding the File | New Project wizard and all its options, typing or copying sample code into a new or existing project to try it out, and even navigating the resulting solution structure, can all be confusing for the uninitiated. Developers who've switched from Visual Basic to Eclipse to Xcode understand just how different these experiences can be.

This article relies on a pre-release version of Xamarin Workbooks. All information is subject to change.

This article discusses:

- Xamarin Workbooks basics
- Using Workbooks to learn Android, iOS and Xamarin.Forms
- Exploring other APIs with Workbooks
- Writing your own Workbooks

Technologies discussed:

Xamarin Workbooks, C#, Windows Presentation Foundation, Xamarin.Forms, iOS, Android

Code download available at:

bit.ly/2dKhkqm

Roslyn—the Microsoft open source .NET compiler service—facilitates new experiences that mitigate these problems by removing the need for the IDE. Learning experiences, such as Gistlyn (bit.ly/2d00D7b) and the new online C# tutorial from Microsoft (bit.ly/28WyuvW), immerse the developer in documentation and code, without the overhead of solutions and projects. These new tools make learning simpler and more interactive.

Introducing Xamarin Workbooks

Xamarin Workbooks brings this interactive-documentation-plus-live-coding concept to mobile and desktop application development. In conjunction with device simulators, Workbooks gives you the same immersive experience as the online tools I mentioned, but provides the added ability to learn and experiment with the entire native SDKs for Android, iOS, Mac and Windows Presentation Foundation (WPF):

- Learning Xamarin Mobile app development becomes an interactive and exploratory process. Rather than just reading the docs, Workbooks lets you interactively code and test native mobile app features.
- Exploring online APIs—including Microsoft Azure services—can be done without the hassle of starting up a sample app and navigating through a maze of source files. Because you're using the same tools your mobile apps are written with, working code can be copied from Workbooks into your Xamarin app projects.
- Writing your own Workbooks is easy, whether testing out an idea or building your own courseware to teach others. Instead

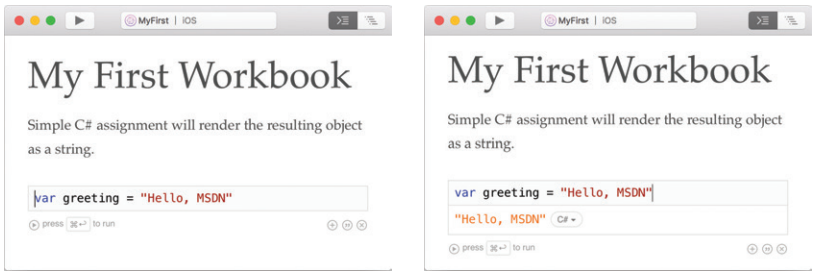


Figure 1 **MyFirst.workbook Before and After Execution**

of creating your hundredth File | New Project to try something new, you get a faster, easier way to experiment. As an added bonus, you can interact with the UI in the simulator, and explore the visual tree in the inspector view.

Getting Started

What, exactly, is Workbooks? In short: It's live, interactive documentation for mobile and desktop platforms. You can download the Workbooks app for Mac and Windows at bit.ly/2ejXBj8. For Android and iOS, you should have Xamarin and the platform SDKs and simulators installed on your system.

Workbooks files are essentially Markdown files with a YAML header; they contain formatted text and code-fenced C#. The file extension is ".workbook." You can write a workbook with any text editor, but the Xamarin Workbooks app includes the ability to create and edit workbooks, so you don't really need to learn the underlying format. Here's some sample Markdown source:

```
---
uti: com.xamarin.workbook
platform: iOS
---
# My First Workbook
Simple C# assignment will render the resulting object as a string.
```csharp
var greeting = "Hello, MSDN"
```
```

Figure 1 shows the resulting workbook before and after execution.

Executable code cells have a grey background and utilize the same editing experience as Visual Studio Code. Features like syntax highlighting, code completion, compilation errors and keyboard shortcuts are all supported.

When you click inside a code cell to give it focus, a set of action buttons appears below it (as shown in **Figure 1**). Press the Play button (arrow inside circle, on the far left) to execute that code cell (and any preceding code). You can also run the entire workbook at once from the Run menu.

Once the code has executed, a result cell is displayed below the code cell. You can edit any part of the code cell, to change how it runs or to experiment with the API, and press Play to run it again. (And again. This is how developers learn with Workbooks—by reading the documentation, and running and modifying the example code as they go.)

The result cell contains a representation of the last object reference in the code cell. By default, the object's string representation is used, but Workbooks supports a number of additional visualizers to help you explore the results. Use the small menu next to the result to choose from the available visualizations, including:

C#: A string escaped for C# (for example, new lines encoded as "\n").

Plain Text: An unescaped string value (for example, new lines are rendered as multi-line text).

> What, exactly, is Workbooks? In short: It's live, interactive documentation for mobile and desktop platforms.

Object Members: Renders a complete list of the object's properties in a table. Large, cyclic object graphs are evaluated on demand, so they don't slow down your coding, but are available to be explored if required, just like with a runtime debugger.

Enumerable: Automatically expands a collection so you can explore each item. Only the first 10 elements are initially displayed, but you can explore more by clicking the Continue Enumerating button.

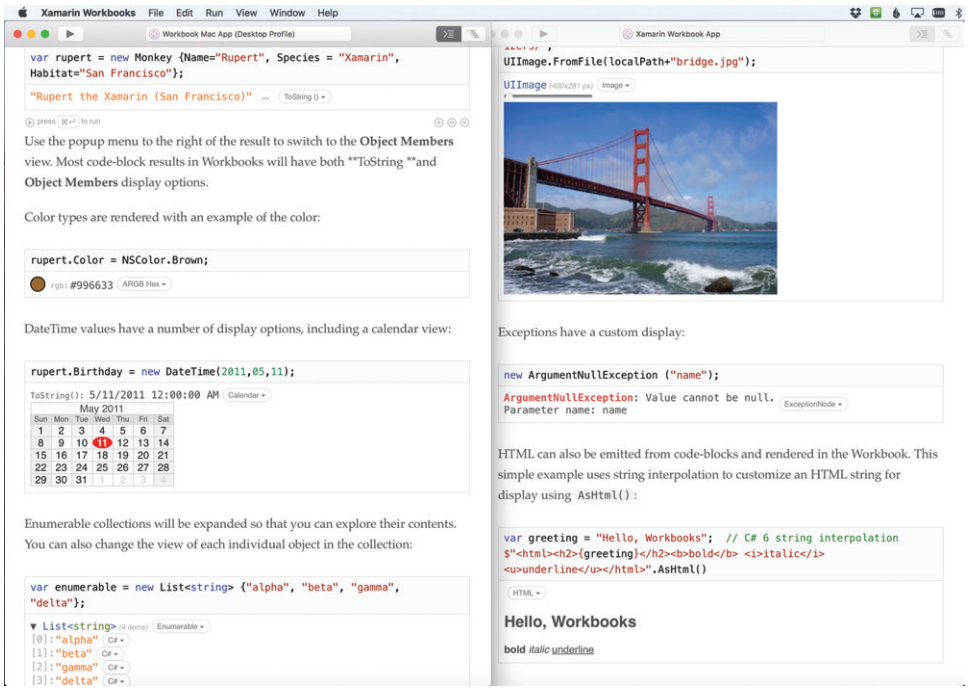


Figure 2 **Visualization Options**

Figure 3 **XamariniOS.workbook**

```
---
uti: com.xamarin.workbook
platform: iOS
packages:
- id: Newtonsoft.Json
  version: 9.0.1
---
# iOS UITableView
This example uses **Json.NET** and `WebClient` to download a file
*monkeydata.json* that is used to bind to the iOS UITableView.

```csharp
#r "Newtonsoft.Json"
#load "json_monkey.csx"
```

To add a UITableView control to the screen, instantiate an instance, set
the bounds, and add to the `RootViewController`. You can experiment with
changing the `Frame`.
```csharp
var tableView = new UITableView();
tableView.Frame = UIScreen.MainScreen.Bounds;
RootViewController.View.AddSubview(tableView);
```

Wiring up a data source (like a generic list of `Monkey` objects) to a
table requires a `UITableViewSource`. This class converts the data into
`UITableViewCell` classes to be rendered.
```csharp
public class MySource : UITableViewSource
{
  string identifier = "mycell";
  public List<Monkey> Data {get;set;} = new List<Monkey>(); // C# 6
  public override nint RowsInSection (UITableView tableview, nint section)
  {
    return Data.Count;
  }
  public override UITableViewCell GetCell (UITableView tableView,
    NSIndexPath indexPath)
  {
    // First, get or create a cell
    UITableViewCell cell = tableView.DequeueReusableCell (identifier);
    if (cell == null)
    { cell = new UITableViewCell (UITableViewCellStyle.Subtitle, identifier); }
    // Then, get the data and set the UI
    cell.TextLabel.Text = Data[indexPath.Row].Name;
    cell.DetailTextLabel.Text = Data[indexPath.Row].Location;
    return cell;
  }
}
```

To display the data in the table, create the source object, assign the
Monkey object list, then assign it to the table
```csharp
var source = new MySource(); // Create the class
source.Data = monkeys;       // Assign the list of strings
tableView.Source = source;   // Give it to the table view
tableView.ReloadData();      // and show on the screen
```
```

Color: A small swatch of color is displayed, along with Hex or RGB representations of its value.

Location: Objects that represent a latitude/longitude location are shown on a map (Mac only).

Image: Valid image view classes can show the image inline in the workbook.

Exception: Highlights exception messages in the result cell.

Html: Workbooks authors can use the AsHtml method in their code cells to have the result cell render a string as HTML.

Figure 2 shows the Visualizers Workbook, including examples of some of these options.

So far you've just seen Workbooks displaying their results inline, but they're even more interesting when exploring native mobile APIs with Xamarin via mobile device simulators.

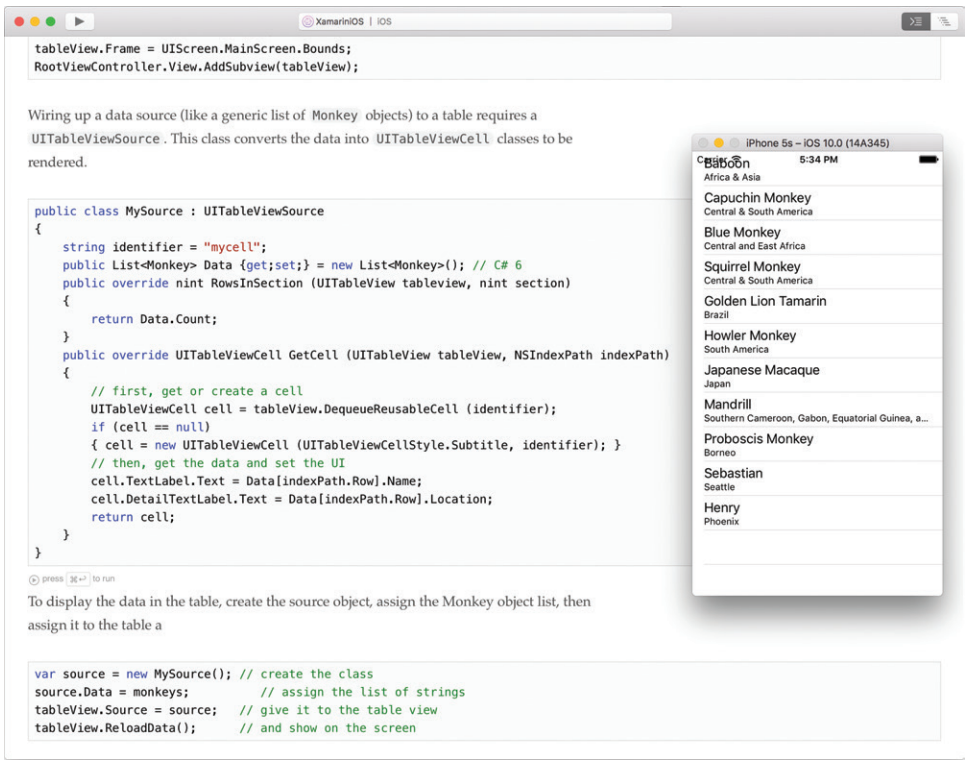


Figure 4 **Workbooks and iOS Simulator**

Learning Xamarin with Workbooks

Once you're comfortable running code cells and interpreting the results, you can visit developer.xamarin.com/workbooks and download Workbooks to learn Android, iOS and Xamarin.Forms. The native platform APIs are available in Workbooks, for tasks as simple as adding a label to the screen, up to rendering 3D content that responds to touch.

When you create or open an iOS, Android or Xamarin.Forms workbook, a mobile device simulator will start up (Mac and Windows Presentation Foundation [WPF] workbooks just start an empty window). These external simulators (or windows) run an agent app that communicates with the workbook—you'll see a status message when the agent is

GROUPDOCS
Document Manipulation APIs

Manipulating Files?

View, annotate, compare, convert, assemble, sign and share over **50 types of documents on the web.**

GroupDocs.**Viewer**
GroupDocs.**Annotation**
GroupDocs.**Conversion**
GroupDocs.**Comparison**
GroupDocs.**Signature**
GroupDocs.**Assembly**
GroupDocs.**Metadata**
GroupDocs.**Search**

Get started now

.NET Libraries | Java Libraries | Cloud APIs | Cloud Apps

Contact Us:
US: +1 903 306 1676
EU: +44 141 628 8900
AU: +61 2 8006 6987
sales@groupdocs.com

Visit us at www.groupdocs.com

being started. Once the agent is running, Xamarin Workbooks is ready to be explored.

As you read through and run each code cell, the app in the simulator changes to reflect each new piece of code. If you're curious about how something works, edit the code and run it again. **Figure 3** shows a simple iOS UITableView example, and **Figure 4** shows it running in the simulator.

Workbooks can also display an exploded view of the UI hierarchy to explore. You can select items in the simulator, the 3D rendering or the visual tree list, and view their properties in a pad. To access this on the Mac, select View | Visual Inspector. On Windows the pads are docked, ready to access at any time. To switch back to the code and documentation, use the View | Workbook menu item.

Exploring APIs

Workbooks is also great for exploring other APIs, such as those provided by NuGet packages or REST endpoints. An example that combines both is a workbook that connects to Azure EasyTables, using the free tryappservice.azure.com back end.

The complete workbook source is shown in **Figure 5**, including the required NuGet packages; and the complete source code to save and retrieve data from an Azure EasyTable is shown running in **Figure 6**.

Of course, there's a lot more to Azure—this example just begins to demonstrate how Workbooks can be applied to the idea of teaching an API.

Writing Your Own Workbooks

Although you can write workbooks in Markdown, the easiest way to create your own is within the Workbooks app itself. Choose the File | New menu item, then choose a platform: iOS and Android can be created on both macOS and Windows. Mac and WPF workbooks can be created only on macOS or Windows, respectively.

Workbooks consists of three types of elements: text cells, code cells and result cells. A brand-new workbook contains a single code cell, where you can immediately type and execute C# code.

Cells can be added and deleted using the action buttons that appear below a cell when it has focus. There are three buttons (visible in **Figure 1**):

- **Plus (+)** – Add a new code cell.
- **Double Quote (")** – Add a new text cell.
- **Delete (x)** – Delete the preceding cell.

All the code cells in a workbook run in the same context, so variables and classes created in one code cell are available in subsequent code cells.

> Workbooks is also great for exploring other APIs, such as those provided by NuGet packages or REST endpoints.

Text can be formatted by selecting it and using the formatting bar shown in **Figure 7**. Options include bold, italic, adding links, and code format, as well as bulleted lists, numbered lists, and blockquotes. There are also six heading levels and the ability to insert images or a horizontal rule.

Code cells support C# keyword highlighting and both member and signature/override auto-completion. Errors are indicated by a red underline, and the error message is displayed when you hover over it.

Each platform has its own mechanism for accessing the UI, as follows:

Android: Building workbooks that utilize the simulator means you need a way to reference the UI from code cells. For Android,

Figure 5 **Azure-TryAppService.workbook**

````
---
uti: com.xamarin.workbook
platform: WPF
packages:
- id: Microsoft.Bcl
  version: 1.1.10
- id: Newtonsoft.Json
  version: 9.0.1
- id: Microsoft.Net.Http
  version: 2.2.29
- id: Microsoft.Azure.Mobile.Client
  version: 3.0.1
---

# Azure TryAppService
Two NuGets - **Microsoft.Azure.Mobile.Client** & Newtonsoft.Json - have been added:

```csharp
#r "Newtonsoft.Json"
#r "Microsoft.WindowsAzure.Mobile"
#r "Microsoft.WindowsAzure.Mobile.Ext"
```
This `TodoItem` class matches the one configured in the TryAppService back end.

```csharp
using Newtonsoft.Json;
using Microsoft.WindowsAzure.MobileServices;

public class TodoItem
{
  [JsonProperty(PropertyName = "id")]
  public string ID {get;set;}
  [JsonProperty(PropertyName = "text")]
  public string Name {get;set;}
  [JsonProperty(PropertyName = "complete")]
  public bool Done {get;set;}
  [Version]
  public string Version { get; set; }
  public override string ToString() {
    return $"{Name} is " + (Done?"done":"not done");
  }
}
```
Sign up at [tryappservice.azure.com](https://tryappservice.azure.com/ "Try Azure for free!") **Mobile App > TodoList**. Replace the URL with *your* temporary, generated endpoint URL:

```csharp
var mobileService = new MobileServiceClient (
  "https://da0cfa57-0ee0-4-231-b9ee.azurewebsites.net/");
  // Replace this with your own
var table = mobileService.GetTable<TodoItem> ();
```
The following code creates a new `TodoItem` class, inserts it into the table on the server, and retrieves the list from the server:

```csharp
var rememberTo = new TodoItem {Name="buy apples"};
await table.InsertAsync (rememberTo);
List<TodoItem> todos = await table.Take (10).ToListAsync ();
```
````

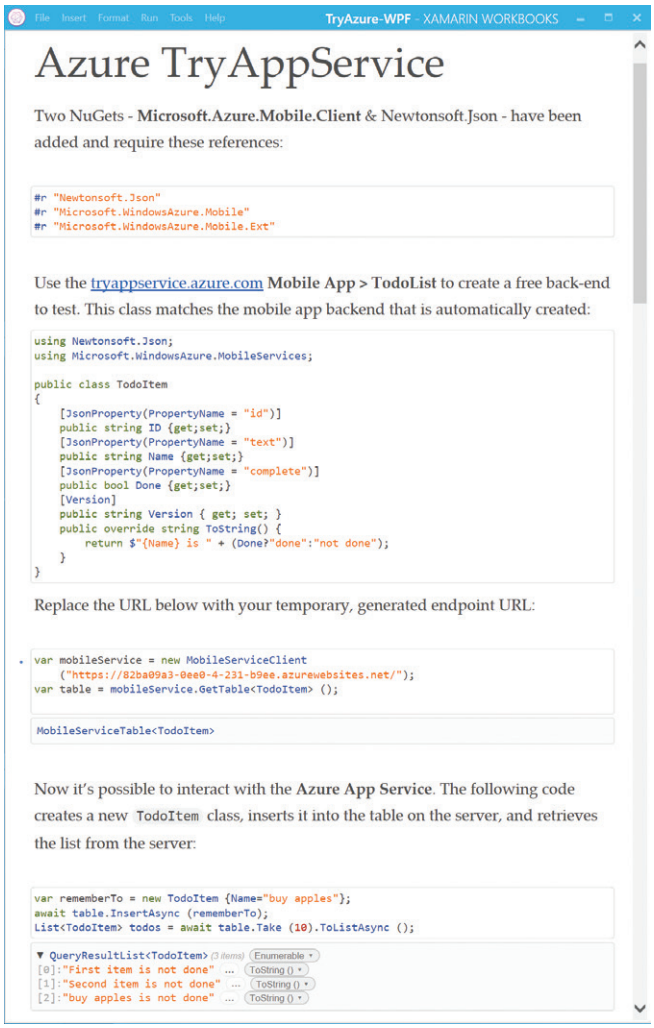


Figure 6 **Using Azure TryAppService Interactively**

use the following code to get a reference to the Activity being used by the simulator:

```
var mainActivity = StartedActivities.First();
var label = new Android.Widget.TextView(mainActivity) {
  Text = "Hello, Workbooks",
  TextSize = 36
};
mainActivity.SetContentView(label);
```

iOS: iOS workbooks expose the RootViewController of the simulator so that you can build up the UI by adding subviews, as shown here:

```
var label = new UIKit.UILabel(new CGRect(10,10,300,50)) {
  Text = "Hello, Workbooks",
  Font = UIFont.SystemFontOfSize(36)
};
RootViewController.Add(label);
```

WPF: Adding controls to the app window in WPF workbooks can be accomplished by setting the Content property on System.Windows.Application.Current.MainWindow, for example:

```
var label = new System.Windows.Controls.Label {
  Text = "Hello, Workbooks",
  FontSize = 36
};
System.Windows.Application.Current.MainWindow.
  Content = label;
```

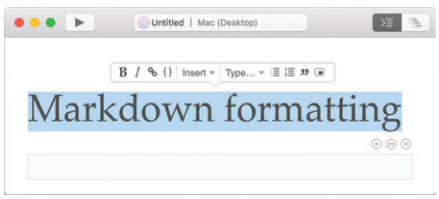


Figure 7 **Formatting Text**

Workbooks also includes features to help you create more-sophisticated code scenarios, so you can:

Reference additional namespaces: By default, only a limited set of namespaces are available in each workbook. Some, such as System.Xml, require a using clause at the start of the code cell before they're used (just like in a regular C# file).

Add NuGet packages: Use the File | Add Package menu item to add NuGet packages to a workbook, and ensure that the assemblies are referenced using the following syntax:

```
#r "Microsoft.WindowsAzure.Mobile"
```

Assemblies referenced this way still require the appropriate using statement in a code cell (as shown in **Figure 6**).

> Workbooks can include NuGet packages and can reference additional namespaces, as well as the entire Android, iOS, macOS and WPF SDKs.

Include additional code: More-complex examples might require lots of code, which would only clutter the workbook if it's displayed inline. External code files can be referenced by a workbook with the following syntax:

```
#load "my-extra-code.csx"
```

Use external files to create supporting classes, UI elements, Model and ViewModel classes or other objects that are required for the workbook to function but aren't necessary to the documentation (and are unlikely to need modification by the user). The workbook in **Figure 3** shows where some data-setup code is added with an external file, to keep the instructions easy-to-read.

Get help: Type help as the last line in any code cell to quickly view the convenience methods available in the workbook.

Wrapping Up

Xamarin Workbooks is now the fastest way to learn mobile application development. You can learn more about the native mobile platforms and explore online APIs. It's also easy to create your own custom workbooks for your blog, training or anything else. Workbooks can include NuGet packages and can reference additional namespaces, as well as the entire Android, iOS, macOS and WPF SDKs.

Visit developer.xamarin.com/workbooks to get started. Xamarin is freely available as part of Visual Studio Community Edition (and also on the Mac), and the Workbooks app is free to download and use. ■

Craig Dunn *is the program manager for Xamarin documentation at Microsoft. Find him on Twitter: @conceptdev or via LinkedIn at linkedin.com/in/conceptdev.*

Thanks to the following Microsoft technical experts for reviewing this article:
Aaron Bockover and David Britch

Rugged DevOps: Integrating Security into the Development and Release Pipeline

Sam Guckenheimer and Jean-Marc Prieur

Security can be a scary topic. According to the Microsoft Security Intelligence Report (bit.ly/2drid6a), last year 17.9 percent of reporting computers encountered security threats. And the urgency to protect is greater than ever. The time to compromise is almost always days or less, if not minutes or less, finds Verizon in its "2016 Data Breach Investigations Report" (vz.to/2dnpNk8).

Often there's a perceived conflict between DevOps practices, which aim for speed, and security practices, which emphasize thoroughness. The conflict doesn't have to exist. In fact, a number of vendors have been working to make DevOps pipelines more secure. And this is part of a movement toward Rugged DevOps. This article discusses four extensions newly available through the Visual Studio Team Services (VSTS) Marketplace.

Let's start with an overview of a Rugged DevOps pipeline (see **Figure 1**).

This article discusses:

- Managing binary components workflow
- Using DevOps to check security of open source components
- The WhiteSource extension for VSTS and TFS
- The Fortify Static Code Analyzer for VSTS
- The Checkmarx extension for VSTS
- The Veracode extension for VSTS

Technologies discussed:

Visual Studio Team Services (VSTS), Team Foundation Server (TFS)

The goal of a Rugged DevOps pipeline is to allow development teams to go fast without breaking things by introducing unwanted vulnerabilities. For case studies of teams implementing Rugged DevOps, see chapter 22 of "The DevOps Handbook" (IT Revolution Press, 2016) by Gene Kim, Jez Humble, Patrick Debois and John Willis (bit.ly/2dUXJW3).

Package Management

Just as a team uses version control as the single source of truth for current source code, it can rely on a package manager as the unique source of binary components. By using binary package management, a development team can create not only a local cache of approved components, but also make this a trusted feed for the continuous integration (CI) pipeline. Package management—now in VSTS and coming to Team Foundation Server (TFS)—is an integral part of the component workflow. The Package Management extension is available from the Visual Studio Marketplace (bit.ly/1VLXIzG).

The Role of OSS Components

Developers today are much more productive than ever, due to the wide availability of reusable open source software (OSS) components. There's now a practical approach to reuse, with runtimes available on Windows and Linux such as .NET Core and Node.js. At the same time, the productivity of reusing OSS comes with the risk that the reused dependencies bring security vulnerabilities. For example, the snyk.io service claims that 76 percent

of its users find security vulnerabilities in their applications due to the versions of the Node.js packages they consume.

In the world of OSS, there's a new concept, sometimes called Software Composition Analysis (SCA). It involves scenarios like this:

> *As a developer or dev lead, when I consume an OSS component, create a dependency or consume dependencies, I want to: start with the latest correct version in order to avoid any old vulnerabilities or license misuse; validate that they are in fact the correct binaries for this version; in the release pipeline, validate binaries to ensure they're correct and to keep a traceable bill of materials; and in the event of a vulnerability, be notified immediately and be able to correct and redeploy automatically in order to prevent any security vulnerability or license misuse from reused software.*

Two vendors that now provide services for managing OSS with VSTS and TFS are WhiteSource and Veracode. Both have extensions available in the Visual Studio Marketplace.

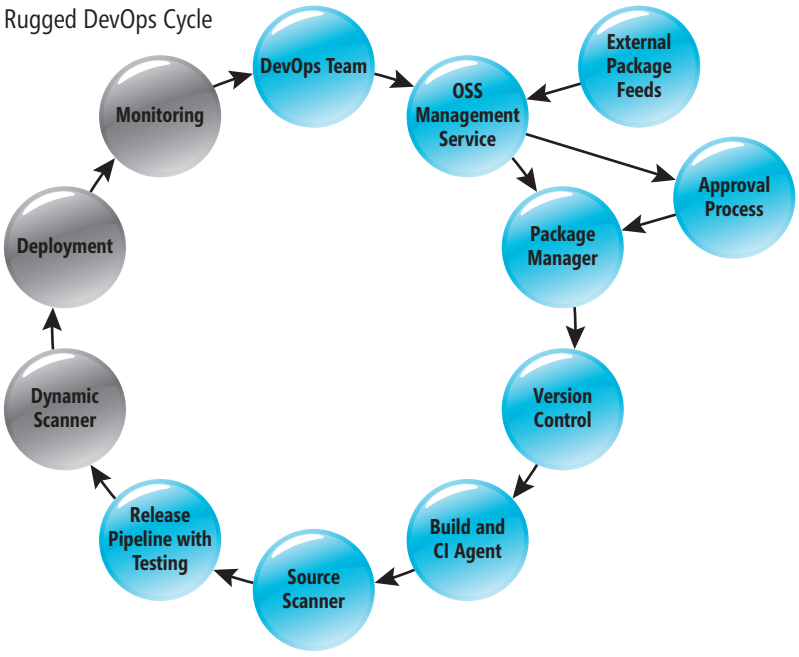


Figure 1 **The Rugged DevOps Workflow**

Solution Spotlight: WhiteSource Extension for VSTS and TFS

For a team consuming external packages, the WhiteSource Extension specifically addresses the questions of open source security, quality and license compliance. In a world where most breaches target known vulnerabilities in known components, this is essential hygiene for consuming open source. We'll discuss some of its capabilities.

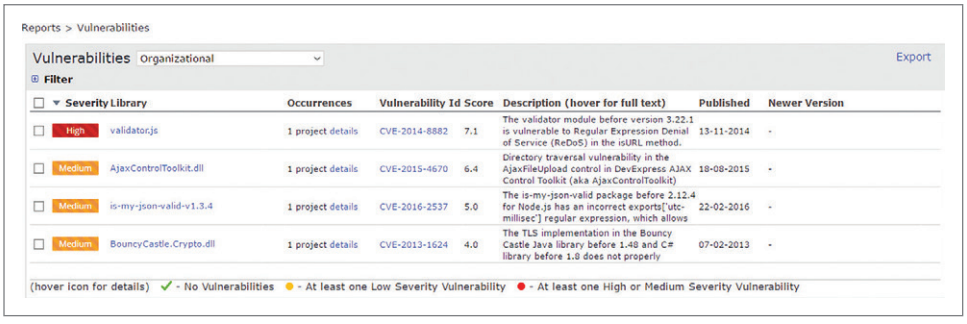


Figure 2 **WhiteSource Detection of Vulnerable Components**

Continuously Detects All Open Source Components in Your Software The WhiteSource extension (bit.ly/2dEMCCO) will automatically detect all open source components—including their transitive dependencies—every time you run a build. This means you can generate a comprehensive inventory report within minutes, based on the last build you ran. It also gives your security, DevOps and legal teams full visibility into your organization's software development process.

Alerts on Open Source Security Vulnerabilities and Their Fixes When a new security vulnerability is discovered, WhiteSource automatically generates an alert and provides targeted remediation guidance (see **Figure 2**). This can include links to patches, fixes, relevant source files and even recommendations to change system configuration to prevent exploitation.

Automatically Enforces Open Source Security and License Compliance Policies According to a company's policies, WhiteSource automatically approves, rejects or triggers a manual approval process every time a new open source component is added to a build. Developers can set up policies based on parameters such as security-vulnerability severity, license type or library age. As soon as a developer attempts to add a problematic open source component, the service will send an alert and fail the build.

> Often there's a perceived conflict between DevOps practices, which aim for speed, and security practices, which emphasize thoroughness.

For searching online repositories such as GitHub and Maven Central, WhiteSource also offers an innovative browser extension. Even before choosing a new component, a developer can see its security vulnerabilities, quality, and license issues, and whether it fits a company's policy.

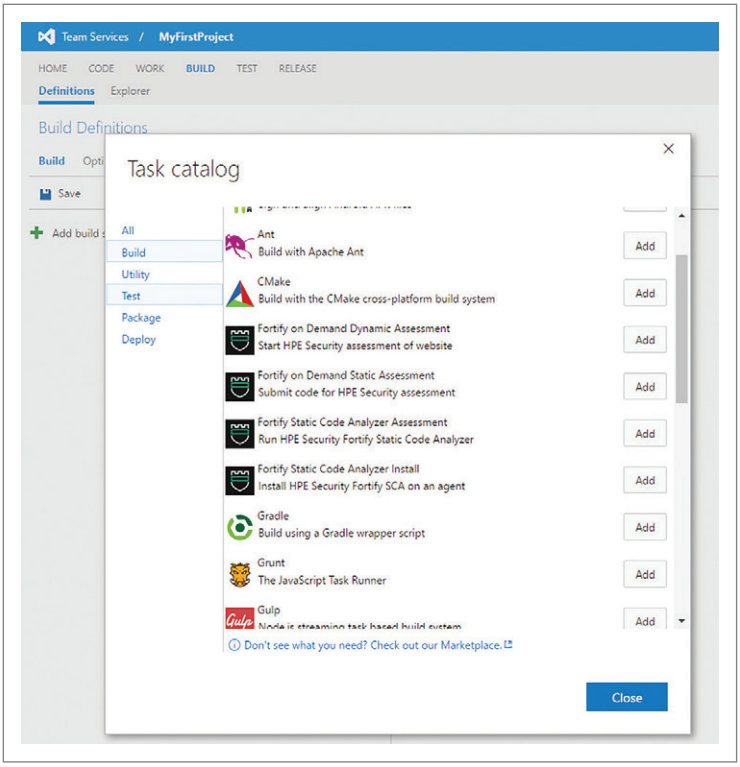


Figure 3 **Visual Studio Team Services Build Tasks for HPE Security Fortify SCA**

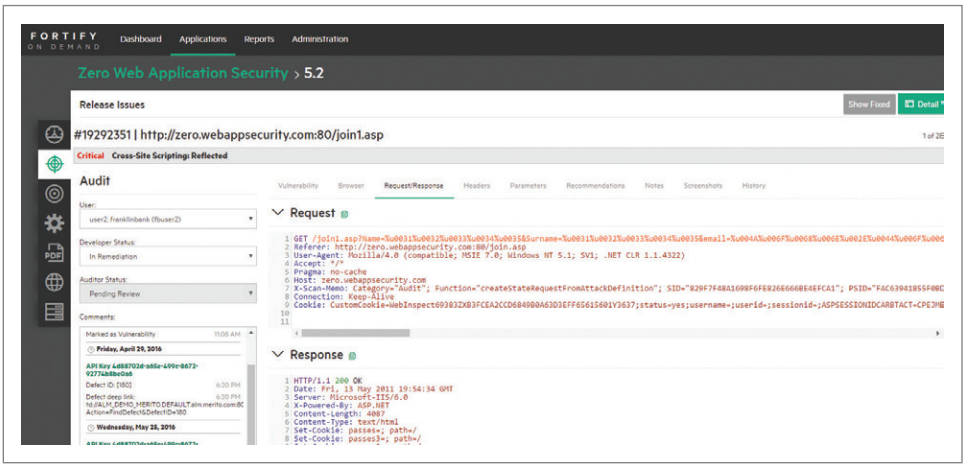


Figure 4 **HPE Security Fortify on Demand Vulnerability Detection**

Solution Spotlight: Extension for VSTS: The Fortify Static Code Analyzer (SCA)

HPE Security Fortify SCA provides static analysis for application security testing through VSTS and TFS (bit.ly/2dEWEOW). This makes software security a seamless part of the coding process by empowering developers to find security vulnerabilities earlier in the DevOps lifecycle.

Fortify SCA provides a comprehensive set of software security analyzers that search for violations of security-specific coding rules and guidelines. Development groups and security professionals use it to analyze application source code for security issues (see **Figure 3**). Fortify SCA identifies root causes of software-security vulnerabilities and delivers accurate, risk-ranked results with line-of-code remediation guidance.

Fortify on Demand also delivers application Security as a Service (SaaS). Fortify on Demand tasks automatically submit static and dynamic scan requests to the application SaaS platform (see **Figure 4**). For static assessments, the project is uploaded to Fortify on Demand. For dynamic assessments, Fortify on Demand uses the application's pre-configured URL.

Balancing Speed and Depth

In the past, security scanning was typically an "over-the-wall" activity, done perhaps only once per release. This creates a horrible pattern in which security specialists find large batches of issues at exactly the time when developers are under the most pressure to ignore them and release. Rugged DevOps strives to make all quality activity—including security—continuous and automated.

All of the extensions described here can fully scan the team's source code. There are multiple points to integrate scanning into the team's workflow.

Pull requests (PRs) are the way DevOps teams submit changes. Prior to the PR, a developer needs to be able to see the effect of code changes and be confident they'll merge correctly and not introduce new issues. In a DevOps process, each PR is typically small and merges are continual, so the master branch of code can stay fresh. Ideally, the developer can check for security issues prior to the PR. WhiteSource facilitates this for validating dependencies with its binary fingerprinting; Checkmarx provides an incremental scan of changes; and Veracode has the concept of a developer sandbox. These approaches allow a developer to experiment with changes before submitting them.

Similarly, CI needs to be optimized for speed to give the development team immediate feedback of any build breaks. Just as in the PR flow, when the scanning is fast enough, it can and should be integrated into the CI build definition. A failed scan can break the build, and security issues can be fixed right away to restore the build to green.

At the same time, continuous delivery (CD) needs to be thorough. In VSTS, CD is typically managed through either release definitions, which progress the build output across environments, or via additional build definitions that can be scheduled—perhaps daily—rather than triggered with each commit. In either case, the definition can perform a longer static analysis scan. The full code project can be scanned and any errors or warnings reviewed offline without blocking the CI flow.

Spreadsheets Made Easy.

Fastest Calculations

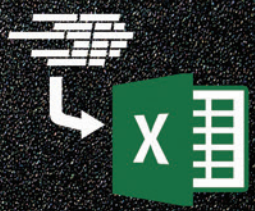
Evaluate complex Excel-based models and business rules with the fastest and most complete Excel-compatible calculation engine available.

Powerful Controls

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

Comprehensive Charting

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



Scalable Reporting

Easily create richly formatted Excel reports without Excel from any ASP.NET, Windows Forms, WPF or Silverlight application.

Download your free fully functional evaluation at SpreadsheetGear.com

SpreadsheetGear

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

Solution Spotlight: Checkmarx Extension for VSTS

The Checkmarx extension for VSTS (bit.ly/2dVyuDg) allows developers to not only scan all source code, but also to just scan new or modified code. This incremental scanning capability is a key enabler for developers in CI environments as it reduces scan times from hours to only a few minutes.

Once the analysis is complete, Checkmarx delivers a concise report detailing the security posture of the scanned code on the VSTS project summary page, which allows developers to address and mitigate the vulnerabilities (see **Figure 5**).

Checkmarx also provides developers with the "Best Fix Location" in order to minimize the time to remediate. This includes a visual chart of the data flow graph that indicates the ideal location in the code to address multiple vulnerabilities within the data flow in a single line of code (see **Figure 6**).

With rule customization, development teams also can modify the existing vulnerability detection preset queries to enhance detection and accuracy. Checkmarx describes the rules in a simple C# syntax.

Secure Code Is Only Part of the Picture

Secure code is necessary, but not sufficient to achieve Rugged DevOps. The Verizon "2016 Data Breach Intelligence Report" makes clear the many attack vectors that skilled criminals use to compromise their targets. In addition to protecting your code, it's essential to protect credentials and secrets. In particular, phishing is becoming ever more sophisticated.

There are several operational practices that a team ought to apply to protect itself, as we'll discuss now.

Authentication and Authorization Use multi-factor authentication even across internal domains and just-in-time administration, such as the PowerShell Just Enough Administration (JEA), to protect against escalation of privilege (aka.ms/jea). Different passwords for different user accounts will limit the damage if one set of credentials is stolen.

Use the CI/CD Release Pipeline Do this to rebuild infrastructure so the release pipeline and cadence can also contain damage. If you manage Infrastructure as Code with Azure Resource Manager or use Azure PaaS or a similar service, then your pipeline will automatically create new instances and destroy them, giving attackers no place to hide inside your infrastructure (bit.ly/2dEY5wR). VSTS will encrypt the secrets in your pipeline, and you should rotate the passwords just as you would other credentials.

Manage Permissions Do this to secure the pipeline with role-based access control just as you would for your source code. You want to control who can edit the build and release definitions you use for production.

Dynamic Scanning This is the process of testing the running application with known attack patterns. For example, OWASP Zed (bit.ly/1fjloVy) is a popular open source dynamic scanner to check against the primary vulnerabilities that owasp.org tracks. Two of the partners discussed in this article, HPE Security and Veracode, provide dynamic scanning services, as well.

Monitoring Production This is a key DevOps practice. The specialized services for detecting anomalies related to intrusion are known as Security Information and Event Management. Azure Security Center focuses on the security incidents related to the Azure cloud (bit.ly/2dzcj5r).

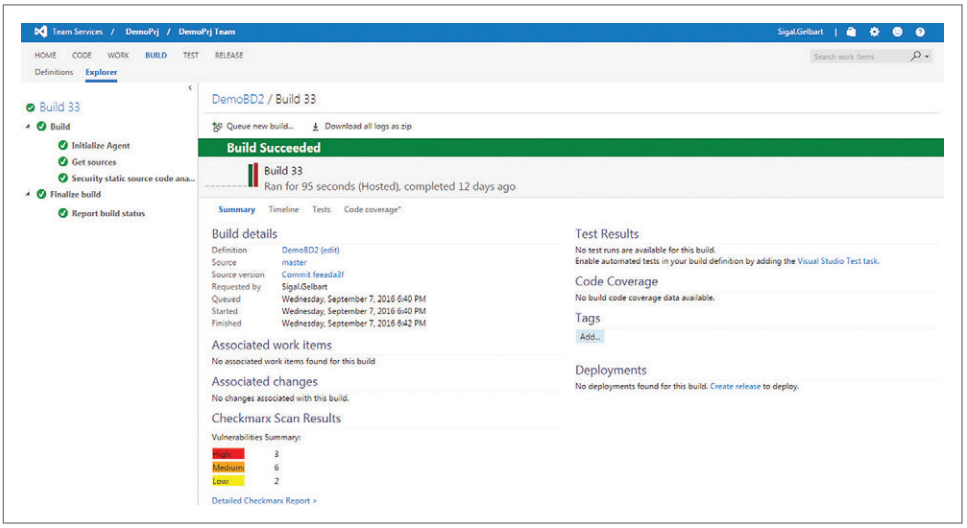


Figure 5 **Visual Studio Team Services Build Report with Checkmarx Scan Results**

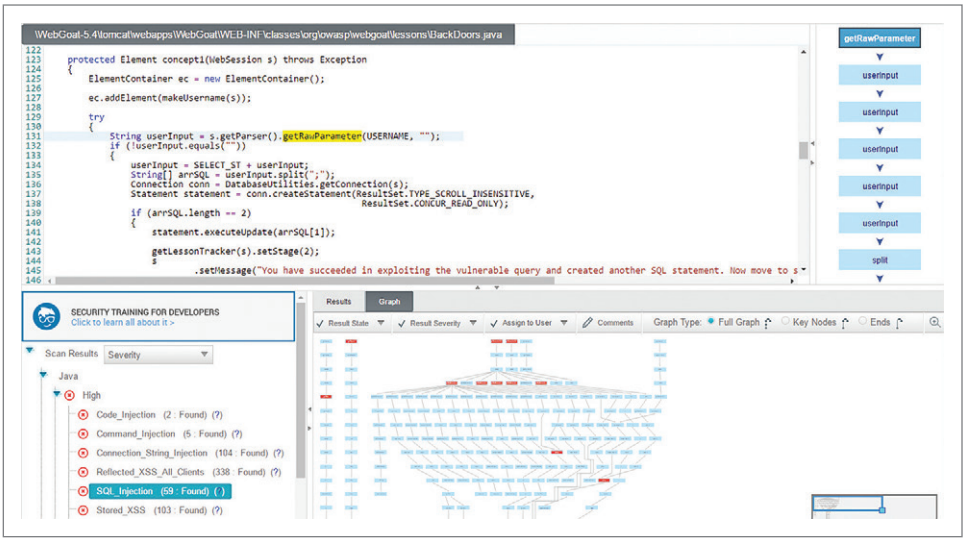


Figure 6 **Checkmarx Best Fix Location in Data Flow Graph**

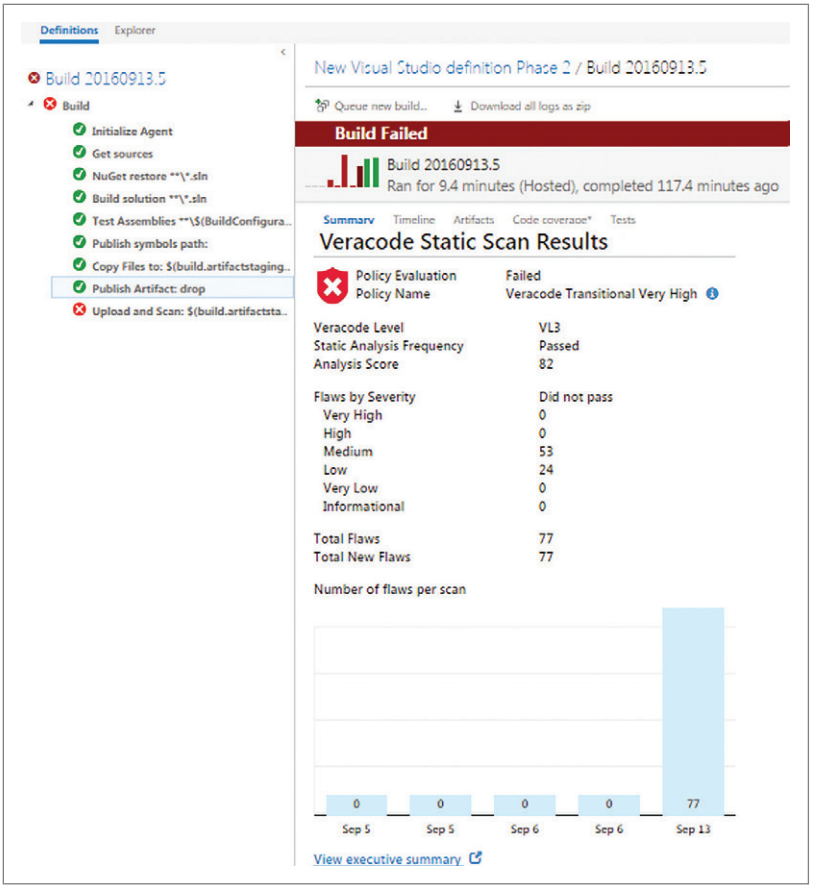


Figure 7 **Visual Studio Team Services Build Report with Veracode Scan Results**

Solution Spotlight: Veracode Extension for VSTS

The Veracode Application Security Platform is a SaaS that enables developers to automatically scan an application for security vulnerabilities. Veracode provides static application security testing (SAST), dynamic application security testing (DAST) and SCA, allowing development teams to assess both first-party code and third-party components for security risk (see **Figure 7**).

The Veracode VSTS extension (bit.ly/2dme4Vr) allows teams to configure continuous and automated assessment of their applications as a build or release step from their CI/CD pipeline. The build or release step can also be configured to automatically stop a build or release based on application security policy, allowing developers to integrate security testing into a fully automated CD pipeline. In addition, Veracode offers "Developer Sandbox" scans to provide results on an individual developer's changes before submission.

Why Integrate Security Scanning into Your DevOps Pipeline?

Because it's now practical. With the new extensions in the VSTS Marketplace, you can make security scanning a continuous part of your team's release pipeline. Unlike in the past, when scans were infrequent and generated a wall of issues right before a release, you can address warnings and errors as they occur. By addressing the security warnings in small batches—either with each PR or even daily—you can reduce the work in process and address component and code security continuously.

In order to optimize for speed, DevOps promotes the idea of shift-left. In other words, whatever is worth doing, is worth doing continuously as part of a DevOps workflow. When you make changes, make them and version them with code. This creates one source of truth: your source repo and trusted package management store. Whenever you update, update the single source of truth.

According to the 2016 "State of DevOps Report" (bit.ly/28NI32i) from Puppet, high performers, in addition to higher agility and reliability outcomes, also had better security outcomes. By integrating information security (InfoSec) objectives into the daily work of Dev and Ops, they spent 50 percent less time remediating security issues. If you integrate security into your DevOps pipeline, then you have an automatic way to create, test and deploy updates. This will shorten your time to remediate when new vulnerabilities are discovered. Security updates become just like other updates and can follow the same automated flow.

The practices make reuse safe. When you scan packages continuously, you can depend on them and know that you aren't picking up their vulnerabilities. Moreover, when new vulnerabilities are discovered in the wild, you can be notified immediately so security intelligence becomes actionable. You can pick up the new package versions, modify your code as needed, test and release—without needing to wait for attackers to discover your vulnerability.

DevOps started by breaking down silos between development and operations. Now you can break down the next wall, between DevOps and InfoSec. Rather than build up security debt that needs to be addressed too late, you can prevent it from creeping into the pipeline.

The combination of these benefits helps you go fast. By integrating security automation into your pipeline, you can use it as an accelerator. After all, the bad guys will go for easy targets. And as the old joke has it, if you're in a camping party in the woods and a bear appears, it's not the bear you need to outrun, but the bear's other potential prey. ■

Sam Guckenheimer *works in Visual Studio Cloud Services. He is the author of three books on agile development practices and an e-book on the Visual Studio Team Services journey to cloud cadence. Reach him at samgu@microsoft.com or on Twitter: @samguckenheimer.*

Jean-Marc Prieur *is a senior program manager at Microsoft envisioning and driving the delivery of experiences in Visual Studio and Visual Studio Team Services focused on controlling technical debt, including architecture analysis tools. Reach him at jmprieur@microsoft.com or on Twitter: @jm_prieur.*

Thanks to the following technical experts for reviewing this article: *Amit Ashbel, Michael Right, Joanna Rosenberg and Maya Rotenberg*

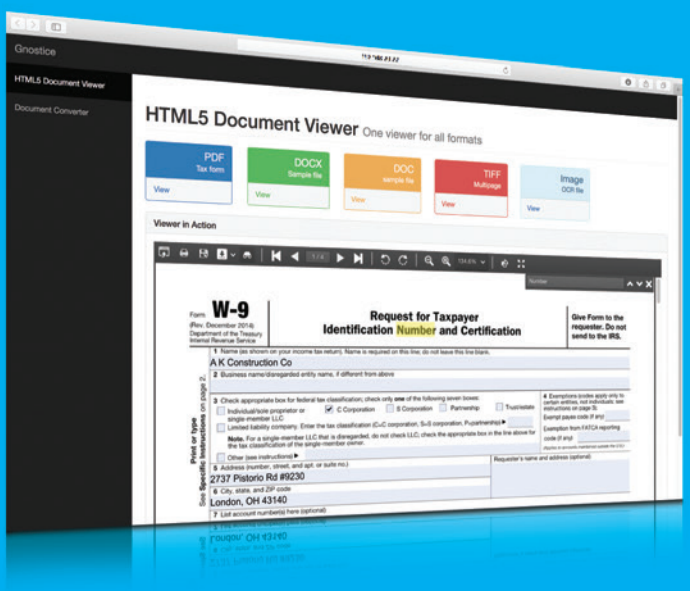
Document Technology for Everybody

PDF DOCX DOC RTF PPTX XLSX XLS TIFF MTIFF JPEG PNG BMP WMF EMF

HTML5 Web MVC WPF Win API API Java VCL FMX

Create | View | Print | Edit | Convert | Digitize

Get the Power of Great Design



Interactively Fill PDF Forms In Your Web App

- Responsive HTML5 control, automatically adjusts for Mobile, Desktop and Pad/Tablet.
- 100% Independent. No browser plug-ins required. No ActiveX.
- Single viewer for PDF, Office documents, text files and Images.
- Text Search, On-the-fly OCR on images, and more…
- Full-fledged client-side JavaScript to configure and perform all operations.
- TypeScript Support.

Royalty-Free Licensing

Gnostice™
Smart needs...Smarter solutions...

Download free trial from:
www.gnostice.com

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

7 LOCATIONS TO CHOOSE FROM

JOIN US

ROCK YOUR CODE TOUR 2017

LAS VEGAS

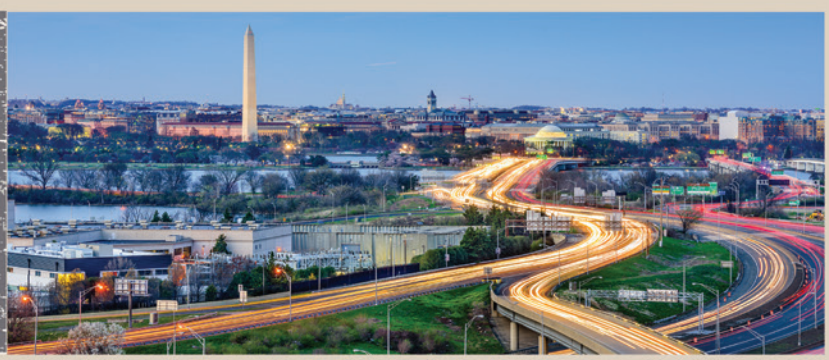
MARCH 13-17

AUSTIN

MAY 15-18

WASH DC

JUNE 12-15



SUPPORTED BY

Microsoft | Visual Studio | msdn magazine | Visual Studio MAGAZINE

PRODUCED BY

1105 MEDIA
YOUR GROWTH. OUR BUSINESS.

Microsoft Graph: Gateway to Data and Intelligence

Yina Arenas

A key strategy for Microsoft is to "reinvent productivity" by empowering developers to build smart, people-centric applications on the Microsoft platform so users can get more out of work and life.

The way we work is rapidly evolving. People seem to always be connected and users expect applications to help them with the task at hand—in the precise moment where they're needed, right where they are, and with the relevant context that can intelligently leverage the data available to make meaningful connections between people and information, and make productive use of the most valuable commodity: time.

Imagine an app that can look at your next meeting and help you prepare for it, such as providing useful profile information for attendees that includes not only their job titles, but also who they work with and information on the latest documents or projects on which they're working.

Or, imagine an app that not only has access to your calendar, but suggests the best times for the next team meeting.

How about an app that can get the latest sales projection chart from an Excel file sitting in your OneDrive and lets you update the forecast in real time, all from your mobile phone?

This article discusses:

- Empowering developers to build smart, people-centric applications
- Microsoft Graph is the gateway to data and intelligence in Office 365
- Enabling access to data in the consumer, commercial and sovereign clouds, as well as hybrid deployments.
- Microsoft Graph is a unified REST API with lots of developer resources to get started simple and fast

Technologies discussed:

Microsoft Graph, Office 365 and Microsoft Cloud Services, REST APIs

Or, how about an app that can subscribe to changes to your calendar, alert you when you're spending too much time in meetings, and provide recommendations for the ones you could miss or delegate based on how relevant the attendees are to you?

Or, how about an app that can help you sort out personal and work information on your phone, like pictures that should go to your OneDrive because they're your children's pictures versus pictures that should go to your OneDrive for business because they're pictures of receipts for an expense report.

All of these app examples can easily be powered by Microsoft Graph.

Unparalleled Opportunities for Developers

In an age of information abundance, we know that people are seeking integrated experiences to help them leverage many sources of data and connect information from multiple touchpoints in meaningful ways. Here's where Microsoft Graph is the key enabler, empowering developers to create powerful and personalized cloud-based apps that can transform the productivity landscape.

If you look at the massive amount of data available for developers, you'll see that there are about 850 million Outlook meetings scheduled per month and more than 100 million of those are Skype meetings. There have been 4 trillion e-mails sent to date using Office and hundreds of petabytes of data stored in Office 365. Users spend an average of two to three hours each day in Office and, now that Office apps are mobile, the mobile Office apps just surpassed 340 million downloads; this reach wasn't possible just two years ago!

Microsoft Graph: Easing Integration

Microsoft Graph (graph.microsoft.com) was created to meet user demand for smart contextual experiences and to ease the developer pain of integrating with Microsoft services one at a time to create them. Microsoft Graph is the unified gateway for developers to access all

LIVE! 360 | ORLANDO ROYAL PACIFIC RESORT AT UNIVERSAL ORLANDO | DEC 5-9

*REGISTER WITH DISCOUNT CODE L360NOV AND **SAVE $300!**

Office & SharePoint LIVE!

ON-PREMISE, CLOUD & CROSS-PLATFORM TRAINING

Taking Collaboration on the Road

Today, organizations expect people to work from anywhere at any time. Office & SharePoint Live!, provides leading-edge knowledge and training to administrators, developers, and planners who must customize, deploy and maintain SharePoint Server on-premises and in Office 365 to maximize the business value.

Whether you are a Manager, IT Pro, DBA, or Developer, Office & SharePoint Live! brings together the best the industry has to offer for 5 days of workshops, keynotes, and sessions to help you work through your most pressing collaboration projects.

* Must use discount code L360NOV

Scan the QR code to register or for more event details.

A Part of Live! 360: The Ultimate Education Destination

6 GREAT CONFERENCES, 1 GREAT PRICE

Visual Studio LIVE! | Modern Apps LIVE! | TECHMENTOR | Office & SharePoint LIVE! | SQL Server LIVE! | APPDEV TRENDS NEW!

SPLIVE360.COM

EVENT PARTNERS
Microsoft
Magenic

PLATINUM SPONSORS
Alachisoft
HYPERGRID
SMARTBEAR

GOLD SPONSORS
GOVERLAN
QuickBase
IDERA

SILVER SPONSOR
JET BRAINS

SUPPORTED BY
SharePoint
msdn magazine
Redmond MAGAZINE

PRODUCED BY
1105 MEDIA

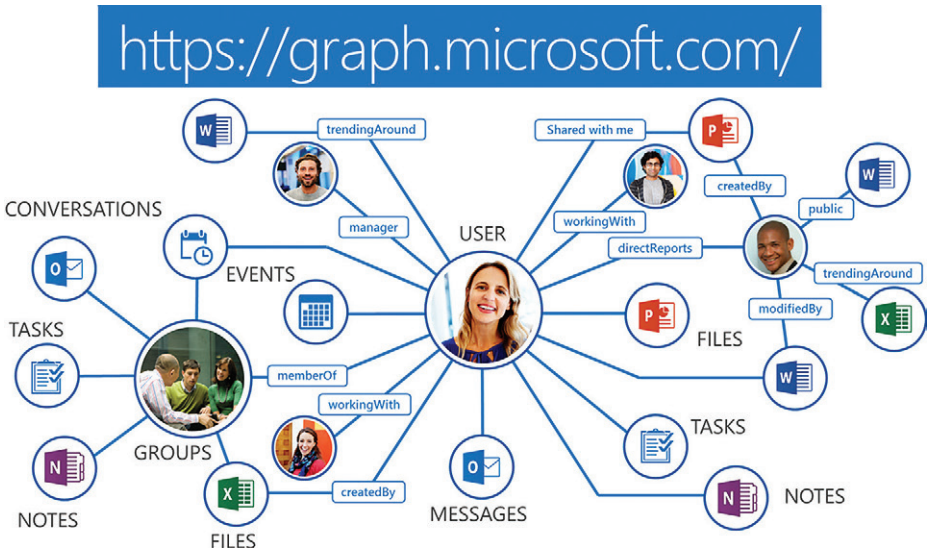


Figure 1 **Microsoft Graph Lets Apps Access Digital Work and Digital Life Data**

the data, intelligence and APIs housed in Microsoft's intelligent cloud including Exchange, SharePoint, Azure Active Directory, OneDrive, Outlook, OneNote, Planner, Excel and more. Microsoft Graph also includes calculated insights and rich relationships based on machine learning performed by its intelligent engine. All this is available through the same REST API endpoint, providing a much simpler developer experience across all of Microsoft's APIs by bringing them together in a single URI namespace with a single authentication story.

The reason why Microsoft Graph is so important is because the data that's in Office 365—the organizational hierarchy, the calendar, the mailbox, the files and so on—are cornerstones for organizations and how people get things done. And the easier your applications can leverage all that data, the smarter they can be and the better experiences they can provide for users.

As shown in **Figure 1**, Microsoft Graph aggregates information from multiple services and makes it available to the application in a single request. Developers can build user- and group-centered experiences that help users achieve more. An example: a productivity app that gives you the profile and picture of all the people you're about to meet and can tell you their organizational structure and the topics relevant to them. It doesn't matter where the data is stored; with Microsoft Graph you get a single endpoint to access it.

Microsoft Graph can also be used to traverse data across services to empower rich content scenarios. An example: an educational app that models classrooms around groups and lets the teacher track the documents students submit for their projects, see who modified the files, track their collaboration and progress, and have conversations around topics relevant to the class.

Access to Intelligence

Microsoft Graph surfaces intelligent insights by bringing together smart machine learning algorithms with a wealth of data and user behavior. Using Microsoft Graph, developers can access this relevant data to make applications contextual and smarter. For example: people picking controls powered by the People API in Microsoft Graph, where leveraging its fuzzy matching functionality, users don't have to remember how to spell some complicated last name and can get to the data just by remembering how it phonetically sounds. Imagine a sales app where the sales representative can quickly get to his customer's contact information and have it on the spot when needed. How many times have you forgotten how to spell a name and then have to scramble to find the contact based on other keywords?

Another example of Microsoft Graph intelligence is its ability to get trendind documents. Microsoft Graph listens to signals and activities such as file uploads, file views and modifications, e-mail conversations, and so on. Then it uses its intelligent engine to calculate rich relationships and inferred insights between people and documents. When a file becomes popular in your circle, Microsoft Graph creates a trending insight; this information then becomes available to power contextual experiences such as Delve in Office 365 and now is also available to developers as an API in Microsoft Graph.

Reach Millions of Users

Microsoft Graph is also the unified endpoint for consumer and commercial clouds. The lines between work and personal productivity are increasingly blurring. That app that can sort your personal and work photos to OneDrive and OneDrive for Business can be written with a single code base and a single app registration using Microsoft Graph. This means that developers can use this single endpoint and the same code to access personal data sitting in Outlook.com, Hotmail.com, Live.com, and other personal accounts in the Microsoft cloud, as well as with work and school data sitting in Office 365 and Azure Active Directory accounts. So, with Microsoft Graph, you use the same code with a single app registration and a single auth flow.

In addition to being the unified endpoint for consumer and commercial services, Microsoft Graph is also the unified endpoint for sovereign deployments. Microsoft announced the general availability of Microsoft Graph in China this year. That has strengthened the ISV ecosystem in China and empowered multi-national ISVs to build smarter apps for the Chinese market. As more sovereign clouds become available in other markets, Microsoft Graph becomes the gateway to access their data.

Furthermore, Microsoft Graph wants to close the programmability gap between the cloud and on-premises. Now in preview, Microsoft Graph can reach out to Exchange 2016 mailboxes sitting on-premises for customers with hybrid deployments. From the development perspective, the code can be agnostic to where the data is coming from and the same code can get data from a mailbox in the cloud (whether it's an Office 365 mailbox or an Outlook.com/Hotmail.com mailbox) or a mailbox on-premises. Microsoft Graph takes care or finding where the data lives and retrieving it for the app.

Microsoft Graph Is Core to the Office Platform

Microsoft Graph alters the landscape of work and productivity for IT, users and developers. For IT, apps are easier to deploy and manage due to their graph integration and because data access is secured. For users, apps are smarter, richer and contextual. For developers, Microsoft Graph brings tremendous value by shortening development time and making it simple to integrate with data and intelligence.

Today, innovative businesses are transforming work and productivity through Microsoft Graph. For example:

- **Zapier:** Uses the Microsoft Graph to tap into Excel data and let users create powerful "zaps" or personal workflows that automate data collection into Excel and integrate it with other cloud services. This is an integration that wasn't possible before Microsoft Graph and its Excel REST API.
- **Smartsheet:** Integrates with Microsoft Graph in Outlook to give the Outlook user the ability to interact with the sheets and projects in Smartsheet right from the Outlook experience while leveraging data from OneDrive and other services.
- **SkyHigh Networks:** Leverages Microsoft Graph to enable security teams to gain visibility into sensitive data, apply data loss prevention policy to users or groups, and identify or alert high-risk behavior.
- **Hyperfish:** A brand-new startup that enables organizations to automatically identify and populate missing user profile information quickly and easily. It turns blank people cards into rich cards that enable faster people connections.
- **Workday:** Uses Microsoft Graph to integrate with Office 365 groups such that when an employee starts a new position in the organization, all changes in Workday are automatically reflected in Office 365 groups and the employee gets immediate access to all events, conversations and documents.

The API

Microsoft Graph uses Web standards that enable any device capable of making an HTTP request to interact with it. It's a RESTful API that follows Microsoft REST API guidelines recently made public to the API community and available at bit.ly/2dzFp1a. Many portions of the Microsoft REST API guidelines evolved from standardization and rationalization exercises to unify the existing service APIs and their direct endpoints and schemas, so that they could participate in Microsoft Graph. I personally wrote the first proposal for naming conventions and casing that went into the guidelines and was part of long, internal API debates among more than 15 teams across Microsoft that collaborated in the creation of Microsoft Graph.

Microsoft Graph supports a rich set of query parameters such as select, filter, expand, and orderBy that can be used to specify and control the amount of data returned in the response. Microsoft Graph also has a growing set of SDKs for devices and services—so whether you're working on iOS, Android, or Universal Windows Platform (UWP) apps, creating a .NET Azure Web site, or building a service with Node.js, Python, PHP, or Ruby, you can quickly incorporate Microsoft Graph data into your application.

Getting Started with Microsoft Graph

So how can you leverage Microsoft Graph? Start by navigating to graph.microsoft.io. This will take you to the developer portal, where you'll find quick-start experiences that can bootstrap your development and, in less than five minutes, you'll have a working application in the platform of your choice calling Microsoft Graph.

At the Microsoft Graph developer portal, you'll also find documentation, complete API reference, a full suite of SDKs and code samples on a variety of platforms, and the Graph explorer. Using the Graph explorer, you'll be able to send requests to Microsoft Graph and inspect the response right away using your personal account, your work or school account, or even a demo account. **Figure 2** shows some sample requests showcasing the type of data that can be accessed using Microsoft Graph. Data can be read, created, updated and deleted using the APIs. You can easily try all of these requests and more using the Graph explorer.

Now I'll take a look at Microsoft Graph in more detail. You start by using the Getting Started link at graph.microsoft.io to create an ASP.NET MVC Web application that uses the Microsoft Graph .NET SDK to send mail on the user's behalf. Then you enhance it with an additional call to the OneDrive API exposed in Microsoft Graph to query for the user's recent files.

After navigating to the Get started page and interacting with the try experience, navigate toward the bottom of the page and select the ASP.NET MVC entry point.

The next step is to follow the links to register the app. Remember to copy the "secret" and save it as it won't be presented to you again. After this process, you'll get a .zip package with the project. Extract the files, open the project in Visual Studio, build it and run it. If you need to make updates or changes to the registered app you can make them at apps.dev.microsoft.com.

This quick-start experience grabs the code sample, inserts the application id and the secret in the web.config file and leaves the

Figure 2 **Sample Requests in Microsoft Graph**

Operation	Service Endpoint
Get my profile	https://graph.microsoft.com/v1.0/me
Get my files	https://graph.microsoft.com/v1.0/me/drive/root/children
Get my photo	https://graph.microsoft.com/v1.0/me/photo/$value
Get my mail	https://graph.microsoft.com/v1.0/me/messages
Get my calendar	https://graph.microsoft.com/v1.0/me/calendar
Get my manager	https://graph.microsoft.com/v1.0/me/manager
Get last user to modify file foo.txt	https://graph.microsoft.com/v1.0/me/drive/root/children/foo.txt/lastModifiedByUser
Get users in my organization	https://graph.microsoft.com/v1.0/users
Get group conversations	https://graph.microsoft.com/v1.0/groups/<id>/conversations
Get people relevant to me	https://graph.microsoft.com/beta/me/people
Get my tasks	https://graph.microsoft.com/beta/me/tasks
Get my notes	https://graph.microsoft.com/beta/me/notes/notebooks
Get files trending around me	https://graph.microsoft.com/beta/me/insights/trending

project ready for you to run. The running app is the resulting ASP.NET 4.6 MVC Web app that connects to a Microsoft work or school (Azure Active Directory) or personal (Microsoft) account using the Microsoft Graph API to send an e-mail. It uses the Microsoft Graph .NET SDK to work with data returned by Microsoft Graph.

Now that you have the project up and running, you'll add the calls to the OneDrive API to get the list of items that have been recently used by the signed-in user. This list includes items that are in the user's drives, as well as items she has access to from other drives.

The first step is to modify the web.config file to add the Files.Read permission scope to let the application read access to the user's OneDrive. In the web.config file, look for the ida:GraphScopes key in the appSettings element and add Files.Read to the value string. The next time the user launches the app, the service will identify the new scope and dynamically ask the user to consent for it:

```
<appSettings>
  ...
  <add key="ida.GraphScopes" value="User.Read Mail.Send Files.Read" />
</appSettings>
```

Next, you'll modify the controller. Open HomeController.cs in the Controllers folder; this class contains the actions that initialize the Microsoft Graph .NET SDK in response to the UI events. Add the method in **Figure 3** to get the top 10 recent OneDrive items.

Figure 3 **Acquiring Top 10 Recent OneDrive Items**

```
[Authorize]
// Get the items that are shared with the current user.
public async Task<ActionResult> GetMyRecentItems()
{
  try
    {
      // Initialize the GraphServiceClient.
        GraphServiceClient graphClient = SDKHelper.GetAuthenticatedClient();

        // Get the recent items.
        ViewBag.RecentItems =
          await graphClient.Me.Drive.Recent().Request().Top(10).GetAsync();
        return View("Graph");
      }
      catch (ServiceException se)
      {
        if (se.Error.Message ==
          Resource.Error_AuthChallengeNeeded) return new EmptyResult();
        return RedirectToAction("Index", "Error", new { message =
          string.Format(Resource.Error_Message, Request.RawUrl,
          se.Error.Code, se.Error.Message) });
      }
}
```

Figure 4 **Update the View Graph.cshtml in the Views Folder**

```
<h2>Recent OneDrive Items</h2>
@using (Html.BeginForm("GetMyRecentItems", "Home"))
{
  <div class="col-sm-12">
    <div class="form-group">
      <button class="btn btn-default">Get Recent OneDrive Items</button>
    </div>
  </div>
}
<div class="col-sm-12">
  <label for="recent-items">Recent Items</label>
  @if (ViewBag.RecentItems != null)
  {
    foreach (var item in ViewBag.RecentItems)
    {
      <div class="row">
        <div class="col-sm-4">@item.Name</div>
        <div class="col-sm-8">@item.CreatedBy.User.DisplayName</div>
      </div>
    }
  }
</div>
```

Notice the [Authorize] statement before the method. This is there to ensure this request will initiate a sign-in if the user isn't already signed in.

Finally, update the view Graph.cshtml in the Views folder to include a button that will trigger the request and divs to render the name of the item and the name of the user who created the item for each of the items in the returned list. Do this by appending the code in **Figure 4** at the end of the file.

The result is an ASP.NET 4.6 MVC Web app that connects to Microsoft Graph and can be used by users with Microsoft work or school (Azure Active Directory) accounts or users with personal (Microsoft) accounts to send an e-mail from their Outlook.com or Office 365 mailbox and access OneDrive or OneDrive for Business data. This is a simple example that illustrates how easy it is to get data from the Microsoft cloud using Microsoft Graph, whether that data comes from Azure Active Directory, SharePoint, OneDrive, Exchange, Outlook.com, Planner, OneNote, Excel, or other services.

Imagine the applications that you can build. Microsoft is working to increase the number of services and functionality available in Microsoft Graph, on deepening its understanding of user activity to make richer inferences and relationships, and simplifying the developer experience so you can access all of this data from a single connected API.

This is a massive opportunity for developers, who can tap into all this data and intelligence using the Microsoft Graph. The momentum behind Microsoft Graph since it launched in November 2015 speaks to this. Microsoft has seen a huge uptake in the number of registered apps, the organizations that consent to apps that use Microsoft Graph, and end-user active usage of those apps. Some of these apps built using Microsoft Graph are already reaching millions of active daily users.

Wrapping Up

With Microsoft Graph, developers are empowered to build smart, people-centric apps that can easily interact with data from all touch points of modern work. It enables developers to take advantage of the tremendous amount of data in Microsoft's cloud services—to build smarter apps and help people be more productive. Microsoft Graph exposes APIs, data, and intelligence across Office 365 and Azure Active Directory. Microsoft is building toward a near future where multiple graphs and all APIs throughout Microsoft contribute to, and are accessible through, a single unified gateway to the power of the Microsoft cloud. This translates into the ecosystem helping reinvent productivity by creating sticky experiences in all industry verticals. It opens a future where developers can re-shape health care, education, finance, law and many more industries; the possibilities are endless. I can't wait to see what you build using Microsoft Graph. ■

Yina Arenas *is a principal program manager at Microsoft and lead for Microsoft Graph. She's taking Office and Microsoft APIs from legacy and disjointed technologies to a new, unified API world. She lives in the Seattle area with her husband and their three energetic boys and actively leads and participates in activities that grow, retain, and empower women in technology. Find her on Twitter: @yina_arenas.*

Thanks to the following Microsoft technical experts for reviewing this article: *Agnieszka Girling, Gareth Jones and Dan Kershaw*

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

LAS VEGAS
MAR 13-17 2017
BALLY'S, LAS VEGAS, NV

EVENT PARTNERS

Microsoft Magenic

SUPPORTED BY

Visual Studio msdn magazine Visual Studio MAGAZINE

PRODUCED BY

1105 MEDIA
YOUR GROWTH. OUR BUSINESS.

INTENSE TRAINING FOR DEVELOPERS, ENGINEERS, PROGRAMMERS, ARCHITECTS AND MORE!

Track Topics include:

- Visual Studio / .NET Framework
- JavaScript / HTML5 Client
- Modern App Development
- Mobile Client
- Software Practices
- Database and Analytics
- Angular JS
- ASP.NET / Web Server
- Agile
- ALM / DevOps
- Cloud Computing
- Windows Client

Sunday Pre-Con Hands-On Labs

Choose From:

- Angular
- Azure
- XAML

NEW! ONLY $595

SPACE IS LIMITED

Register by December 16 and Save $500!*

Use promo code VSLNOV2 Scan the QR code to register or for more event details. *SAVINGS BASED ON 5 DAY PACKAGES ONLY.

ROCK YOUR CODE TOUR 2017

LAS VEGAS	AUSTIN	WASH. DC	REDMOND	CHICAGO	ANAHEIM	ORLANDO
MARCH 13-17	MAY 15-18	JUNE 12-15	AUG 14-18	SEPT 18-21	OCT 16-19	NOV 13-17

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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

vslive.com/lasvegas

Big Data Development Made Easy

Omid Afnan

There's a big buzz around the concept of Big Data in business discussions today. Organizations as diverse as power utilities, medical research firms, and charitable organizations are demonstrating the use of large-scale data to discover patterns, deduce relationships, and predict outcomes. Online services, and the businesses operating them, were the early development grounds for Big Data and remain its most eager adopters. Scenarios such as product recommendation, fraud detection, failure prediction and sentiment analysis have become mainstays of the Big Data revolution.

Big Data is often defined as having the characteristics of the "three V's"—large volume, high velocity and variety. While massive volume is the common association for Big Data, the need to take action on data arriving in near-real time and dealing with a large and changing variety in format and structure are equally defining for Big Data. In fact, any one of these characteristics present in data can be enough to create a Big Data scenario with the others often following soon after. The desire—and increasingly the necessity—to derive value from data that has the three V's is the motivation behind the increased interest in Big Data.

While data science, statistical analysis, and machine learning have all enjoyed renewed attention and growth, the use of Big Data has truly been unlocked by underlying advances in distributed computing. This includes the development of software stacks that can stitch together computing clusters from inexpensive commodity hardware. Associated compilers and job schedulers make it possible to distribute a computing workload across such clusters. This puts the computations close to where the data is stored and aggregates many servers to get higher performance. These big data platforms were developed by companies such as Microsoft, Yahoo and Google for internal use, but have become available for public use through platforms like Azure Data Lake (ADL), Hadoop and Spark.

It comes as no surprise, then, that getting started with Big Data means an investment in building up a data platform based on these new technologies. At the enterprise system level, this means the introduction of a "data lake" in addition to the traditional data warehouse. The data lake is based on the concept of storing a wide variety of data types on a very large scale and in their original formats. Unlike the data warehouse model, data is first captured without any cleaning or formatting, and is used with a late-binding schema. This kind of architecture requires the adoption of the new software platforms, ADL, Hadoop or Spark. For the data developer, it means having to learn new programming models and languages while dealing with more complex execution and debugging situations.

Fortunately, Big Data systems have already come a long way. Let's say you want to build a product recommendation system with information collected from the online shopping site you operate. Your plan is to send e-mails to shoppers to promote new products based on previous shopping patterns. You might also want to recommend products while the customer is shopping on the site based on what other customers did. Both scenarios fit well with Big Data techniques. Until recently you'd have to start by selecting a particular Big Data stack, designing and procuring the cluster hardware, installing and tuning the software, and then you could start to develop the code for collecting, aggregating, and processing the data you have. Based on your investment, you would likely feel locked into the particular hardware and software stack you chose.

While stacks like Hadoop and Spark have become easier to install and manage, the coming of Big Data in cloud services is an even bigger game changer. As a cloud service, ADL provides both the storage and computation power needed to solve Big Data problems. More important, one of its key principles is that Big Data development needs to be easy.

This article discusses:

- Developing code for Big Data analytics
- Debugging distributed queries with parallel scale-out
- Using U-SQL to move into Big Data development

Technologies discussed:

Azure Data Lake, Visual Studio IDE, U-SQL

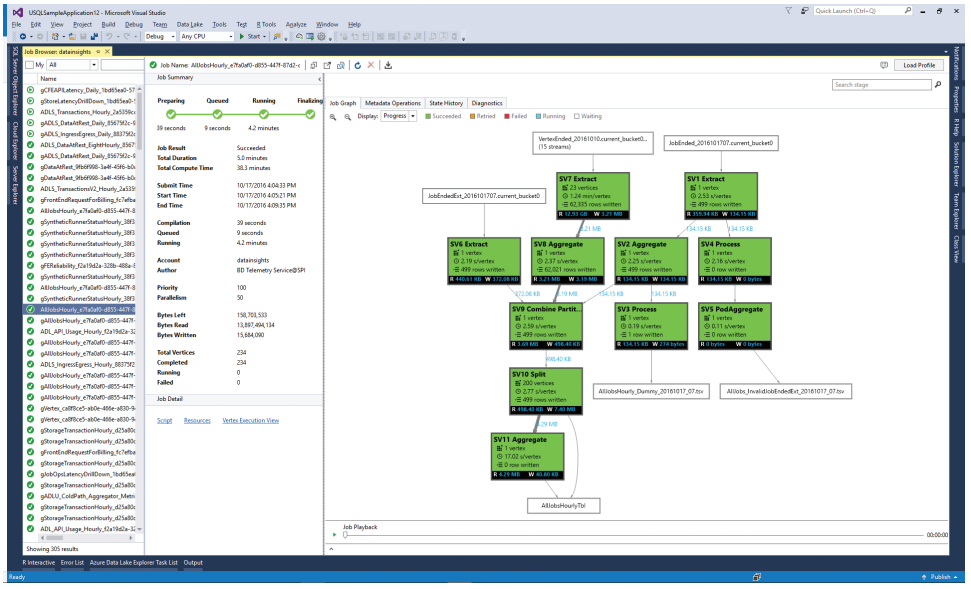


Figure 1 **U-SQL Job Graph Viewed in Visual Studio**

Big Data Gets Easier

ADL is a cloud-based environment that offers a Big Data query service. That means you don't have to set up and manage any hardware. When you're ready to do Big Data development, you simply create ADL accounts on Azure and the service takes care of allocating storage, computation and networking resources. In fact, ADL goes further and abstracts away the hardware view almost completely. ADL Storage (ADLS) works as an elastic storage system, stretching to accommodate files of arbitrary size and number. To run queries and transformation on this data the ADL Analytics (ADLA) layer allocates servers dynamically as needed for the given computations. There's no need to build out or manage any infrastructure—simply ingest data into ADLS and run various queries with ADLA. From a developer's viewpoint, ADL creates something that looks and behaves a lot like a server with infinite storage and compute power. That's a powerful simplification both for setting up the environment and understanding the execution model.

The next simplification comes from the U-SQL language. U-SQL is a unique combination of SQL and C#. A declarative SQL framework is used to build a query or script. This part hides the complexity that arises from the actual distributed, parallel execution environment under the covers. You, the developer, don't have to say how to generate a result. You specify the desired outcome and the system figures out the query plan to do it. This is the same as what happens with SQL on an RDBMS but different than other Big Data stacks where you may have to define map and reduce stages, or manage the creation of various types of worker nodes. The compiler, runtime and optimizer system in ADL creates all the steps for generating the desired data, along with the possible parallelization of executing these steps across the data.

Note that a query can actually be a very complex set of data transformations and aggregations. In fact, the work of developing Big Data programs with U-SQL is writing potentially complex scripts that re-shape data, preparing it for user interaction in BI tools or further processing by other systems such as machine-learning platforms. While many transformations can be done with the built-in constructs of a SQL language, the variety of data formats and possible transformations requires the ability of add custom code. This is where C# comes into U-SQL, letting you create logic that is customized, but can still be scaled out across the underlying parallel processing environment. U-SQL extensibility is covered in-depth in Michael Rys' online article, "Extensibility in U-SQL Big Data Applications," (msdn.com/magazine/mt790200), which is part of this special issue.

The addition of support in development tools such as Visual Studio provides another critical simplification: the ability to use familiar tools and interaction models to build up your Big Data code base. In the remainder of this article I'll cover the capabilities provided for U-SQL programming in Visual Studio and how, together with the previously mentioned capabilities, they provide a major shift in the ease of use for Big Data overall.

Developing U-SQL

To make Big Data development easy, you start with being able to construct programs easily. The fact that U-SQL combines two very familiar languages as its starting point makes it quite easy to learn and removes one of the barriers for adopting Big Data. In the case of development tools such as Visual Studio, it also makes it easy to reuse familiar experiences such as the built-in support for C# development and debugging. The Azure Data Lake Tools for Visual Studio plug-in (bit.ly/2dymk1N) provides the expected behavior around IntelliSense, solution management, sample code and source control integration.

Because ADL is a cloud service, the Visual Studio tools draw on the integration with Azure through the server and cloud explorers. Other experiences must be extended or changed: IntelliSense for U-SQL includes an understanding of the tables and functions defined in the cloud within ADLS, providing appropriate completions based on them. Another key set of features is found in **Figure 1**, which is a representation of the query execution graph that results when a query is compiled for execution in the distributed, parallel cloud environment.

When you develop a U-SQL script, you start it as you do in other languages. You create a new project where you can add U-SQL type files. You can insert code snippets into query files to get you started, or access sample code from a sample project. Once you've written some code, it's natural to compile and run your code to find and fix any errors and test your logic. This then becomes the tight loop for your work until the code reaches an overall functional level.

This work loop has some complications for Big Data. Big Data code runs in Big Data clusters, so the existing situation is for developers to run their code in a cluster. This often requires the

setup and maintenance of a development cluster. It can also take longer to go through the full cycle of submitting, executing and receiving results. The tight loop in this case becomes too slow. In some technologies a "one box" install of a cluster is available. This probably requires special installation tools, but it might be installable on the development machine you use.

U-SQL simplifies this situation by providing a compiler and runtime that can execute on a local machine. While the execution plan (including such things as degrees of parallelism, partitioning and so on) for a local machine and the cloud-based environment are likely to be quite different, the computation and data flow graph will be the same. So, while the performance of the local run won't be comparable to that in the cloud service, it provides a functional debugging experience. The necessary tools for local run are installed with the Azure Data Lake Tools for Visual Studio plug-in and are available immediately. They can also be installed as a NuGet package and run from the command line for automation purposes.

The local execution environment looks like another ADLA account. You can see it in the Server Explorer window under the Data Lake Analytics section, where your cloud-based accounts are listed. On submission dialogs it's shown as an option on the list of target accounts with the label "(Local)." This local account can have all the child nodes that real accounts have. This means you can define metadata objects such as databases, tables and views. You can also register C# libraries here to support execution of U-SQL scripts that have user-defined code. This is necessary to provide parity with the cloud environment for full testability of your U-SQL code.

The U-SQL development loop then looks very much like other languages. You can create a U-SQL project and once enough code is ready, you can compile to find syntax errors. You can also run locally through the Debug (F5) and the Run without debug (Ctrl+F5) commands in Visual Studio. This will expose runtime errors such as data-parsing problems during file ingestions (the EXTRACT command in U-SQL), a very common debug case. At any point you can switch to submitting the code to run in your ADLA account in Azure. Note that this will incur charges based on the overall time for the query.

Managing Data

Given that datasets for Big Data scenarios are often too big to use on a development machine, it becomes necessary to manage data for test and debugging purposes. A common way to handle this is to refer to data files by relative paths. The U-SQL compiler interprets relative paths from the root of the default storage for a given execution environment. Each ADLA account has a default storage account associated with it (you can see this in the Server Explorer window). For execution in the cloud, file paths are found in this root. When executed locally, paths are searched under the global data root directory shown under Tools | Options | Azure Data Lake.

For development, a common practice is to specify a relative path to a file that exists both locally and in the cloud. The script can then be run without modification both locally or as submitted to ADLA. In the case that the input data already exists in Azure, you can download a portion of the file. The ADL experience in Visual Studio lets you do this by navigating to the file from the Job Graph or File Explorer (from the context menu on the storage accounts in Server Explorer) and selecting the download option. If the data doesn't yet exist, then a test file must be created. With the data in place, the local development loop can proceed as before.

Debugging with User-Defined Code

The fact that U-SQL lets you use C# to define customer code introduces additional debugging capabilities. Briefly, C# extensions must be registered in a database in the ADLA account where the related query will be executed. As mentioned earlier, this can also be done in the local run scenario using the "(Local)" account. The general case is that you create a separate C# class library project (there's a project type for this under the ADL area) and then register it.

There's also another easy way to define user code and have Visual Studio manage the registration for you. You'll notice that in a U-SQL project, each file automatically has a codebehind file associated with it. This is a C# file where you can add code for simple extensions that aren't meant to be shared with other projects. Behind the scenes, Visual Studio will manage creating a library, registering and unregistering for you during submission of the query for execution. Again, this works against a "(Local)" account, as well.

Regardless of how the user-defined code is created, it can be debugged like other C# code during execution in the local environment. Breakpoints can be set in the C# code, stack traces examined, variables watched or inspected, and so on. The Start Debugging (F5) command kicks off this capability.

Debugging at Scale

Up to this point, I've discussed the capabilities that let you build U-SQL code in projects, specify data sources, compile and run locally, and step through C#. If you're thinking, "That sounds like every other language I code in," that's great! I mentioned that the goal here is to take an inherently complex distributed, parallel-execution environment and make it look like you're coding for a desktop app. I've also talked a bit about how to manage data sources and C# extensions in your code between local and cloud environments. Now, let's talk about something unique to the debugging of Big Data jobs at scale (that means running in the cloud).

If you've followed the approach of developing your code and debugging on your local machine, then at this point you've probably figured out any logic errors and are getting the outputs you want. In other words, your business logic should be sound. With Big Data, though, you'll have massive amounts of data and the format of the data is likely to change. Remember that in data lake architecture, you store data in its native format and specify structure later. This means data can't be assumed to be well-formatted until after you've processed it to be that way.

Now, when you run your tested code at scale in the cloud and try to process all of your data, you'll see new data and might start to hit problems you didn't find before. Also, your query has been compiled, optimized and distributed across possibly hundreds or thousands of nodes, each of which execute a portion of the logic on a portion of the data. But if one of those chunks of work fails in an unrecoverable way, how can you figure out what happened?

The first way that U-SQL and ADLA help you with this is to manage error reporting as you'd expect a job service would. If an exception

occurs outside of user code, then the error message, accompanying stack trace, and detailed data are collected from the offending node and stored against the original job (query submission). Now, when you view the job in Visual Studio or the Azure Portal, you'll immediately be shown the error information. No need to parse through log or stdout files to try and decrypt the error location.

An even more interesting and common case is when you have custom code and the failure happens there. For example, you're parsing a binary file format with your own Extractor and it fails on a particular input halfway through the job execution. In this case, ADLA again does a lot of work for you. For the parts of the query execution graph that succeed, neither the code image nor the intermediate data is kept in the system. However, if a vertex (an instance of a node in the execution graph) fails with an error in the user code portion, the binary executable and the input data are kept for a period of time to allow debugging. This feature is integrated with Visual Studio, as shown in **Figure 2**.

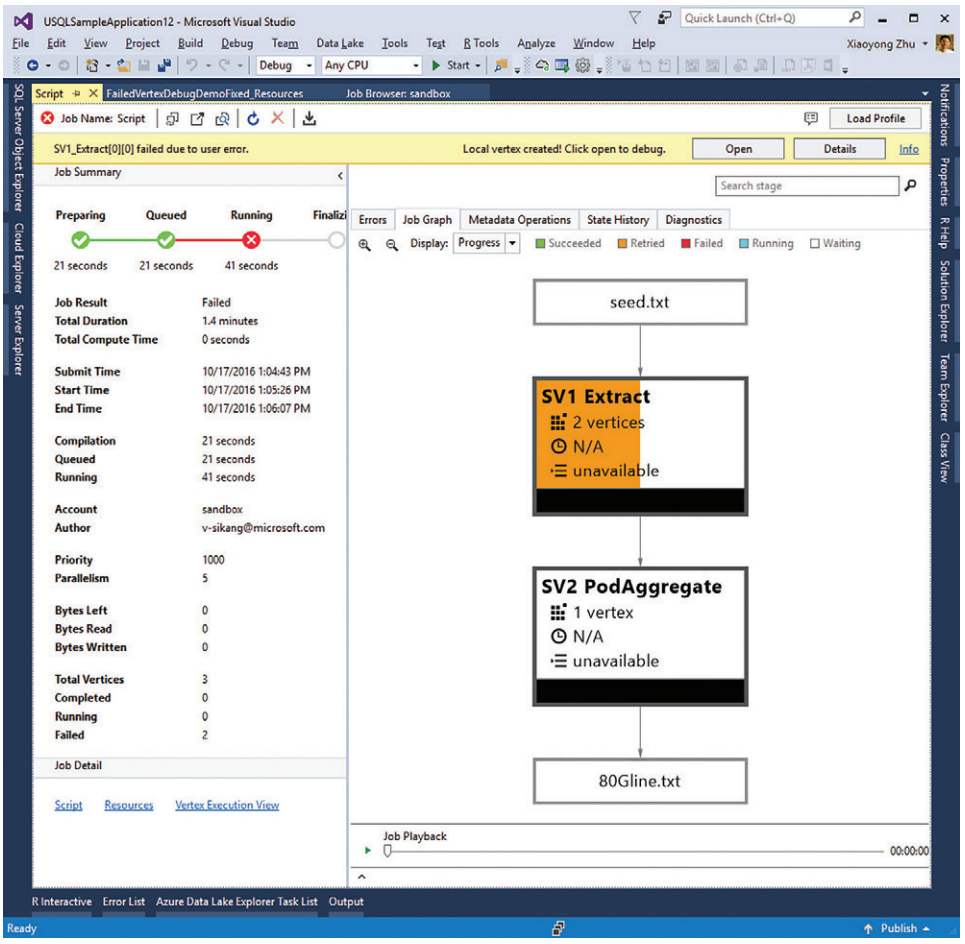


Figure 2 **Debugging a Failed Vertex Locally**

Clicking the debug button starts a copy of the binary, resource and data files from the failed vertex to your local machine. Once the copy is downloaded, a temporary project is created and loaded in a new instance of Visual Studio. You now have the executable version of the failed node and can do all the normal debugging you might expect, such as running to the exception, putting breakpoints in the C# code, and inspecting variables. Remember that the individual vertices in the clusters used to run Big Data jobs are actually commodity servers and are likely to be similar to your development machine. Because you also have a local U-SQL runtime on your machine, this capability becomes possible.

Once you have debugged the problem on your local machine, you'll update your source code separately. The project and code shown in your debug instance are artifacts from a previously run query and any changes you make are local. If you have a registered library of C# code, then you'll have to rebuild and update the library in ADLA. If you made changes to your U-SQL script or codebehind files, then you must update your project.

Wrapping Up

Big Data platforms have been highly specialized systems that required learning new concepts, models and technologies. Most early adopters had to go under the hood to learn the inner workings of these systems, first to set them up and then to be able to reason about programming in those environments. While the field has moved forward to the point that installing these platforms has become simpler, a bigger shift is underway. The opportunity today is to leapfrog to a Big Data service model in the cloud where the system abstraction is at a higher level. While this makes the setup of a Big Data environment trivial, an even more impactful outcome is that the development model is immensely simplified.

The combination of Azure Data Lake and U-SQL simplifies the execution model, programming paradigm, and tools used to develop Big Data queries and applications. This has the dual effect of enabling more developers to get started with Big Data, and for developers to build more complex Big Data solutions more quickly. ADL is supported by a large set of analytics services in Azure that support workflow management, data movement, business intelligence visualization and more. While U-SQL is the best place to start for Big Data application development, look to these other services as your needs grow. ■

Omid Afnan *is a principal program manager in the Azure Big Data team working on implementations of distributed computation systems and related developer toolchains. He lives and works in China. Reach him at omafnan@microsoft.com.*

Thanks to the following Microsoft technical expert for reviewing this article: *Yifung Lin*

Build User Experiences

UI Controls and Developer Productivity Tools for Delivering Apps Across All Platforms and Devices

Spread
Versatile Spreadsheet Data and UI Components
spread.grapecity.com

Wijmo
A New Generation of JavaScript Controls
wijmo.com

Xuni
Cross-Platform Native Mobile UI Controls
goxuni.com

ComponentOne Studio
Powerful, modular .NET Controls for Visual Studio
componentone.com

ActiveReports
Reporting Platform for Essential Business Needs
activereports.grapecity.com

The GrapeCity family of products provides developers, designers, and architects the ultimate collection of easy-to-use tools for building sleek, high-performing, feature complete applications. With over 25 years of experience, we understand your needs and have developed a comprehensive line of products that includes innovative UI controls; cross-platform data visualization controls; and enterprise-level reporting, analysis, and spreadsheet controls for Windows, web, and mobile platforms.

Learn more and receive free 30-day trials at
tools.grapecity.com

GrapeCity®

Visual Studio Microsoft Partner