THE MICROSOFT JOURNAL FOR DEVELOPERS

MAY 2017 VOL 32 NO 5

msdn magazine

DevExpress®

Your Next Great Desktop App Starts Here

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

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

devexpress.com/try

THE MICROSOFT JOURNAL FOR DEVELOPERS

MAY 2017 VOL 32 NO 5

msdn magazine

COLUMNS

Microsoft


TRAFFIC BY REGION
CONVERSIONS
VISITS
SPEND
UNIQUE VISITORS
2.3M
CONVERSION RATE
10%
CAMPAIGN HEALTH
20%
TOTAL
200K
CONVERSIONS
SALES DASHBOARD
MONTH TO DATE 9/1 - 9/16
$303,876
Actual
$340K
Forecast
REGION | DIVISION
REGION TARGET FORECAST
Americas $873,030
Asia $580,200
Africa $359,876
Australia $200,387
Europe $100,876
TOTAL $2,114,369
$2,114,369
LANDSCAPE VIEW FOR MORE DETAILS
SALES HISTORY
upcoming
Johnson
Lasa Ap
Fivel Ma
Nova No
Haddad
Hancock
starting 4/12 Nova No
starting 4/15 Haddad
starting 5/10 Hancock
hours worked
3/2 - 3/8 (this week)
tasks by resource
Prototype Interactions
Design Research
Oversee Prototyping
Bind Data to Server
Review Prototypes
Review Prototypes
Projected End 3/13
Projected End 3/13
Projected End 3/13
Projected End 3/12
Projected End 3/10
Projected End 3/13
90 hours
55 hours
67 hours
60 hours
35 hours
10 hours
0.56 (-2.12%)
1Y 2Y 5Y 10Y MAX
Stock Price
CHART
Daily Range
Annual Range
Open
Volume
Adobe Systems Inc
2013-07-26 $47.64
Microsoft Corporation
2013-07-26 $31.62
Intel Corporation
2013-07-26 $23.26
Jun 2013
Jul 2013
Aug 2013
Sales Volume

msdn magazine

MAY 2017 VOLUME 32 NUMBER 5

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

BPA WORLDWIDE

100 1906 2006 AMERICAN BUSINESS MEDIA

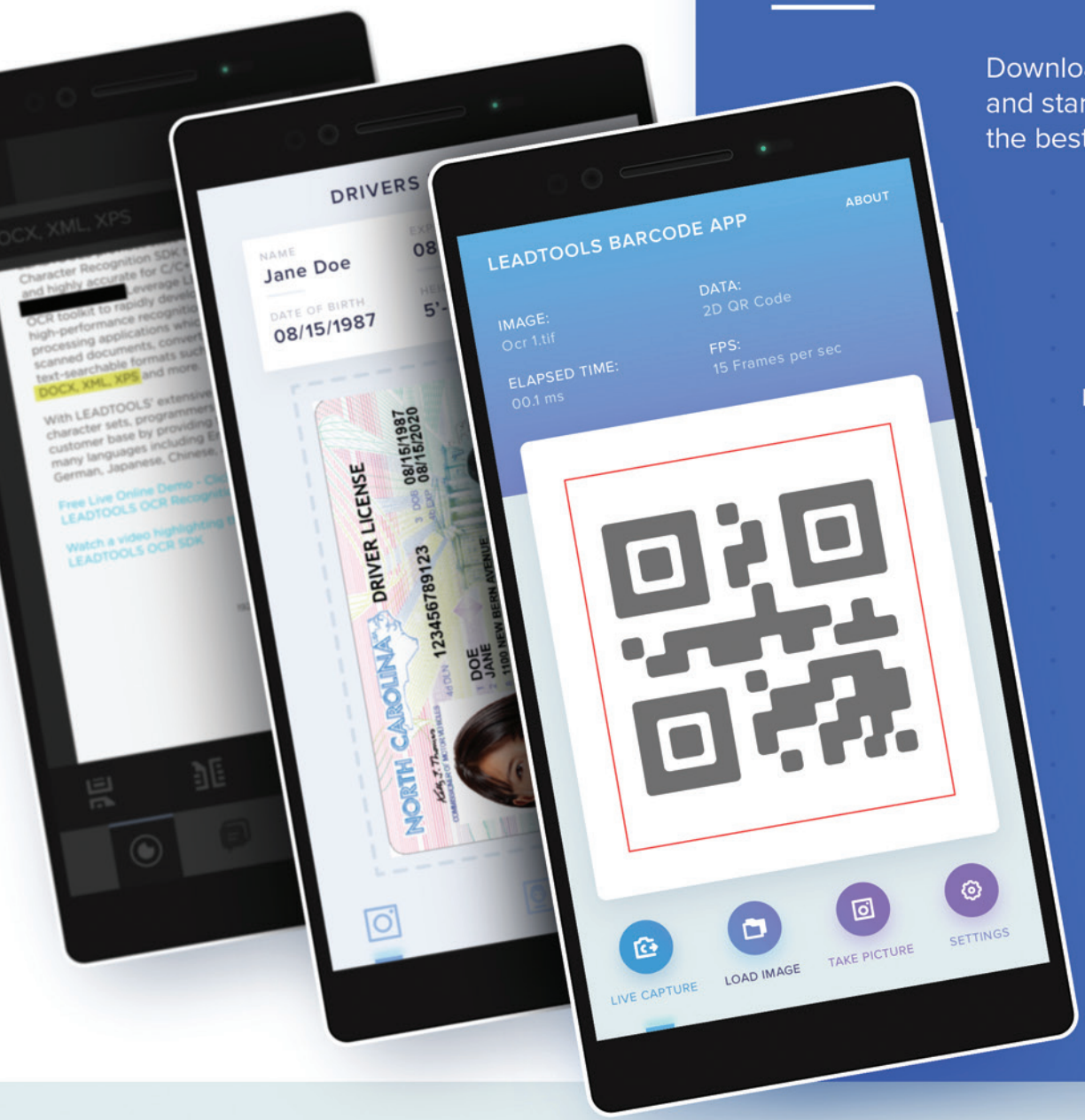
WPA Western Publications Association

LEADTOOLS®

THE WORLD LEADER IN IMAGING SDKs

WORLD-CLASS MOBILE RECOGNITION SDK

Quickly and accurately extract data from any document or image with LEADTOOLS. Developers using the SDK easily add advanced OCR, Barcode, Forms, Driver's License, Passport, Check, and Credit Card recognition functionality into their applications.



Download the SDK today and start developing with the best in:

- RELIABILITY
- ACCURACY
- SPEED

Fully-functional sample apps built with LEADTOOLS are also available for download from **Google Play**, **App Store**, and **Microsoft Store**.

DOWNLOAD OUR 60 DAY EVALUATION
WWW.LEADTOOLS.COM

LEAD TECHNOLOGIES INCORPORATED

SALES@LEADTOOLS.COM
800.637.1840

Editor's Note

MICHAEL DESMOND

Build Quality

The Microsoft Build 2017 conference kicks off on May 10 in Seattle, where more than 5,000 developers will gather to learn about Microsoft's plans to advance its development strategy and platforms. From frameworks like the .NET Framework, Universal Windows Platform (UWP) and ASP.NET Core to programming languages such as C#, the event offers a glimpse at the direction Microsoft will take over the year to come.

We've seen big things unveiled and promoted at Build over the years, from Windows 8 in 2011 to the release of the Microsoft Bot Framework at Build 2016. I don't expect this year to be any different. Microsoft is famously tight-lipped about its intentions ahead of each Build show—in fact, session schedules sometimes aren't released until the day before the event—but here's what I'm told to expect from this year's Build conference.

First, Microsoft will keep pushing productivity enhancements for developers working with Azure, Visual Studio, Xamarin and Windows. This is something that was big at the Connect(); event in New York City in November last year, as Microsoft pitched its tooling as a one-stop shop for devs coding not just for Windows and related platforms, but for Android, iOS and macOS.

In terms of overarching themes for Build 2017, areas of focus will include streamlining the creation of app experiences that immediately engage users with beautiful UIs and natural inputs, and delivering those experiences to users in a safe and reliable way. Team collaboration and connectedness will also be top of mind, leveraging the UWP to streamline and improve the dev experience. Finally, Microsoft will focus on connecting screens and experiences, leveraging a common service layer and code base to remove barriers.

You can expect plenty of guidance for key Microsoft platforms, including .NET Core and ASP.NET Core, Cognitive Services and Azure Machine Learning. And keep an eye out for the Bot Framework, which was launched at Build 2016. It's not unusual for Microsoft to follow up a Build reveal by using the subsequent Build event as a platform to refine, reinforce and reinvigorate the product launch. This is something we saw with Windows 8 in 2011 and 2012, for example, and we could see it with the Bot Framework this month.

Team collaboration and connectedness will also be top of mind, leveraging the UWP to streamline and improve the dev experience.

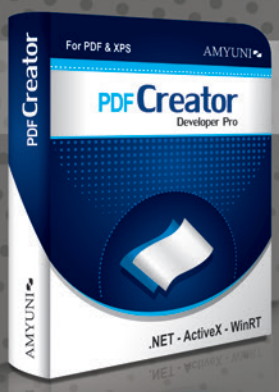
This month's issue of *MSDN Magazine* focuses on many of the platforms and technologies that will be getting attention at Build 2017. Alessandro Del Sole's lead feature, "Cross-Platform Code Generation with Roslyn and .NET Core," explores the powerful .NET Compiler Platform and its ability to transform cross-platform programming. Victor Mushkatin and Sergey Kanzhelev dive into the App Insights telemetry tool, while Maarten Van De Bospoort orchestrates a content-filtering solution employing the Bot Framework, Cognitive Services and Azure Machine Learning. And look for Dino Esposito's Cutting Edge column, which kicks off a multi-part exploration of ASP.NET Core.

By this time next month, we'll all know a lot more about Microsoft's plans for software development. Even if you're not among the fortunate 5,000 invited to Build 2017, you can still take part. Attend the daily keynote and technical sessions online at aka.ms/build-2017-msdn, and tune in to see the latest news and announcements from the event.

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

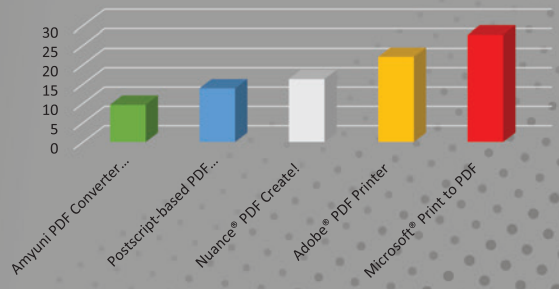
Complete Suite of Accurate PDF Components

- 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

Acing Interviews: Displaying Technical Skill Is Only Half the Battle

I recently got a question from a reader: "What are the skills required to crack the software engineering interview, other than technical skills?"

This is a perceptive question, because it recognizes that technical skills alone aren't enough to get an offer. Even if you're great technically, excellent behavioral performance can boost your offer, your bonus and the level at which you're hired. Here are the five major skills to succeed in behavioral interviews as an engineer:

Skill #1: Dynamic Story Shifting

This is the ability to perceive mid-answer how the interviewer is responding—frowning, eye-rolling, smiling, laughing or looking confused.

When you answer questions—whether technical or not—your goal isn't to make your point in the fastest time possible. It's to take the interviewer along on your journey to the offer, for as long as you can.

Sometimes, this means you go against your preparation.

For example, once you start answering a question on a "greatest challenge," you may quickly realize from an interviewer's skeptical frown that she's just not buying that your project was challenging. In such a case, you can start adding gory details to make the story seem more challenging, play up the actions you took and identify every positive result.

Skill #2: Active Questioning

This means you ask questions that are strategic and guide the interviewer toward the topics about which you're most passionate.

As an example, consider the "Tell me about yourself" question. A big mistake candidates make with this question: When they finish the answer, they leave it to the interviewer to ask yet another question!

The "Tell me about yourself" question is an opportunity for you to control the flow of the interview itself. After your answer, you can ask the interviewer a strategic question, such as: "I've read the recent posts on the company blog. It looks like you just revamped your entire product. How did that affect the area you work on?"

This relieves pressure from the interviewer (so they don't feel like they have to keep asking all of the questions). It also buys you time, gives you some additional intelligence about the job, and builds rapport.

Skill #3: Perspective Recalibrating

You can reorient, rephrase and restructure your answers depending on if you're talking to a manager, a vice president, a product manager, a fresh new engineer, a senior engineer or an architect.

You can ask yourself questions like:

- How technical does she seem to be?
- What characteristics would she most likely judge me on, considering her line of questioning and her title?
- Is he even trying to question me, or is he now trying to sell me on this position or company?
- From his perspective based on his role, what would make him feel the most comfortable and secure?

With this perspective, you can intuitively answer with either technical depth or strategic breadth. You could describe a project with deep technical details to a senior engineer. But you'd describe the same project—emphasizing different details, telling a different story—to a vice president of engineering.

Skill #4: Power Ceding

Cultivate a spirit of surrender throughout the interview, no matter how much better you may feel than the interviewer.

Sometimes, you might intuitively know that the people you're talking to are not as smart, or not as qualified or not as perceptive as you are. If you feel entitled, you'll subconsciously assume power in the interview, and start to treat them with condescension or haughtiness without even meaning to.

In such moments, remember that you don't have a job offer from this company yet. Even if you don't *want* to work there, a job offer will make you more confident with other companies.

Skill #5: Consciously Dissociating

Nurture an attitude of detachment to the results, by focusing primarily on what *you* can do best. *Full effort is full victory*, regardless of whether you get an offer.

If you succeed in an interview, it's *not* a grand accomplishment. An interview isn't something you'll remember on your deathbed. It just means that you're at a stage of development and growth where you could succeed in this interview. It's the beginning of another step in your life's journey.

Similarly, if you don't succeed in an interview, that's OK, too—you've done your best, knowing what you know now. Even if you're rejected, it's merely an opening for something else.

I don't tell people, "It'll all work out," because I don't know what "working out" looks like for everyone. I can only say that in my own case, there's a magical beauty to the course of my life. When interviews haven't worked out, they've laid the foundation for a grander purpose. I wouldn't be a career coach today if I hadn't failed at so many interviews and learned through my own personal experience. ■

KRISHNAN RANGACHARI *is a career coach for software engineers. Be sure to visit RadicalShifts.com for his career secrets.*

DevExpress®

DevExpress Spreadsheet for WPF & WinForms

with Chart, Pivot Table and Cell Editor support

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

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

Free 30-day trial

devexpress.com/spreadsheet

#UseTheBest

DINO ESPOSITO

ASP.NET Core for ASP.NET Developers

Most of the buzz about ASP.NET Core is centered on the multiplatform experience it enables. While this is a huge achievement, it's not necessarily a plus if you're a regular ASP.NET user with a large code base of .NET 4.x code and no plans to leave the familiar IIS and Windows environment. In this case, what's the value proposition of ASP.NET Core for such a regular ASP.NET developer?

At first the new platform might look completely different, as if someone sneakily moved your cheese elsewhere overnight. ASP.NET Core is new, rebuilt from the ground up, according to more modern practices. This might or might not increase your programming power and your ability to address customers' concerns. Nobody can realistically answer that question on your behalf. This column attempts to clear the ground from any hype, benchmarks, and technology focus and go straight to the substance of things. If you're OK with the current platform, which aspects of ASP.NET Core can capture your attention?

Common Practices Engineered in the Framework

When the ASP.NET team members designed the original ASP.NET framework, they took most of the best practices of Active Server Pages and engineered them into a new framework. In doing so, they also introduced a lot of new stuff, such as compiled and managed code, automatic postbacks, and server controls. ASP.NET Core follows the same evolutionary pattern.

Common development practices, such as initial loading of configuration data, dependency injection, NuGet packages, claims-based authentication and Razor improvements, are native features of the new framework. The new framework also has a different startup procedure, a much more modular request-response middleware and even a slightly more flexible infrastructure for defining controllers and views. ASP.NET Core is also a cross-platform framework and lets you develop applications and host them on Windows, as well as macOS and Linux. In a way, ASP.NET Core forces you to write better code where a few additional levels of separation of concerns are forced by default. It's not anything that you can't already achieve with discipline, though.

For any form of greenfield development, ASP.NET Core is an excellent choice. Yet, being a brand-new framework, it has some unavoidable initial costs: Everyone on the team must become proficient with it. In addition, everyone must be, or become, proficient with the Model-View-Controller (MVC) application model. Not everything you can label as greenfield development is brand new. Reusing chunks of existing code, or at least existing skills (that is, data access or security skills), is desirable. How much of that is realistically possible? To address this point, ASP.NET Core comes in two flavors.

Flavors of ASP.NET Core

In **Figure 1**, you see the Visual Studio 2015 dialog box to create a new project. (It's essentially the same in Visual Studio 2017.)

The first template will create a classic, non-Core project. The other two templates can create an ASP.NET Core project for a different .NET Framework. It's the first crossroad you face in your journey through the ASP.NET Core unexplored territory.

Opting for the full .NET Framework gives you access to any existing .NET class libraries, but limits hosting to only Windows and IIS. **Figure 2** summarizes the differences.

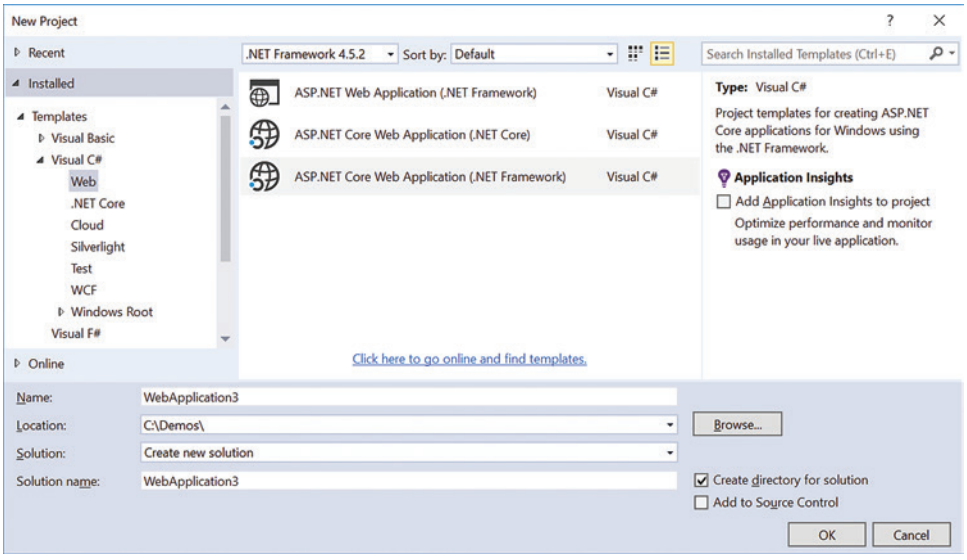


Figure 1 **Creating a New ASP.NET Core Project in Visual Studio**

We didn't invent the Internet...

...but our components help **you** power the apps that bring it to business.

TOOLS • COMPONENTS • ENTERPRISE ADAPTERS

- **E-Business**
  AS2, AS4, EDI/X12, OFTP …
- **Credit Card Processing**
  Authorize.Net, ACH, 3-D Secure …
- **Shipping & Tracking**
  FedEx, UPS, USPS …
- **Accounting & Banking**
  QuickBooks, OFX, SAP …
- **Internet Business**
  Amazon, PayPal, Google …
- **Internet Protocols**
  FTP, SMTP, IMAP, POP, WebDav …
- **Secure Connectivity**
  SSH, SFTP, SSL, Certificates …
- **Encryption & Certificates**
  X.509, OpenPGP, SHA, S/MIME …
- **Network Management**
  SNMP, MIB, LDAP, Monitoring …
- **Compression**
  Zip, Gzip, Jar, AES, 7Zip …

Our **Red Carpet Subscription** includes all product lines + updates for one year.

The Market Leader in Internet Communications, Security, & E-Business Components

Each day, as you click around the Web or use any connected application, chances are that directly or indirectly some bits are flowing through applications that use our components, on a server, on a device, or right on your desktop. It's your code and our code working together to move data, information, and business. We give you the most robust suite of components for adding Internet Communications, Security, and E-Business Connectivity to any application, on any platform, anywhere, and you do the rest. For more than 20 years, we have had one goal: to provide the very best connectivity solutions for our professional developer customers. With more than 100,000 developers worldwide using our software and millions of installations in almost every Fortune 500 and Global 2000 company, our business is to connect business, one application at a time.

connectivity powered by

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

Microsoft .NET | Java | C++ | Delphi | C++ Builder | ActiveX | Linux/Unix | macOS | Android | iOS | PHP | Xamarin | Node.js

Figure 2 **Fundamental Differences Between Flavors of ASP.NET Core**

Framework	Facts
.NET Framework	ASP.NET MVC only; no WebForms New runtime environment and programming API Any libraries targeting the selected version of the .NET Framework Only IIS hosting
.NET Core	ASP.NET MVC only; no WebForms New runtime environment and programming API Only .NET Core libraries Cross-platform hosting

Regardless of the choice of the .NET Framework, going with ASP.NET Core exposes your code to a new runtime environment that unifies the MVC runtime based on system.web with the Web API runtime inspired by the OWIN principles.

Breaking Up with IIS

In recent years, the Web API framework attempted to address the high demand of thin servers capable of exposing a RESTful interface to any HTTP-enabled clients. Web API decoupled the application model from the Web server and led to the OWIN specification, namely a set of rules for Web server and applications to interoperate. Web API, however, needs a host and when hosted in the context of an ASP.NET application, it just adds another runtime environment to the memory footprint. This happens at the same time in which the industry is moving toward super-simple Web. A super-simple, minimal Web server is an HTTP endpoint to get content as quickly as possible, just a thin HTTP layer around some business logic. All that a super-simple server needs to do is process the request as appropriate and return a response with no overhead except for the business logic.

Although customizable to some extent, the current ASP.NET runtime environment wasn't designed to address similar scenarios. Separating the ASP.NET environment from the hosting environment is the key change you find in ASP.NET Core and the reason for many of the subsequent application-level changes.

Startup of an Application

After creating a new project, the first thing you notice is the lack of a global.asax file and the presence of a program.cs file. As shocking as it might be, an ASP.NET Core application is a plain console application launched by the dotnet driver tool (bit.ly/2mLyHxe).

The dotnet tool is the key to multi-platform support. Once the command-line tool (and the .NET Core framework) is available for a new platform, hosting reduces to connecting the Web server to the tool. Under IIS, publishing is accomplished through an ad hoc module, the .NET Core Windows Server Hosting package (bit.ly/2i9cF4d). Under Apache, on an Ubuntu server, it's achieved through a configuration file. An example can be found at bit.ly/2lSd0aF.

The actual Web server works as a reverse proxy and communicates with the console application over a configured port. The console application is built around another, simpler, Web server that receives requests and triggers the internal application pipeline to have them processed. The following code does the job:

```
var host = new WebHostBuilder()
  .UseKestrel()
  .UseContentRoot(Directory.GetCurrentDirectory())
  .UseIISIntegration()
  .UseStartup<Startup>()
  .Build();
Host.Run();
```

Kestrel is the name of the ASP.NET Web server that receives inbound requests and processes them through the pipeline. The IIS Integration module call in the code snippet is only required if you host under IIS.

Having a reverse proxy around the ASP.NET Core application is recommended primarily for security reasons as the Kestrel internal Web server doesn't include (at the moment) filters to prevent things such as distributed denial of service (DDoS) attacks. From a purely functional point of view, you don't strictly need a reverse proxy enabled.

Kestrel is the name of the ASP.NET Web server that receives inbound requests and processes them through the pipeline.

As mentioned, there's no longer a global.asax file in an ASP.NET Core application and the role of the web.config file is wildly diminished. Actually, it only serves the purpose of enabling IIS to do some work on behalf of the application, such as serving some static error pages. Key things such as configuring error handling, logging, authentication and storing global configuration data are done through a new API orchestrated from the startup class.

The Startup Class

The startup class contains at least a couple of methods that the host will call during the initialization phase:

```
public class Startup
{
  public void ConfigureServices(IServiceCollection services)
  public void Configure(IApplicationBuilder app)
  {
    app.Run(async (context) =>
    {
      await context.Response.WriteAsync(DateTime.Now)
    });
  }
}
```

Through the ConfigureServices method you declare the system services the application will use. Technically, the method is optional, but I'd say that one is necessary in any realistic scenarios. To a

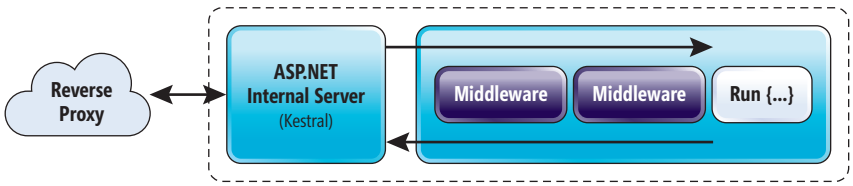


Figure 3 **The ASP.NET Core Middleware**

traditional ASP.NET developer, it might be shocking to find that even the use of the MVC application model must be explicitly declared and enabled. However, this fact gives the measure of how seriously modularization is taken in ASP.NET Core:

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

In the method Configure, you configure any previously requested services. For example, if you requested the ASP.NET MVC service, then in Configure you can specify the list of supported routes. Note that you also need an explicit call to enable the internal Web server to serve static files, including common files such as jQuery and Bootstrap:

```
public void Configure(IServiceCollection services)
{
  app.UseStaticFiles();
  app.UseMvcWithDefaultRoute();
  ...
}
```

The startup class is also the place where you configure the middleware of the application. Middleware is a new term that has a significant conceptual overlap with the HTTP modules of current ASP.NET. In ASP.NET Core, the middleware works as shown in **Figure 3**.

> In ASP.NET Core, creating an effective file server is easier than ever and more effective than ever.

You can register chunks of code that have a chance to pre- and post-process any incoming request, meaning that each middleware can register code that runs before or after the terminating middleware—the method Run in the Configure method of the startup class. The overall model is similar to the old ISAPI model of IIS. Here's some sample middleware:

```
app.Use(async (httpContext, next) =>
{
  // Pre-process the request

  // Yield to the next middleware
  await next();

  // Post-process the request
});
```

A middleware is a function that takes an HttpContext object and returns a Task. The list of middleware components ends with the Run method. Compared to the current ASP.NET pipeline, the ASP.NET Core pipeline is bidirectional and fully customizable. Moreover, it's empty by default.

Super-Simple Web Services

It's understood that without a Run method in the pipeline, no requests will ever produce a response. At the same time, a Run method is all you need to produce a response. This shows how short the pipeline can be in an ASP.NET Core application. In ASP.NET WebForms and MVC, many things happen before your own code runs for each request. Resolving a controller method, for example, is quite a long procedure that involves an entire subsystem centered on the action invoker. The Run method, instead, is a direct call that immediately follows the receipt of the request.

Suppose you want to create a file server that owns a list of image files (say, flags) and returns a properly sized image based on some input parameters or device properties. In today's ASP.NET, the fastest option is probably writing an ad hoc HTTP handler—a class created by implementing the IHttpHandler interface mapped to a fixed URL route. An HTTP handler is faster than an ASPX endpoint and MVC controller action because of the slimmer pipeline. It also has a smaller footprint than a Web API endpoint because it doesn't require a second OWIN pipeline on top of the basic ASP.NET pipeline. (This isn't the case when a Web API solution is hosted outside IIS.)

In ASP.NET Core, creating an effective file server is easier than ever and more effective than ever. All you do is craft the additional logic (that is, resize/retrieval) and bind it to the Run method of the ConfigureServices method:

```
public void Configure(IApplicationBuilder app)
{
  app.Run(async (context) =>
  {
    var code = context.Request.Query["c"];
    var size = context.Request.Query["s"];
    var file = FindAndResizeFlag(code, file);
    await context.Response.SendFileAsync(file);
  });
}
```

In the example, I assume to have some custom logic to find a server file name that matches provided parameters, and then I serve it back to the caller via the Response object. No other code, either visible or invisible, is required. Frankly, it couldn't really be easier than this.

Whatever your gut feeling is about ASP.NET Core, whether you're skeptical or enthusiastic about it, ASP.NET Core provides a unique capability you won't find on any ASP.NET platform: creating super-simple minimal Web services. At the same time, the same infrastructure that lets you build super-simple Web services is the best guarantee that any requests can be served with the minimum overhead that's legitimately possible.

Wrapping Up

To be a productive ASP.NET Core developer, you only need to be familiar with an application model more modern than WebForms: this means ASP.NET MVC or the single-page application model. In spite of the appearances, most of the breaking changes of ASP.NET Core are in the runtime environment. Understanding the hosting model and the middleware is enough to make sense of the new platform. In the end, it's still about creating controllers and rendering Razor views. A few common things like authentication, logging and configuration will need a different API, but learning that only takes a while. As I see it, the challenge is finding what's in it for you. ■

Dino Esposito *is the author of "Microsoft .NET: Architecting Applications for the Enterprise" (Microsoft Press, 2014) and "Modern Web Applications with ASP.NET" (Microsoft Press, 2016). A technical evangelist for the .NET and Android platforms at JetBrains, and frequent speaker at industry events worldwide, Esposito shares his vision of software at software2cents@wordpress.com and on Twitter: @despos.*

Thanks to the following Microsoft technical expert for reviewing this article: *James McCaffrey*

Cross-Platform Code Generation with Roslyn and .NET Core

Alessandro Del Sole

.NET Core is the modular, open source and cross-platform set of tools that allows you to build next-generation .NET applications, which run on Windows, Linux and macOS (microsoft.com/net/core/platform). It can also be installed on the Windows 10 for IoT distribution, and it runs on devices such as the Raspberry PI. .NET Core is a powerful platform that includes the runtime, libraries, and compilers, with full support for languages such as C#, F#, and Visual Basic. This means you can code in C# not only on Windows, but also on different OSes because the .NET Compiler Platform (github.com/dotnet/roslyn), also referred to as "Project Roslyn," provides open source, cross-platform compilers with rich code analysis APIs. As an important implication, you can leverage the Roslyn APIs to perform many code-related operations on different OSes, such as code analysis, code generation and compilation. This article walks through the necessary steps to set up a C# project on .NET Core to use the Roslyn APIs and explains some interesting code-generation and compilation scenarios. It also discusses some basic Reflection techniques to invoke and run code compiled with Roslyn on .NET Core. If you're unfamiliar with Roslyn, you'll want to read the following articles first:

- "Use Roslyn to Write a Live Code Analyzer for Your API" (msdn.com/magazine/dn879356)
- "Adding a Code Fix to Your Roslyn Analyzer" (msdn.com/magazine/dn904670)
- "Maximize Your Model-View-ViewModel Experience with Roslyn" (msdn.com/magazine/mt703435).

This article discusses:

- Building a .NET Core application in C#
- Adding the Roslyn NuGet packages
- Code analysis and code generation
- Getting diagnostic information
- Compiling and executing code on-the-fly with the Emit APIs

Technologies discussed:

C#, .NET Core, Roslyn, Visual Studio 2017, Visual Studio Code

Installing the .NET Core SDK

The first step is installing .NET Core and the SDK. If you work on Windows and you've installed Visual Studio 2017, .NET Core is already included if the .NET Core cross-platform development workload was selected at installation time in the Visual Studio Installer. If not, simply open the Visual Studio Installer, select the workload and click Modify. If you're working on Windows but not relying on Visual Studio 2017, or you're working on Linux or macOS, you can install .NET Core manually and use Visual Studio Code as the development environment (code.visualstudio.com). The latter is the scenario I'll discuss in this article, as Visual Studio Code is cross-platform itself; thus, it's a great companion for .NET Core. Also, remember to install the C# extension for Visual Studio Code (bit.ly/29b1Ppl). The steps to install .NET Core are different depending on the OS, so follow the instructions at bit.ly/2mJArWx. Make sure you install the latest release. It's worth mentioning that the latest releases of .NET Core no longer support the project.json file format, but instead support the more common .csproj file format with MSBuild.

Scaffolding a .NET Core Application in C#

With .NET Core, you can create Console applications and Web applications. For Web applications, Microsoft is making more templates available, besides the ASP.NET Core template, as .NET Core goes forward on its roadmap. Because Visual Studio Code is a lightweight editor, it doesn't provide project templates as Visual Studio does. This means you need to create an application from the command line inside a folder whose name will also be the application name. The following example is based on instructions for Windows, but

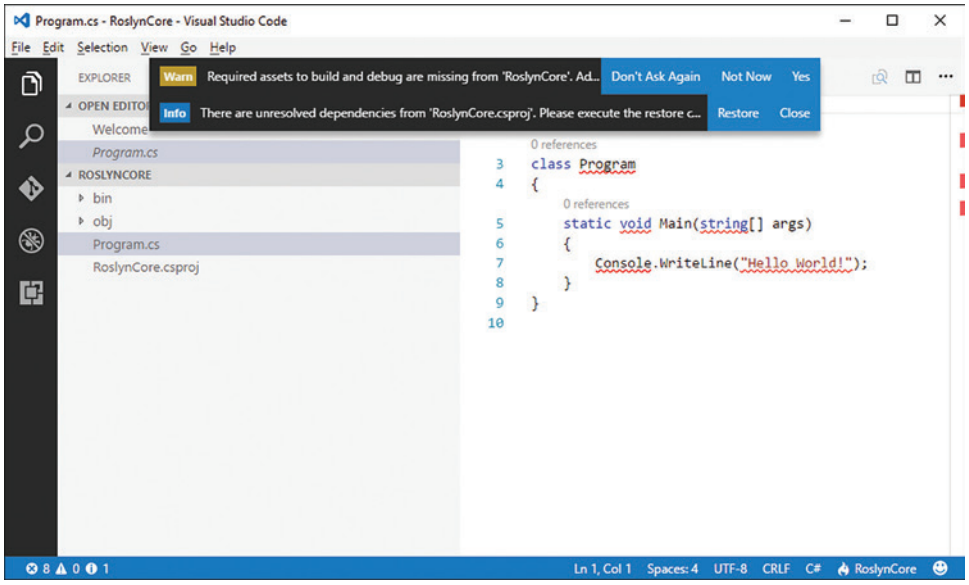


Figure 1 **Visual Studio Code Needs to Update the Project**

the same concepts apply to macOS and Linux. To get started, open a command prompt and move to a folder on your disk. For instance, say you have a folder called C:\Apps, go to this folder and create a new subfolder called RoslynCore, using the following commands:

```
> cd C:\Apps
> md RoslynCore
> cd RoslynCore
```

Therefore, RoslynCore will be the name of the sample application discussed in this article. It will be a Console application, which is perfect for instructional purposes and simplifies the approach to coding with Roslyn. You can also use the same techniques in ASP.NET Core Web applications. To create a new, empty project for a Console application, simply type the following command line:

```
> dotnet new console
```

In this way, .NET Core scaffolds a C# project for a Console application called RoslynCore. Now you can open the project's folder with Visual Studio Code. The easiest way is typing the following command line:

```
> code .
```

Of course, you could open Visual Studio Code from the Windows Start menu and then open a project folder manually. Once you enter any C# code file, it will ask your permission to generate some required assets and to restore some NuGet packages (see **Figure 1**).

The next step is adding the NuGet packages necessary for working with Roslyn.

Adding the Roslyn NuGet Packages

As you might know, the Roslyn APIs can be consumed by installing some NuGet packages from the Microsoft.CodeAnalysis hierarchy. Before installing these packages, it's important to clarify how the Roslyn APIs fit into the .NET Core system. If you've ever worked with Roslyn on the .NET Framework, you might be used to taking advantage of the full set of Roslyn APIs. .NET Core relies on .NET Standard libraries, which means that only the Roslyn libraries that support .NET Standard can be consumed in .NET Core. At the time of this writing, most of the Roslyn APIs are already available to .NET Core, including (but not limited to) the Compiler APIs (with the Emit and Diagnostic APIs) and the Workspaces APIs. Only a few APIs are not yet portable, but Microsoft is investing hugely in Roslyn and .NET Core, so it's reasonable to expect full .NET Standard compatibility in future releases. A real-world example of a cross-platform application that runs on .NET Core is OmniSharp (bit.ly/2mpcZeF), which leverages the Roslyn APIs to empower most of the code editor features, such as completion lists and syntax highlighting.

In this article, you'll see how to leverage the Compiler and the Diagnostic APIs. To accomplish this, you need to add the Microsoft.CodeAnalysis.CSharp NuGet package to your project. With the new .NET Core project system based on MSBuild, the list of NuGet packages is now included in the .csproj project file. In Visual Studio 2017, you can use the client UI for NuGet to download, install and manage packages, but in Visual Studio Code there's no equivalent option. Fortunately, you can simply open the .csproj file and locate the <ItemGroup> node that contains <PackageReference> elements, each representing a required NuGet package. Modify the node as follows:

```
<ItemGroup>
  ...
  <PackageReference Include="Microsoft.CodeAnalysis.CSharp"
    Version="2.0.0 " />
  <PackageReference Include="System.Runtime.Loader" Version="4.3.0" />
</ItemGroup>
```

Figure 2 **Parsing Source Code and Retrieving a Syntax Node**

```
using System;
using RoslynCore;
using Microsoft.CodeAnalysis;
using Microsoft.CodeAnalysis.CSharp;

class Program
{
  static void Main(string[] args)
  {
    GenerateSampleViewModel();
  }

  static void GenerateSampleViewModel()
  {
    const string models = @"namespace Models
{
  public class Item
  {
    public string ItemName { get; set }
  }
}
";
    var node = CSharpSyntaxTree.ParseText(models).GetRoot();
    var viewModel = ViewModelGeneration.GenerateViewModel(node);

    if(viewModel!=null)
      Console.WriteLine(viewModel.ToFullString());

    Console.ReadLine();
  }
}
```

Figure 3 **Generating a New Syntax Node**

```
using Microsoft.CodeAnalysis.CSharp.Syntax;
using Microsoft.CodeAnalysis;
using System.Linq;
using Microsoft.CodeAnalysis.CSharp;

namespace RoslynCore
{
  public static class ViewModelGeneration
  {
    public static SyntaxNode GenerateViewModel(SyntaxNode node)
    {
      // Find the first class in the syntax node
      var classNode = node.DescendantNodes()
       .OfType<ClassDeclarationSyntax>().FirstOrDefault();

      if(classNode!=null)
      {
        // Get the name of the model class
        string modelClassName = classNode.Identifier.Text;
        // The name of the ViewModel class
        string viewModelClassName = $"{modelClassName}ViewModel";
        // Only for demo purposes, pluralizing an object is done by
        // simply adding the "s" letter. Consider proper algorithms
        string newImplementation =
          $@"public class {viewModelClassName} : INotifyPropertyChanged
{{
public event PropertyChangedEventHandler PropertyChanged;
// Raise a property change notification
protected virtual void OnPropertyChanged(string propname)
{{
  PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propname));
}}
private ObservableCollection<{modelClassName}> _{modelClassName}s;
public ObservableCollection<{modelClassName}> {modelClassName}s
{{
  get {{ return _{modelClassName}s; }}
  set
  {{
    _{modelClassName}s = value;
    OnPropertyChanged(nameof({modelClassName}s));
  }}
}}
public {viewModelClassName}() {{
// Implement your logic to load a collection of items
}}
}}
";
        var newClassNode =
          CSharpSyntaxTree.ParseText(newImplementation).GetRoot()
          .DescendantNodes().OfType<ClassDeclarationSyntax>()
          .FirstOrDefault();

        // Retrieve the parent namespace declaration
        if(!(classNode.Parent is NamespaceDeclarationSyntax)) return null;

        var parentNamespace = (NamespaceDeclarationSyntax)classNode.Parent;
        // Add the new class to the namespace and adjust the white spaces
        var newParentNamespace =
          parentNamespace.AddMembers(newClassNode).NormalizeWhitespace();
        return newParentNamespace;
      }
    }
    else
    {
      return null;
    }
  }
}
```

Note that adding a reference to the Microsoft.CodeAnalysis.CSharp package allows you to access the C# compiler's APIs, and that the System.Runtime.Loader package is required for Reflection and will be used later in the article.

When you save your changes, Visual Studio Code will detect missing NuGet packages and will offer to restore them.

Code Analysis: Parsing Source Text and Generating Syntax Nodes

The first example is about code analysis, and demonstrates how to parse source code text and generate new syntax nodes. For instance, imagine you have the following simple business object and you want to generate a View Model class based on it:

```
namespace Models
{
  public class Item
  {
    public string ItemName { get; set }
  }
}
```

The text for this business object might come from different sources, such as a C# code file or a string in your code or even from user input. With the code analysis APIs, you can parse the source text and generate a new syntax node that the compiler can understand and manipulate. For example, consider the code shown in

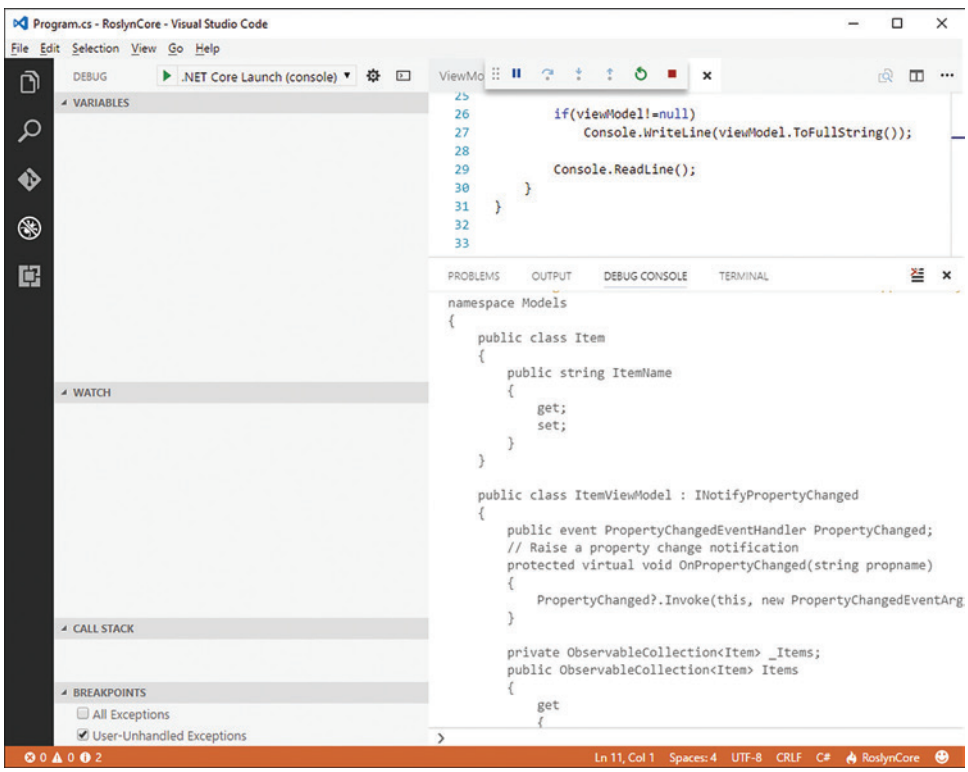


Figure 4 **The View Model Class Has Been Correctly Generated**

File Format APIs

Working with Files?

CREATE CONVERT PRINT

MODIFY COMBINE

FREE TRIAL

Aspose.Total

Manipulate Word, Excel, PDF, PowerPoint, Outlook and more than 100 other file formats in your applications without installing Microsoft Office.

DOC, XLS, PDF, PPT, MSG, BMP, PNG, XML and many more!

Platforms supported: .NET, Java, Cloud, Android, SharePoint, Reporting Services, and JasperReports

ASPOSE
File Format APIs

CONTACT US

US: +1 903 306 1676
EU: +44 141 628 8900
AU: +61 2 8006 6987

sales@asposeptyltd.com

Try for FREE at
www.aspose.com

Figure 2, which parses a string containing a class definition, gets its corresponding syntax node and calls a new static method that generates a View Model from the syntax node.

The GenerateViewModel method is going to be defined in a static class called ViewModelGeneration, so add a new file called ViewModelGeneration.cs to the project. The method looks for a class definition in the input syntax node (for demonstration purposes, the first instance of a ClassDeclarationSyntax object), then constructs a new View Model based on the class's name and members. **Figure 3** demonstrates this.

In the first part of the code in **Figure 3**, you can see how the View Model is first represented as a string, with string interpolation that makes it easy to specify object and member names based on the original class name. In this sample scenario, plurals are generated just by adding an "s" to the object/member name; in real-world code you should use more specific pluralization algorithms.

In the second part of **Figure 3**, the code invokes CSharpSyntaxTree.ParseText to parse the source text into a SyntaxTree. GetRoot is invoked to retrieve the SyntaxNode for the new tree; with DescendantNodes().OfType<ClassDeclarationSyntax>(), the code retrieves only the syntax nodes that represent a class, selecting only the first one with FirstOrDefault. Retrieving the first class in the syntax node is enough to get the parent namespace where the new View Model class will be inserted. Obtaining a namespace is possible by casting the Parent property of a ClassDeclarationSyntax into a NamespaceDeclarationSyntax object. Because a class could be nested into another class, the code first checks for this possibility by verifying that Parent is of type NamespaceDeclarationSyntax. The final piece of the code adds the new syntax node for the View Model class to the parent namespace, returning this as a syntax node. If you now press F5, you'll see the result of the code generation in the Debug Console, as shown in **Figure 4**.

The generated View Model class is a SyntaxNode that the C# compiler can work with, so it can be further manipulated, analyzed for diagnostic information, compiled into an assembly with the Emit APIs and utilized via Reflection.

Getting Diagnostic Information

Whether source text comes from a string, a file or user input, you can take advantage of the Diagnostic APIs to retrieve diagnostic information about code issues such as errors and warnings. Remember that the Diagnostic APIs not only allow for retrieving errors and warnings, they also allow writing analyzers and code refactorings. Continuing the previous example, it's a good idea to check for syntactic errors in the original source text before attempting to generate a View Model class. To accomplish this, you can invoke the SyntaxNode.GetDiagnostics method, which returns an IEnumerable<Microsoft.CodeAnalysis.Diagnostic>

Figure 5 **Checking for Code Issues with the Diagnostic APIs**

```
using Microsoft.CodeAnalysis.CSharp.Syntax;
using Microsoft.CodeAnalysis;
using System.Linq;
using Microsoft.CodeAnalysis.CSharp;
using System;

namespace RoslynCore
{
  public static class ViewModelGeneration
  {
    public static SyntaxNode GenerateViewModel(SyntaxNode node)
    {
      // Find the first class in the syntax node
      var classNode =
        node.DescendantNodes().OfType<ClassDeclarationSyntax>().FirstOrDefault();

      if(classNode!=null)
      {
        var codeIssues = node.GetDiagnostics();
        if(!codeIssues.Any())
        {
          // Get the name of the model class
          var modelClassName = classNode.Identifier.Text;
          // The name of the ViewModel class
          var viewModelClassName = $"{modelClassName}ViewModel";
          // Only for demo purposes, pluralizing an object is done by
          // simply adding the "s" letter. Consider proper algorithms
          string newImplementation =
            $@"public class {viewModelClassName} : INotifyPropertyChanged
{{
public event PropertyChangedEventHandler PropertyChanged;
// Raise a property change notification
protected virtual void OnPropertyChanged(string propname)
{{
  PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propname));
}}
private ObservableCollection<{modelClassName}> _{modelClassName}s;
public ObservableCollection<{modelClassName}> {modelClassName}s
{{
  get {{ return _{modelClassName}s; }}
  set
  {{
    _{modelClassName}s = value;
    OnPropertyChanged(nameof({modelClassName}s));
  }}
}}
public {viewModelClassName}() {{
// Implement your logic to load a collection of items
}}
}}
";
            var newClassNode =
              SyntaxFactory.ParseSyntaxTree(newImplementation).GetRoot()
              .DescendantNodes().OfType<ClassDeclarationSyntax>()
              .FirstOrDefault();

            // Retrieve the parent namespace declaration
            if(!(classNode.Parent is NamespaceDeclarationSyntax)) return null;

            var parentNamespace = (NamespaceDeclarationSyntax)classNode.Parent;
            // Add the new class to the namespace
            var newParentNamespace =
              parentNamespace.AddMembers(newClassNode).NormalizeWhitespace();
            return newParentNamespace;
          }
          else
          {
            foreach(Diagnostic codeIssue in codeIssues)
          {
            string issue = $"ID: {codeIssue.Id}, Message: {codeIssue.GetMessage()},
              Location: {codeIssue.Location.GetLineSpan()},
              Severity: {codeIssue.Severity}";
            Console.WriteLine(issue);
          }
          return null;
        }
      }
      else
      {
        return null;
      }
    }
  }
}
```

Data Quality Made Easy.
Your Data, Your Way.

Melissa provides the full spectrum of data quality to ensure you have data you can trust.

We profile, standardize, verify, match and enrich global **People Data** – name, address, email, phone, and more.

Our data quality solutions are available on-premises and in the Cloud – fast, easy to use, and powerful developer tools and plugins for the **Microsoft® Product Ecosystem**.

Microsoft® SQL Server® Integration Services

Microsoft Dynamics® CRM

Microsoft® .NET

Microsoft Azure

Microsoft® Excel

Start Your Free Trial

www.Melissa.com/msft-pd

Melissa Data is Now Melissa.
Why the change?
See for Yourself at the New www.Melissa.com

melissa™

1-800-MELISSA

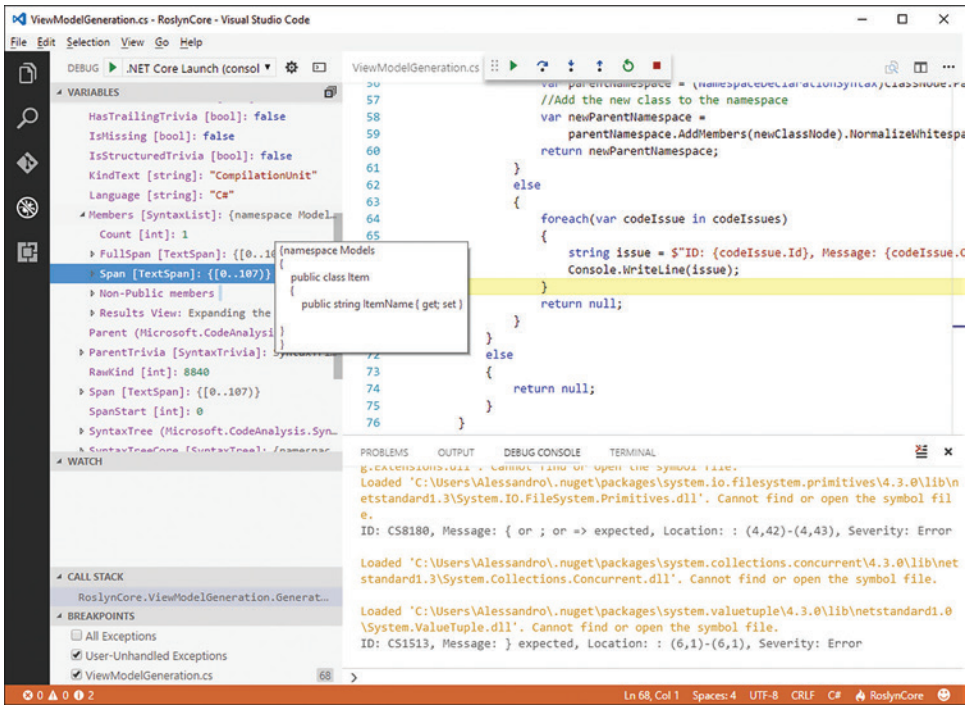


Figure 6 **Detecting Code Issues with the Diagnostic APIs**

object, if any. Take a look at **Figure 5**, which offers an extended version of the ViewModelGeneration class. The code checks if the result of invoking GetDiagnostics contains any diagnostics. If not, the code generates the View Model class. If instead the result contains a collection of diagnostics, the code shows information for each diagnostic and returns null. The Diagnostic class provides very granular information about each code issue; for example, the Id property returns a diagnostic id; the GetMessage method returns the full diagnostic message; GetLineSpan returns the diagnostic position in the source code; and the Severity property returns the diagnostic severity, such as Error, Warning or Information.

Now, if you introduce some intentional errors into the source text contained in the models variable, inside the GenerateSampleViewModel method in Program.cs, and then run the application, you'll be able to see how the C# compiler returns full details about every code issue. **Figure 6** shows an example.

It's worth noting that the C# compiler produces a syntax tree even if it contains diagnostics. Not only does this result in full fidelity with the source text, it also gives developers an option to fix those issues with new syntax nodes.

Figure 7 **Compiling and Executing Code with Emit APIs and Reflection**

```
using System;
using System.IO;
using System.Reflection;
using System.Runtime.Loader;
using Microsoft.CodeAnalysis;
using Microsoft.CodeAnalysis.CSharp;
using Microsoft.CodeAnalysis.Emit;

namespace RoslynCore
{
  public static class EmitDemo
  {
    public static void GenerateAssembly()
    {
      const string code = @"using System;
using System.IO;

namespace RoslynCore
{
 public static class Helper
 {
  public static double CalculateCircleArea(double radius)
  {
    return radius * radius * Math.PI;
  }
  }
}";
      var tree = SyntaxFactory.ParseSyntaxTree(code);
      string fileName="mylib.dll";

      // Detect the file location for the library that defines the object type
      var systemRefLocation=typeof(object).GetTypeInfo().Assembly.Location;

      // Create a reference to the library
      var systemReference = MetadataReference.CreateFromFile(systemRefLocation);

      // A single, immutable invocation to the compiler
      // to produce a library
      var compilation = CSharpCompilation.Create(fileName)
        .WithOptions(
          new CSharpCompilationOptions(OutputKind.DynamicallyLinkedLibrary))
        .AddReferences(systemReference)
        .AddSyntaxTrees(tree);


      string path = Path.Combine(Directory.GetCurrentDirectory(), fileName);
      EmitResult compilationResult = compilation.Emit(path);

      if(compilationResult.Success)
      {
        // Load the assembly
        Assembly asm =
          AssemblyLoadContext.Default.LoadFromAssemblyPath(path);

        // Invoke the RoslynCore.Helper.CalculateCircleArea method passing an argument
        double radius = 10;
        object result =
          asm.GetType("RoslynCore.Helper").GetMethod("CalculateCircleArea").
          Invoke(null, new object[] { radius });
        Console.WriteLine($"Circle area with radius = {radius} is {result}");
      }
      else
      {
        foreach (Diagnostic codeIssue in compilationResult.Diagnostics)
        {
          string issue = $"ID: {codeIssue.Id}, Message: {codeIssue.GetMessage()},
            Location: {codeIssue.Location.GetLineSpan()},
            Severity: {codeIssue.Severity}";
          Console.WriteLine(issue);
        }
      }
    }
  }
}
```

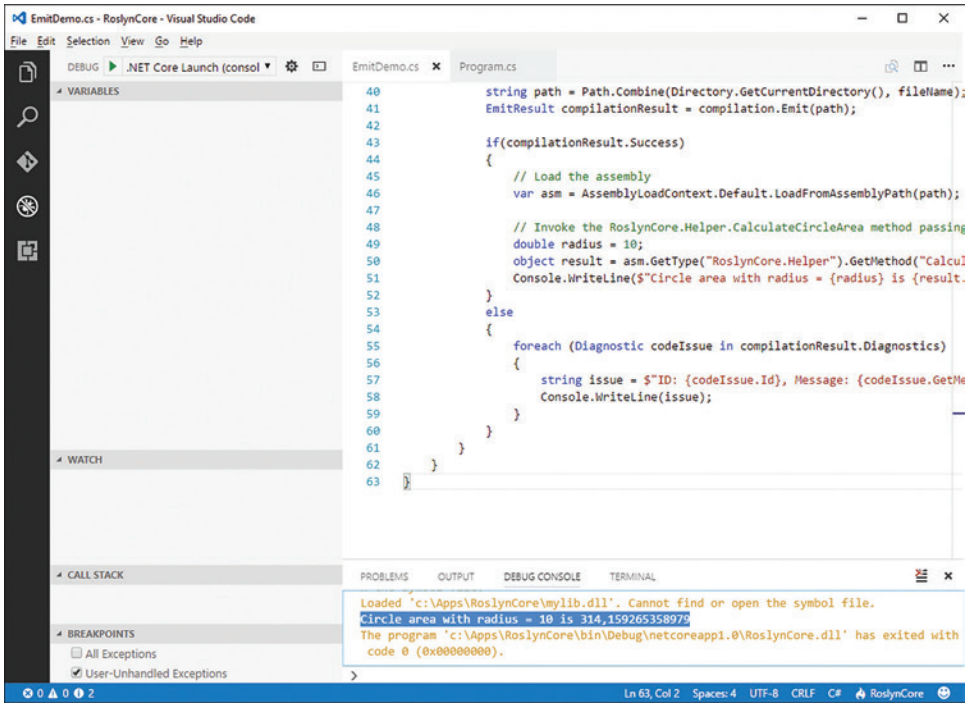


Figure 8 **The Result of the Invocation via Reflection of Roslyn-Generated Code**

Executing Code: the Emit APIs

The Emit APIs allow for compiling source code into assemblies. Then, with Reflection you can invoke and execute code. The next example is a combination of code generation, emit and diagnostic detection. Add a new file called EmitDemo.cs to the project, then consider the code listing shown in **Figure 7**. As you can see, a SyntaxTree is generated from source text that defines a Helper class containing a static method that calculates the area of a circle. The goal is to produce a .dll from this class and execute the CalculateCircleArea method, passing the radius as an argument.

In the first part, the code creates a new compilation that represents a single, immutable invocation to the C# compiler. The CSharpCompilation object allows the creation of an assembly through its Create method, and WithOptions lets you specify what kind of output to produce, in this case DynamicallyLinkedLibrary. AddReferences is used to add any references your code might need; to accomplish this, you must supply a type that has the same references needed by your code. In this particular case, you simply need the same references on which the object type relies. With GetTypeInfo().Assembly.Location you retrieve the assembly name for the reference, then MetadataReference.CreateFromFile creates a reference to the assembly inside the compilation. In the end, the syntax tree is added to the compilation with AddSyntaxTrees.

In the second part of the code, an invocation to CSharpCompilation.Emit attempts to produce the binary and returns an object of type EmitResult. The latter is very interesting: It exposes a Success property of type bool that indicates whether or not the compilation succeeded, and it also exposes a property called Diagnostics, which returns an immutable array of Diagnostic objects that can be very useful to understand why the compilation failed. In **Figure 7**, you can easily see how the Diagnostics property is iterated if the compilation fails. It's important to mention that the output assembly is a .NET Standard Library, so compiling source text will succeed only if the code parsed with Roslyn relies on APIs that are included in .NET Standard.

Now let's look at what happens if the compilation succeeds. The System.Runtime.Loader namespace, included in the same-named NuGet package you imported at the beginning of the article, exposes a singleton class called AssemblyLoadContext, which exposes a method called LoadFromAssemblyPath. This method returns an instance of the Assembly class, which allows you to use Reflection to first get a reference to the Helper class, then to get a reference to the CalculateCircleArea method, which you can invoke by passing a value for the radius parameter. The MethodInfo.Invoke method receives null as the first argument because CalculateCircleArea is a static method; therefore, you don't need to pass any type instance. If you now call the GenerateAssembly method from Main in Program.cs, you'll see the result of this work, as demonstrated in **Figure 8**, where the result of the calculation is visible in the Debug Console.

As you can imagine, Emit APIs plus Reflection in .NET Core give you great power and flexibility, because you can generate, analyze and execute C# code regardless of the OS. In fact, all of the examples discussed in this article will certainly run not only on Windows, but also on macOS and most of the Linux distributions. Additionally, invoking code from a library can be accomplished using the Roslyn Scripting APIs, so you're not limited to Reflection.

Wrapping Up

.NET Core allows you to write C# code to create cross-platform applications that run on multiple OSes and devices, and this is because compilers are cross-platform themselves. Roslyn, the .NET Compiler Platform, empowers the C# compiler on .NET Core and allows developers to leverage the rich code analysis APIs to perform code generation, analysis and compilation. This implies that you can automate tasks by generating and executing code on the fly, analyze source text for code issues, and perform a large number of activities on source code—on Windows, macOS, and Linux. ■

Alessandro Del Sole *has been a Microsoft MVP since 2008. Awarded MVP of the Year five times, he has authored many books, eBooks, instructional videos and articles about .NET development with Visual Studio. Del Sole works as a senior .NET developer, focusing on .NET and mobile app development, training and consulting. You can follow him on Twitter: @progalex.*

Thanks to the following Microsoft technical expert for reviewing this article: *Dustin Campbell*

Protect Web Apps Using Microsoft Content Moderator

Maarten van de Bospoort and Sanjeev Jagtap

Every day billions of users take pictures and videos to share on social media. As anyone who's dealt with user-generated content on the Internet knows, the net's anonymity doesn't necessarily surface the prettiest human behavior.

Another important recent trend is the proliferation of chat bots. Not a day goes by without the release of a new set of bots that can do everything from booking travel to coordinating your meetings to online banking. While those bots are useful without a doubt, the killer chatbot is still elusive: the one bot that all the messaging platforms want in order to crack the goal of 1 billion daily active users.

Now let's imagine you created just that: Butterfly, the bot that everyone feels compelled to engage with. Users can share media with your bot and through your secret machine learning algorithm, the bot will predict their future for the next 24 hours. After a year of hard work, you released your bot. Overnight, Butterfly went viral. Unfortunately, your startup dream quickly turns into a public relations nightmare. Users are submitting adult and racy content, which is then shared and publicly available to other bot users. And some of the content is bad. Really bad. Users are suing you; the phone is ringing off the hook; and you receive threats that your Web service will be shut down. You need a powerful solution to help detect and prevent the bad content from being visible to other users. And you need it quick.

That's where Microsoft Content Moderator comes to the rescue!

In this article, we'll show you how Content Moderator can help. We'll start by creating a chatbot using the Microsoft Bot Framework, but keep in mind that the information applies equally to any Web or client application. Butterfly will enable end users to share text, images and videos, and will use Content Moderator to filter out the inappropriate material before it gets published. Along the way, you'll learn how to configure custom Content Moderator workflows and to adjust the thresholds for the content classifiers. We'll also discuss the different connectors that can be used in the workflow, such as Text and Child Exploitation. Let's start with an overview of content moderation.

This article discusses:

- How content moderation works
- Creating a bot to let users share images
- Using Content Moderator to evaluate image content
- Using the Moderator Review API to enable human review of content
- Creating custom workflows

Technologies discussed:

Microsoft Content Moderator, Microsoft Bot Framework, Moderator Review API

Code download available at:

msdn.com/magazine/0517magcode

Content Moderation

Microsoft has a long and proven track record combatting digital crime. The Microsoft Digital Crimes Unit works hard to take down botnets, limit tech support fraud, thwart phishing schemes and more. One less-visible active area is how the unit assists law

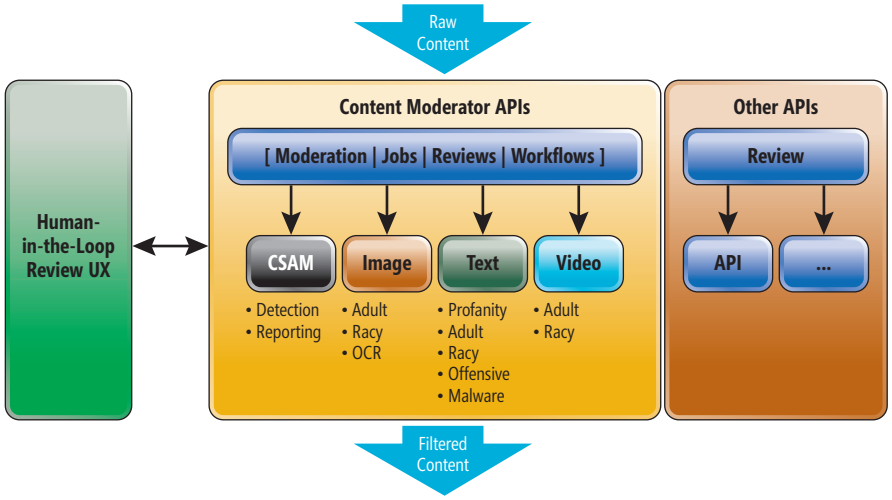


Figure 1 **User Content Flows Through Content Moderator, Where Machine Learning Models and Humans Work Together to Filter out Indecent Material**

enforcement worldwide with curbing child exploitation. Microsoft has offered PhotoDNA as a free service since 2009, and the same team now also presents content moderation.

Content moderation is an area of machine learning where computers can help humans tremendously. The amount of data generated by users is simply too much for humans to review quickly and in a cost-effective way. More important, content moderation isn't a pleasant activity for humans. For a little background, see tcrn.ch/2n1d9M0.

Content Moderation is a member of the growing set of the Microsoft Cognitive Services APIs that run in Azure. These APIs are all specific implementations of machine learning models, which means Microsoft has trained these models with lots of data. As a developer, you just call one of the APIs to get a result, whether for computer vision, speaker recognition or language understanding, to name but a few. The Content Moderator Image API uses image recognition—an area of machine learning where a lot of progress has been made in recent years.

Figure 1 shows how the Content Moderator pipeline is set up. Depending on your needs, content moderation offers different APIs to call, including moderation, review and jobs in increasing levels of customizability. For example, the workflow API allows you to programmatically modify workflows that are used for jobs. Next, you can see the different classifiers for image, text, and video, and CSAM, which stands for Child Sexual Abuse Material. PhotoDNA is the technology that helps organizations fight the spread of these images. The CSAM "classifier" works a little differently from the ones I mentioned previously: PhotoDNA uses a hash and match technology that compares against a database of known images. Within Content Moderator, you can set up workflows that connect several filters (for example, first check for CSAM, then for Adult/Racy in images). The workflow can call out to humans for review, as well. Finally, the Content Moderator pipeline is flexible and in the future other APIs can be hooked in, as well.

Figure 2 **Dialog Code**

```
bot.dialog('/moderateImage', [
  function (session, args) {
    builder.Prompts.text(session, 'Give me a picture URL to moderate, please.');
  },
  function (session, results){
    var input = session.message.text;
    var cm = require('./moderate.js');
    cm( 'ImageUrl', input, function(err, body) {
      if (err) {
        session.send('Oops. Something went wrong.');
        return;
      }
      var output = JSON.stringify(body);
      session.endDialog(output);
    });
  }
]);
```

Moderate Your Users' Content

Now that you have some understanding of the moderation technology, let's plug it into our Butterfly bot. We'll build the bot with the Microsoft Bot Framework, using the Node.js flavor of the Bot Framework for Butterfly. Because all of these APIs are mere REST API calls, you could moderate your content just as easily from C#; actually, this could arguably even be easier as there's a .NET SDK for Content Moderator (bit.ly/2mXQqQV).

There have been several articles in this magazine that give excellent overviews of bots. If you haven't built one yet, I highly recommend checking out the following:

- "Solving Business Problems with the Microsoft Bot Framework" (msdn.com/magazine/mt788623)
- "Use Bot Framework for Anytime, Anywhere Access to Application Data" (msdn.com/magazine/mt790202)
- "Making Bots More Intelligent" (msdn.com/magazine/mt795186)

Alternatively, the quick starts on dev.botframework.com will have you up and running in no time.

Here, we'll use the simple starter solution for Node.js. We use the dialog model that the Node.js framework provides to separate the conversation into separate dialogs. In the dialog code shown in **Figure 2**, the bot prompts the user for a URL to a picture in the first function. Control is then passed back to the user. When the user sends some text, the dialog flow passes the user input to the second function. The bot then forwards the input for evaluation in moderate.js. In our first attempt, we call the simple Moderator API (as opposed to the more sophisticated Review and Job APIs).

To call the Content Moderator API, you need credentials, which you can get from either the Content Moderator Web site or from Azure. Here, we'll take the latter approach. In the Azure Portal (portal.azure.com), create a new Cognitive Services account by clicking on the green plus sign and specifying Cognitive Services. After you click Create, specify Content Moderator as the API type (see **Figure 3**). Use the F0 tier because it's free and allows for one call per second, which should be enough to play around with for now. Once the account is created, you'll find the Account Name and Key under Keys in Resource Management. Note that you'll use one of the Keys and the Resource ID string (from Properties)

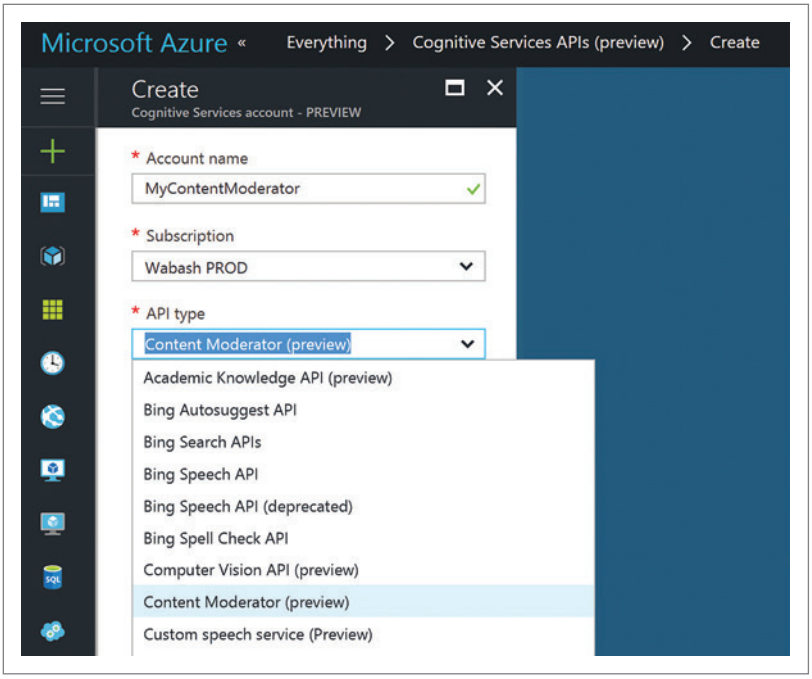


Figure 3 **Select Content Moderator from the Cognitive Services List**

to hook up the Content Moderator API to the Content Moderator review portal later.

The Azure Portal also shows you the endpoint as https://westus.api.cognitive.microsoft.com/contentmoderator. While that's the correct base address, it's a little too short. The full endpoint is in the Content Moderator documentation.

As shown in **Figure 4**, we specify "URL" as DataRepresentation to send the URL to the picture, but you can just as easily send the image in a blob. Once you've called the moderator API, the body of the returned result contains JSON with the scores for the image. The scores range from 0.0 (innocent) to 1.0 (very adult/racy).

You can see that the result, shown in **Figure 5**, contains the adult and racy prediction scores and the conclusions about whether it made the threshold in the classified flags. Happy as a clam, you throw these few lines of code in your bot and block all the content that's racy or adult. You deploy the new version of the bot to Azure and users stream in again to consult your oracle. Phew. Happy with your results, you and your team gather around your kegerator to celebrate with a fresh beverage.

Barely two sips into your micro-brewed IPA, tweets are keeping your phone on constant vibrate. There's a flood of angry customers: "Why are you blocking my innocent pictures?" "My brother can get pictures with substantially more skin through. You need to fix this!"

Image classification is good, but there's no one size fits all. The bare Content Moderator APIs we just used are clearly able to assist humans to make good decisions, but they're not perfect. One improvement we could've made is to fine-tune the moderation by using the raw scores instead of the true/false adult-and-racy classifications. Additionally, it appears that users tend to use the same images repeatedly. Fortunately, Content Moderator provides a List API to manage a custom set of images or text you've already screened. The moderator API does some fuzzy matching against the images, to prevent users from easily fooling it with slight modifications or resizing. This is a nice enhancement over the first approach, but it wouldn't rule out those false positives the help desk had to contend with. As always, the optimal solution is found when humans and machines work as a team on the more difficult cases. Computer Vision can help detect the extremes when images are clearly racy or adult, or clearly not. For the edge cases in between, we as humans can decide on which side of the fence the content falls for our particular scenario. This is where the Content Moderator review tool and API really shine. Let's look at how we can use it to improve our solution.

Figure 4 **Sending a Picture URL to Content Moderator**

```
var unirest = require("unirest");
var url = "https://westus.api.cognitive.microsoft.com/contentmoderator/
  moderate/v1.0/ProcessImage/Evaluate";

module.exports = function(input, cb) {
  unirest.post(url)
    .type("json")
    .headers({
      "content-type": "application/json",
      "Ocp-Apim-Subscription-Key":<Key from the Azure portal>,
    })
    .send({
      "DataRepresentation": "URL",
      "Value": input
    })
    .end(function (res) {
      return cb(res.error, res.body );
    });
};
```

Calling the Moderator Review API

The approach up to now has been straightforward: Send a picture and block or allow it based on the Content Moderator labels. Now we're going to expand the solution. The idea is to set up a flow as shown in **Figure 6**.

In this scenario, the user first sends an image to the Butterfly bot. In Step 2, the bot's Web service sends the picture to the Content Moderator using the Review API's Job operation, which takes a workflow Id as a parameter. We'll set up this workflow in the review tool. Our specific workflow (Step 3) will immediately allow all pictures that are below a certain adult/racy score (for example, 0.7) and flag others that exceed a certain limit (such as 0.9). Our bot will allow the

Figure 5 **Adult and Racy Classification Scores for an Image**

```
"AdultClassificationScore":0.0324602909386158,
"IsImageAdultClassified":false,
"RacyClassificationScore":0.06506475061178207,
"IsImageRacyClassified":false,
"AdvancedInfo":[],
"Result":false,
"Status":{
  "Code":3000,
  "Description":"OK",
  "Exception":null},
"TrackingId":"WU_ibiza_4470e022-4110-48bb-b4e8-7656b1f6703f_
  ContentModerator.F0_3e598d6c-5678-4e24-b7c6-62c40ef42028"
```

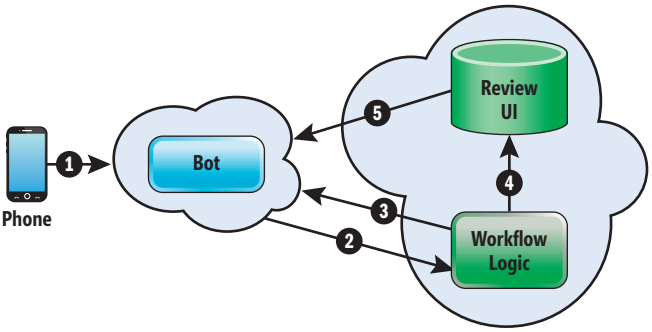


Figure 6 **The Butterfly Bot Working with a Content Moderator Workflow**

content with the low scores and block the content when it's clearly too racy or adult. In the gray area in between, we want the content to go to the review tool for human moderators to inspect (Step 4). Our team of reviewers can then decide how to deal with the content. When they're done reviewing, the Content Moderator will call our bot's app service back to share the result. At that point, the bot can take down the content if it has been flagged as offensive. Note the flexibility here. You can adjust the scores in your workflow and the reviewers can decide what's appropriate for your specific app.

To get started, you'll need to sign up for the Content Moderator review tool at bit.ly/2n8XUB6. You can sign up with your Microsoft account or create a local account. Next, the site asks you to create a review team, whose purpose is to review gray-area content. You can create multiple sub teams and create workflows that assign reviews to different sub teams. In the credentials tab of the portal's Settings page, you can link up your Content Moderator settings with the Azure Cognitive Services resource you created previously. Just copy the Key and Resource ID from the Azure portal to the Subscription Key and Resource ID settings in the Moderator UI. When you first create your account, you get an auto-configured "default" workflow. As you can see in the Review UI, this workflow will create a human review if an image is found to be adult. Let's start by using this workflow in the Review API's Job operation.

To call the Review Job API, you use the code shown in **Figure 7**.

Note that the URL contains the team name Butterfly and the postfix jobs. In CallBackEndpoint we specify the REST endpoint that Content Moderator will call to notify the review results. We also specify a unique ContentId so we can correlate the image when Content Moderator calls us back and we send the actual image URL in ContentValue. When the call succeeds, the body of the result doesn't contain any Content Moderator result. Instead, it returns the JobId:

```
{"JobId":"2017035c6c5f19bfa543f09ddfca927366dfb7"}
```

You'll get the result through the callback you specify in CallBackEndpoint. This result will again have the JobId, potentially a ReviewId, and a ContentId so you can cross reference it. For the default workflow, Content Moderator will call back immediately with the result in Metadata if the image isn't considered adult. The actual JSON will look similar to what's shown in **Figure 8**.

The status for this Job is set to Complete and the CallbackType is Job. If, however, the image is considered adult material, Content Moderator will create a review and populate the ReviewId field with an identifier. The image will then end up in the Review UI for your review team (see **Figure 9**).

The review tool and its use would benefit from a bit of explanation. The tool is designed for handling large volumes of images. A reviewer looks at all the pictures on a screen, tags the ones that don't pass muster and then moves to the next screen. The tool gives the reviewer a few seconds to go back in case he thinks he made a mistake. After those few seconds, Content Moderator saves the images with the final tags and calls the callback function we specified again, now with the final judgement. We can now take appropriate action—either taking down the content or publishing it based on our business requirements. The second call back will look like what's shown in **Figure 10**.

The CallBackType is now Review instead of Job and you can see the added ReviewerResultTags, while ContentId and ReviewId match the results from the first callback.

Custom Workflows

Now that we have a good understanding of the default workflow, we can start turning some knobs and dials. For Butterfly, we want to allow everything with a racy score less than 0.7, but block anything with a racy score higher than 0.9. For anything in between, we want the review team to take a second look. Therefore, in the workflow editor, we'll create a new workflow.

You'll see that there are lots of options in the dropdowns for Connect to. These options allow you to build advanced workflows

Figure 7 **Calling the Review Job API**

```
var url = 'https://westus.api.cognitive.microsoft.com/contentmoderator/
  review/v1.0/teams/butterfly/jobs';

var req = unirest.post(url)
  .type("application/json")
  .query({
    ContentType: 'Image',
    ContentId: '67c21785-fb0a-4676-acf6-ccba82776f9a',
    WorkflowName: 'default',
    CallBackEndpoint: 'http://butterfly.azure.com/review'
  })
  .headers({
    "Ocp-Apim-Subscription-Key": <ocp_key>
  })
  .send({
    "ContentValue": pictureUrl
  })
  .end(function (res) {
    return callback(res.error, res.body );
});
```

Figure 8 **Default Workflow Results**

```
{
  "JobId": "2017035c6c5f19bfa543f09ddfca927366dfb7",
  "ReviewId": "",
  "WorkFlowId": "default",
  "Status": "Complete",
  "ContentType": "Image",
  "CallBackType": "Job",
  "ContentId": "67c21785-fb0a-4676-acf6-ccba82776f9a",
  "Metadata": {
    "adultscore": "0.465",
    "isadult": "False",
    "racyscore": "0.854",
    "isracy": "True"
  }
}
```

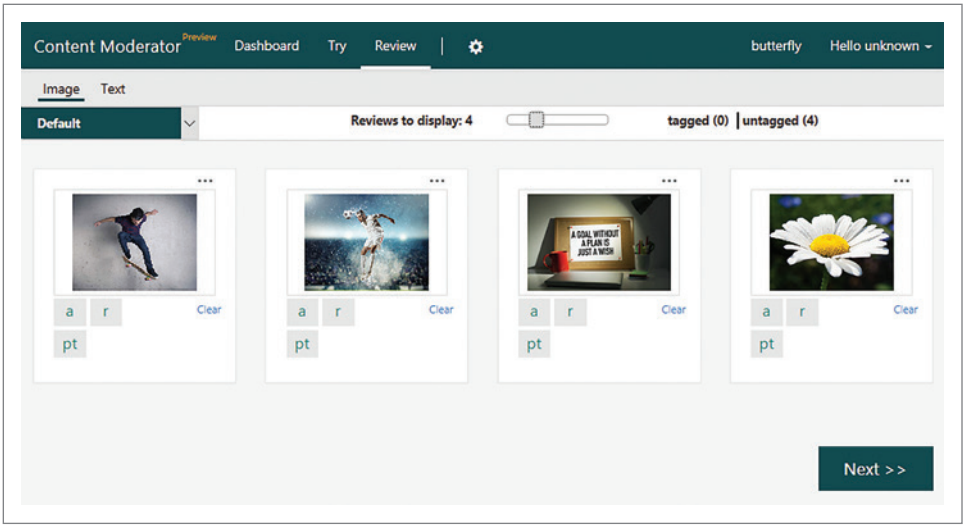


Figure 9 **The Content Moderator Review Tool with Pictures and Unselected Tags**

with, for example, the Optical Character Recognition (OCR) Text from Images and Face detection APIs. The tool also allows you to declare a callback endpoint for a review. If you specify a callback in the CallBackEndpoint to the Job API, as well as here, the one in the workflow overrides the CallBackEndpoint.

Now, when you call the Review Job API and specify this workflow, you'll get a JobId back, just like when you called the default workflow. Depending on the racy score of your picture (between 0.7 and 0.9 in our case), Content Moderator will again create a review and you'll see those images in the Content Moderator review UI.

There are two final notes about workflows. First, if the picture doesn't qualify for a review in the initial Job callback, we still must find out whether the picture was on the high end and needs to be blocked, or on the low end and is allowed. To do this, you must duplicate the logic a bit, and that means things can get out of sync. Fortunately, the review tool exposes the workflow as JSON. Even better, there's a Workflow REST API you can use to submit the workflows to the Content Moderator API. With a bit of plumbing you can use the same JSON to keep your bot's logic and the Review UI tool in sync.

A second note concerning the workflows relates to their extensibility. A focal point for the team is to make the review tool a common destination for various APIs. When you navigate to the Connectors tab in the review UI, you can see the currently available connectors. You can activate these connectors by entering the corresponding subscription keys. The case for PhotoDNA is easy to make. If your product gets any sort of user content, you want to make sure that no child exploitation images are being shared. Hooking this up to an existing workflow is easy once you've signed up for the service. It surely beats having to call the REST APIs separately. At the time of this writing, the Text Analytics API and the Face Detection API are available as connectors. For those you can go to the Azure Portal, create a Cognitive Service as we did earlier and enter the subscription key in the Content Moderator UI.

Figure 10 **Results of the Review Callback**

```
{
  "ReviewId": "201703367f430472074c1fb18651a04750448c",
  "ModifiedOn": "2017-03-07T18:34:17.9436621Z",
  "ModifiedBy": "Bob",
  "CallBackType": "Review",
  "ContentId": "67c21785-fb0a-4676-acf6-ccba82776f9a",
  "ContentType": "Image",
  "Metadata": {
    "adultscore": "0.465",
    "isadult": "False",
    "racyscore": "0.854",
    "isracy": "True"
  },
  "ReviewerResultTags": {
    "a": "True",
    "r": "True",
    "pt": "False"
  }
}
```

Wrapping Up

There are other advanced features we didn't have space to dig into. For example, you can create your own Tags under Settings to use in your workflows. We created a "pt" tag for tagging profanity in text. We'll use this in a workflow that's set up for text content moderation. Additionally, workflows have alternate inputs to handle situations where the input format doesn't match a qualifier. For example, when you need to detect text profanity in an image through OCR. You can sign up for Video Moderation, which is currently in private preview. Finally, you can expect more connectors to show up in the portal that you'll be able to use to build and extend your workflows.

Using Content Moderator allows you to scale out your content moderation capabilities to all media formats across large volumes. Content Moderator is a platform—APIs and solutions that we're building specifically for the content moderation vertical. Using it, you can scale and transition into other media formats and new content moderation capabilities as they become available. Content Moderator uses the best machine learning-based classifiers and other technologies that are getting better all the time. Improvements in the classifiers will automatically improve your results. ■

Maarten van de Bospoort *is a principal software development engineer in Developer Experience at Microsoft in Redmond. He works with developers and architects from large consumer ISVs to facilitate adoption of Microsoft technologies such as bots, cognitive services and occasionally a Universal Windows Platform app.*

Sanjeev Jagtap *is a senior product manager in the Content Moderator team at Microsoft in Redmond. He is passionate about customers, Microsoft technologies and Hackathons.*

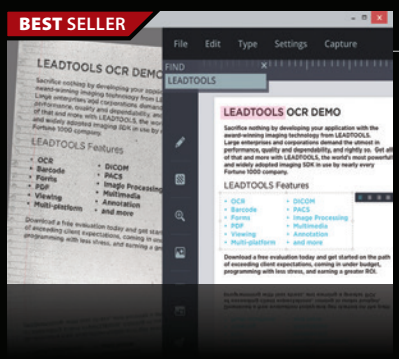
Thanks to the following Microsoft technical expert for reviewing this article: *Christopher Harrison and Sudipto Rakshit*

DevExpress DXperience 16.2 | from $1,439.99

DevExpress

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

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



LEADTOOLS Document Imaging SDKs V19 | from $2,995.00 SRP

LEAD TECHNOLOGIES

Add powerful document imaging functionality to desktop, tablet, mobile & web applications.

- Universal document viewer & conversion framework for PDF, Office, CAD, TIFF & more
- OCR, MICR, OMR, ICR and Forms Recognition supporting structured & unstructured forms
- PDF SDK with text edit, hyperlinks, bookmarks, digital signature, forms, metadata
- Barcode Detect, Read, Write for UPC, EAN, Code 128, Data Matrix, QR Code, PDF417
- Zero-footprint HTML5/JavaScript UI Controls & Web Services

Help & Manual Professional | from $586.04

ec software

Help and documentation for .NET and mobile applications.

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

Aspose.Total for .NET | from $2,939.02

ASPOSE File Format APIs

Every Aspose .NET API in one package.

- Programmatically manage popular file formats including Word, Excel, PowerPoint and PDF
- Work with charts, diagrams, images, project plans, emails, barcodes, OCR, Visio, OneNote, 3D and CAD files alongside many more document management features in your .NET applications
- Common uses also include mail merging, adding barcodes to documents, building dynamic reports on the fly and extracting text from most document types

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

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

European Headquarters
ComponentSource
2 New Century Place
East Street
Reading, Berkshire
RG1 4ET
United Kingdom

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

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

GSA Schedule Contract GS-35F-0188R

Compliance as Code with InSpec

Michael Ducy

Regulatory compliance is a fact of life for every enterprise. At the same time, competitive pressures are increasing with the advent of game-changing new technologies and customer expectations for digital services. Is it possible for industries to deliver new products and services at high velocity while still satisfying their obligations for regulatory compliance?

The answer is yes. The solution is to embed regulatory compliance into the software production line analogously to the way we embed other qualities, such as frame stiffness in automobiles or round-trip response time in banking applications.

Making compliance an integral part of the deployment process is possible when compliance is expressed as code. Just as the configuration of systems has shifted toward infrastructure as code (for example, PowerShell Desired State Configuration or Chef), you can manage compliance using a programmatic language.

This article discusses:

- Continuous delivery
- DevOps
- Infrastructure compliance
- Regulatory compliance
- Security compliance

Technologies discussed:

Windows Server, Linux, Unix, InSpec, Continuous Delivery/Release Pipelines

InSpec is an open source project that lets you define your compliance requirements in a human- and machine-readable language. Once you've codified your requirements, you can run them as automated tests that audit your systems. InSpec provides a local agent, as well as full remote testing support.

InSpec supports a variety of different platforms, from Windows to Linux. **Figure 1** lists some of the more popular ones. (A full list of supported platforms can be found on the InSpec Web site at inspec.io.)

The InSpec broad platform support makes it a complete solution for managing compliance across your entire infrastructure. Because InSpec is an open source project, some OS vendors have contributed support for their own platforms. For instance, IBM has contributed much of the support for its AIX OS.

Getting Started with InSpec

It's easy to get started with InSpec. InSpec is included in the Chef Development Kit (Chef DK) or you can download packages for a variety of platforms from the Chef download Web site at downloads.chef.io/inspec. Once you've downloaded the package and installed it, you can begin writing compliance rules. (Note that an alternative name for a compliance rule, often used by security teams, is auditing control.)

InSpec rules are simple to write once you understand the format. All rules begin with a resource. A resource is a configuration item you want to test. For instance, here's an InSpec rule that uses the windows_feature resource:

```
describe windows_feature('DHCP Server') do
  it { should_not be_installed }
end
```

Figure 1 **List of Popular Platforms Supported by InSpec**

Platform	Versions
AIX	6.1, 7.1, 7.2
Mac OS X	10.9, 10.10, 10.11
Oracle Enterprise Linux	5, 6, 7
Red Hat Enterprise Linux (and variants)	5, 6, 7
Solaris	10, 11
Windows	7, 8, 8.1, 10, 2008, 2008 R2, 2012, 2012 R2, 2016
Ubuntu Linux	
SUSE Linux Enterprise Server	11, 12
OpenSUSE	13.1, 13.2, 42.1
HP-UX	11.31

The windows_feature resource declares the name of a Windows feature and tests to see if it matches a particular configuration. In this example, the rule tests that the DHCP Server is not installed.

There are resources for many standard pieces of your network, such as files, directories, users, groups and registry keys. For a complete list, you can refer to the InSpec documentation at bit.ly/2n3ekZe. You can also easily extend InSpec with your own resources to check configuration items that aren't supported out of the box or that are specific to your particular application.

Including Metadata

InSpec lets you include metadata about your compliance rules. Metadata helps you tie tests to specific regulatory or security requirements. Traditionally, you'd find compliance requirements published in documents, spreadsheets or some other format that's not actionable. The information in these official compliance documents is important because it gives administrators context as to why the compliance policy matters, however it's often not conveniently available.

> InSpec uses a client-server model, which means that you can audit remote systems from a centralized workstation.

Figure 2 shows an example of an InSpec rule that includes this information as metadata.

This example is for a rule (or control) called ssh-8. The impact, title, and desc fields define metadata that describes the rule's importance, its purpose, and a description. The tag field includes optional information and the ref field references external documents.

The describe field signals the beginning of the block that contains the rule. The resource being tested is sshd_config, which is the OpenSSH daemon on Linux and Unix platforms. The rule tests to see if the SSH server listens to port 22.

There are three important points to notice. First, without the metadata, the rule would be isolated and lack context. Next, all the pertinent information is included with the rule. You don't have to check it against other documents. Finally, the InSpec language is extremely easy to read. Stakeholders such as compliance officers, who may not be programmers, can understand what the rule tests and the metadata tells them why the rule exists and what requirements it audits. They might even be inclined to contribute their own rules.

Using Open Source Profiles

To make life easier, InSpec has many open source profiles available that already include all the relevant rules and metadata. For example, there's a DevSec Linux Baseline profile and a DevSec Apache Baseline profile. You can download these profiles at bit.ly/2mBVXNr.

Many of the open source profiles InSpec provides are based on the industry standard Center for Internet Security (CIS) benchmarks for system security. While the CIS baselines provide a good starting point, you might need to modify them to meet your particular compliance needs. InSpec allows you to create your own profiles and to inherit rules from other profiles. InSpec also lets you ignore rules from profiles. This is useful because you don't need to directly modify the open source profiles InSpec provides. You can create your own profiles that inherit the open source profiles you need, and then ignore rules that aren't applicable. When new open source profiles are released, you can simply update your version of the open source rules without having to modify your custom rules.

Scanning a Host

InSpec uses a client-server model, which means that you can audit remote systems from a centralized workstation. There are also options that let InSpec scans to run as part of a continuous automation system, such as Chef Automate (chef.io/automate). There's a brief example of this option later in this article.

To run a compliance scan, you need a target system, which is the server you want to test and a compliance profile, which is the set of rules you use to test the target system. For this example, the target system is a Windows Server, and I'll use the Dev-Sec Windows baseline as defined by CIS, which is stored in a GitHub repo. **Figure 3** shows an example of an InSpec run.

If you examine the results, the run shows that there are a number of configuration settings that do not meet the CIS baselines for

Figure 2 **Example of InSpec with Metadata About Compliance Rules**

```
control 'sshd-8' do
  impact 0.6
  title 'Server: Configure the service port'
  desc '
    Always specify which port the SSH server should listen to.
    Prevent unexpected settings.
  '
  tag 'ssh','sshd','openssh-server'
  tag cce: 'CCE-27072-8'
  ref 'NSA-RH6-STIG - Section 3.5.2.1',
    url: 'https://www.nsa.gov/ia/_files/os/redhat/rhel5-guide-i731.pdf'
  describe sshd_config do
    its('Port') { should eq('22') }
  end
end
```

```
michael@ricardo:~$ inspec exec -t winrm://Administrator@54.174.127.151 --password 'D  ' https://gi
thub.com/dev-sec/windows-baseline
[2017-02-27T10:31:20-05:00] WARN: URL target https://github.com/dev-sec/windows-baseline transformed to
 https://github.com/dev-sec/windows-baseline/archive/master.tar.gz. Consider using the git fetcher

Profile: DevSec Windows Security Baseline (windows-baseline)
Version: 1.0.1
Target:  winrm://Administrator@http://54.174.127.151:5985/wsman:3389

  ×  cis-enforce-password-history-1.1.1: 1.1.1 Set Enforce password history to 24 or more passwords (ex
pected: >= 24
         got:    0)
     ×  Security Policy PasswordHistorySize should be >= 24
     expected: >= 24
          got:    0
  ✔  cis-maximum-password-age-1.1.2: 1.1.2 Set Maximum password age to 60 or fewer days, but not 0
     ✔  Security Policy MaximumPasswordAge should be <= 60
     ✔  Security Policy MaximumPasswordAge should be > 0
  ×  cis-minimum-password-age-1.1.3: 1.1.3 Set Minimum password age to 1 or more days (expected: >= 1
         got:    0)
     ×  Security Policy MinimumPasswordAge should be >= 1
     expected: >= 1
          got:    0
  ×  cis-minimum-password-length-1.1.4: 1.1.4 Set Minimum password length to 14 or more characters (exp
ected: >= 14
         got:    0)
     ×  Security Policy MinimumPasswordLength should be >= 14
     expected: >= 14
          got:    0

  <output truncated>

  ×  windows-base-203: Enable Strong Encryption for Windows Network Sessions on Servers (1 failed)
     ✔  Registry Key HKLM\System\CurrentControlSet\Control\Lsa\MSV1_0 should exist
     ×  Registry Key HKLM\System\CurrentControlSet\Control\Lsa\MSV1_0 NtlmMinServerSec should eq 537395
200

     expected: 537395200
          got: 536870912

     (compared using ==)


Profile Summary: 14 successful, 28 failures, 0 skipped
Test Summary: 27 successful, 34 failures, 0 skipped
michael@ricardo:~$
```

Figure 3 **Example of an InSpec Run**

compliance. It's worth noting that the server being tested is the default Windows Server 2016 image offered by a major cloud provider, so you can immediately see how InSpec gives you visibility into how well your network conforms to your company's security policies.

If you look at the actual InSpec rule for the first failing test, cis-enforce-password-history-1.1.1, you can see how the rule translates into something actionable:

```
control 'cis-enforce-password-history-1.1.1' do
  impact 0.7
  title '1.1.1 Set Enforce password history to 24 or more passwords'
  desc 'Set Enforce password history to 24 or more passwords'
  describe security_policy do
    its('PasswordHistorySize') { should be >= 24 }
  end
end
```

The test fails because policy requires that there be a password history of at least 24 entries, but, in fact, no history is kept at all. Obviously, the current configuration setting needs to be changed to comply with the rule.

Using InSpec with Automated Release Pipelines

InSpec can, by itself, help you manage the compliance of your systems, but InSpec can also run as a series of automated tests that execute as part of your standard release pipelines. InSpec tests can be easily added to act as a quality gate for compliance. In this section, I'll use InSpec with Chef Automate.

Chef Automate is an integrated solution for managing and deploying infrastructure and applications. It rests on a foundation of open source products that include InSpec and Chef, which is for infrastructure automation. Chef Automate provides an automated pipeline for change management and includes features for ensuring the visibility of those changes.

With Chef Automate, you can run your InSpec compliance tests on demand, see the results on the dashboard, and remediate the problem. You can also generate audit reports whenever you need them.

For example, patch management is one of the most critical aspects of IT security. It's important that you be able to identify out-of-date systems and upgrade them. Most regulatory frameworks, such as the Payment Card Industry Data Security Standard (PCI DSS), require it. To ensure that your systems are current, you can use Chef Automate to manage the entire process, from the initial identification to remediation.

You can first scan your systems to see if they're in compliance with policy and their software is up-to-date. You'll receive a report telling you the status of your infrastructure. **Figure 4** shows an example of such a report. It shows the status of the servers in a network, in terms of how well they meet compliance requirements.

Once you have the report, you can use the Chef DK to build your remediation and then test it locally before you deploy them to production. Chef DK contains all the tools you need to create and test your code.

> Chef Automate is an integrated solution for managing and deploying infrastructure and applications.

After you're satisfied with the changes, you can send them through the Chef Automate pipeline to deploy the remediation. The pipeline contains stages for testing your changes and making sure they work. Within the pipeline are two manual gates. One of them is for code review and the other sends the code to the release environments. You can involve compliance and security officers at either or both of these points to make sure they're actively engaged in the release process.

Finally, when the changes have passed all the stages in the pipeline, you can send them to the Chef server. The Chef server can then begin to bring the nodes up-to-date. Chef Automate gives you visibility into everything that's happening in your infrastructure once the changes are deployed.

Automating Compliance with InSpec

One of the largest banks in India has begun using InSpec in its Banking Services division, which is responsible for most of the bank's transactions. Compliance is particularly critical for it. The division has approximately 500 HP-UX servers that make up its development, test and production environments.

Of course, there are many regulatory and security guidelines the bank must follow and each month the team checks to make sure its servers are compliant. There are around 100 checks and, before InSpec, they were performed manually. The process was very difficult. The team had to log in to each machine, check the configuration settings, provide the results on paper and then log them. Completing a single check took about 5 minutes, so vetting just one server took about 8 hours.

When the team began using automated compliance with InSpec, the impact was evident. It could see the entire scan result in minutes. The team could see how many servers were in compliance, how many weren't in compliance, and based on that it could make quick decisions. What had taken 500 minutes to perform on one server what could now be performed in 2 minutes.

InSpec also made it much easier to satisfy the bank's auditors. IT auditors sometimes asked to see the status of a particular machine and retrieving the information was slow. Team members had to run scripts manually, get the output and make it suitable for a report. Now, with a single click, the team could instantly show the auditor what checks have been performed.

Also, InSpec is human readable and easy to learn. Most vendors for security and auditing use a binary format and the tools are difficult to use. When the banking team members saw InSpec, they felt that they could easily learn it within a few days because the learning curve was very small. (You can read about this on the Learn Chef Web site at bit.ly/2mGthmE.)

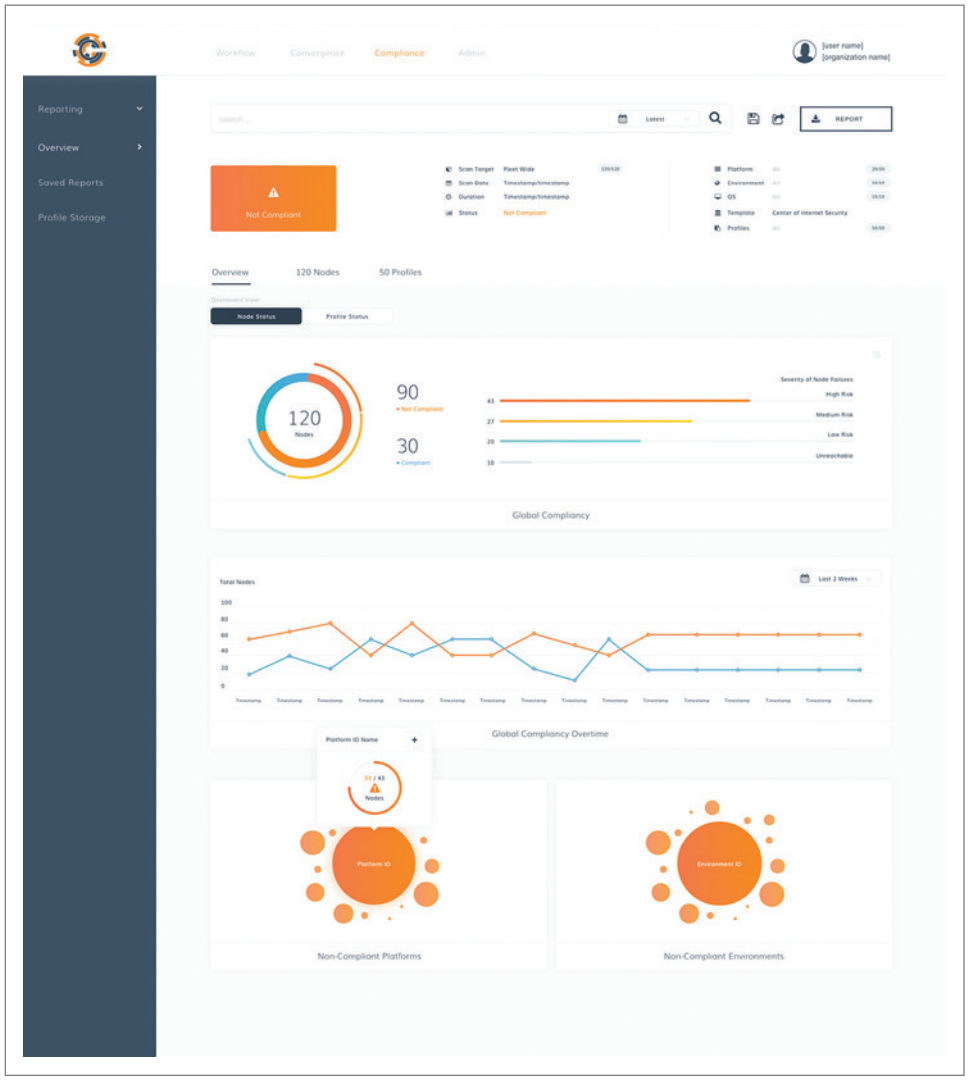


Figure 4 **Example of a Compliance Report**

Wrapping Up

InSpec is an open source testing language that lets you treat compliance as code. When compliance is code, rules are unambiguous and can be understood by everyone on the team. Developers know what standards they're expected to meet and auditors know exactly what's being tested. With InSpec, you can replace documents and manual checklists with programmatic tests that have a clear intent.

You can also integrate your compliance tests into your deployment pipeline and automatically test for adherence to security policies. Run tests as often as you need, start testing for compliance on every change and catch problems earlier in the development process, well before you've released to production. ■

Michael Ducy *is director of Open Source Product Marketing for Chef Software. He's used, managed and advocated for open source software for almost 20 years. Ducy has held a number of roles in technology from Linux systems engineer and IT instructor, to presales engineer and more. He's always interested in engaging with the broader community and can be found on Twitter: @mfdii.*

Thanks to the following technical experts for reviewing this article: *Bakh Inamov, Adam Leff and Roberta Leibovitz*

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

6 LOCATIONS TO CHOOSE FROM

JOIN US

ROCK YOUR CODE TOUR 2017

AUSTIN
MAY 15-18

Austin
May 15-18
See pages 64-65

WASH. DC
JUNE 12-15

Washington, DC
June 12-15
See pages 34-35

REDMOND
AUG 14-18

Redmond
August 14-18
See pages 36-37

CHICAGO
SEPT 18-21

Chicago
September 18-21
See pages 78-79

ANAHEIM
OCT 16-19

Anaheim
October 16-19
See pages 70-71

ORLANDO
NOV 13-17

Orlando
November 12-17
See pages 74-75

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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

vslive.com

TURN THE PAGE FOR MORE EVENT DETAILS →

DC AGENDA AT-A-GLANCE

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

START TIME	END TIME	Visual Studio Live! Pre-Conference Workshops: Monday, June 12, 2017 *(Separate entry fee required)*			
7:30 AM	9:00 AM	Pre-Conference Workshop Registration - Coffee and Morning Pastries			
9:00 AM	6:00 PM	**M01** Workshop: Distributed Cross-Platform Application Architecture - *Jason Bock & Rockford Lhotka*	**M02** Workshop: Practical ASP.NET DevOps with VSTS or TFS - *Brian Randell*	**M03** Workshop: SQL Server 2016 for Developers - *Andrew Brust & Leonard Lobel*	
6:45 PM	9:00 PM	Dine-A-Round			
START TIME	**END TIME**	**Visual Studio Live! Day 1: Tuesday, June 13, 2017**			
7:00 AM	8:00 AM	Registration - Coffee and Morning Pastries			
8:00 AM	9:00 AM	**KEYNOTE:** Microsoft PowerApps and Flow—Building Apps that Mean Business, with Low to No Code - **Saurabh Pant,** *Principal Product Manager, Customer and Partner Success Team, Microsoft*			
9:15 AM	10:30 AM	**T01** Go Mobile With C#, Visual Studio, and Xamarin - *Kevin Ford*	**T02** Build Better JavaScript Apps with TypeScript - *Brian Noyes*	**T03** What's New in Azure IaaS v2 - *Eric D. Boyd*	**T04** Tactical DevOps with VSTS - *Brian Randell*
10:45 AM	12:00 PM	**T05** Conquer the Network - Making Resilient and Responsive Connected Mobile C# Apps - *Roy Cornelissen*	**T06** Getting Started with Aurelia - *Brian Noyes*	**T07** Cloud Oriented Programming - *Vishwas Lele*	**T08** PowerShell for Developers - *Brian Randell*
12:00 PM	1:30 PM	Lunch - Visit Exhibitors			
1:30 PM	2:45 PM	**T09** Lessons Learned from Real World Xamarin.Forms Projects - *Nick Landry*	**T10** Getting to the Core of .NET Core - *Adam Tuliper*	**T11** To Be Announced	**T12** New SQL Server 2016 Security Features for Developers - *Leonard Lobel*
3:00 PM	4:15 PM	**T13** Xamarin vs. Cordova - *Sahil Malik*	**T14** Assembling the Web - A Tour of WebAssembly - *Jason Bock*	**T15** Go Serverless with Azure Functions - *Eric D. Boyd*	**T16** No Schema, No Problem! Introduction to Azure DocumentDB - *Leonard Lobel*
4:15 PM	5:30 PM	Welcome Reception			
START TIME	**END TIME**	**Visual Studio Live! Day 2: Wednesday, June 14, 2017**			
7:30 AM	8:00 AM	Registration - Coffee and Morning Pastries			
8:00 AM	9:15 AM	**W01** To Be Announced	**W02** Creating Reactive Applications With SignalR - *Jason Bock*	**W03** Bots are the New Apps: Building Bots with ASP.NET WebAPI & Language Understanding - *Nick Landry*	**W04** What's New in Visual Studio 2017 - *Robert Green*
9:30 AM	10:45 AM	**W05** Write Once Run Everywhere, Cordova, Electron, and Angular2 - *Sahil Malik*	**W06** Explore Web Development with Microsoft ASP.NET Core 1.0 - *Mark Rosenberg*	**W07** Exploring C# 7 New Features - *Adam Tuliper*	**W08** Get Started with Git and GitHub - *Robert Green*
11:00 AM	12:00 PM	General Session: To Be Announced			
12:00 PM	1:30 PM	Birds-of-a-Feather Lunch - Visit Exhibitors			
1:30 PM	2:45 PM	**W09** Distributed Architecture: Microservices and Messaging - *Rockford Lhotka*	**W10** Angular 101: Part 1 - *Deborah Kurata*	**W11** A Busy Developer's Intro to Windows Containers - *Vishwas Lele*	**W12** Continuous Delivery 3.0, The Next Step - *Marcel de Vries*
3:00 PM	4:15 PM	**W13** Agile Failures: Stories From The Trenches - *Philip Japikse*	**W14** Angular 101: Part 2 - *Deborah Kurata*	**W15** Use Docker to Develop, Build and Deploy Applications, a Primer - *Mark Rosenberg*	**W16** Mobile DevOps Demystified with Xamarin, VSTS and HockeyApp - *Roy Cornelissen*
4:30 PM	5:45 PM	**W17** Top 10 Ways to Go from Good to Great Scrum Master - *Benjamin Day*	**W18** SASS and CSS for Developers - *Robert Boedigheimer*	**W19** Microservices with Azure Container Service & Service Fabric - *Vishwas Lele*	**W20** A/B Testing, Canary Releases and Dark Launching, Implementing Continuous Delivery on Azure - *Marcel de Vries*
6:45 PM	10:30 PM	Visual Studio Live! Monuments by Moonlight Tour			
START TIME	**END TIME**	**Visual Studio Live! Day 3: Thursday, June 15, 2017**			
7:30 AM	8:00 AM	Registration - Coffee and Morning Pastries			
8:00 AM	9:15 AM	**TH01** Let's Write a Windows 10 App: A Basic Introduction to Universal Apps - *Billy Hollis*	**TH02** JavaScript for the C# (and Java) Developer - *Philip Japikse*	**TH03** Entity Framework Core for Enterprise Applications - *Benjamin Day*	**TH04** Overcoming the Challenges of Mobile Development in the Enterprise - *Roy Cornelissen*
9:30 AM	10:45 AM	**TH05** Implementing the MvvM Pattern in Your Xamarin Apps - *Kevin Ford*	**TH06** Integrating AngularJS & ASP.NET MVC - *Miguel Castro*	**TH07** Power BI: Analytics for Desktop, Mobile and Cloud - *Andrew Brust*	**TH08** Exploring Microservices in a Microsoft Landscape - *Marcel de Vries*
11:00 AM	12:15 PM	**TH09** Building Cross-Platform Business Apps with CSLA. NET - *Rockford Lhotka*	**TH10** Debugging Your Website with Fiddler and Chrome Developer Tools - *Robert Boedigheimer*	**TH11** Big Data with Hadoop, Spark and Azure HDInsight - *Andrew Brust*	**TH12** End-to-End Dependency Injection & Testable Code - *Miguel Castro*
12:15 PM	1:15 PM	Lunch			
1:15 PM	2:30 PM	**TH13** Creating Great Looking Android Applications Using Material Design - *Kevin Ford*	**TH14** Building Single Page Web Applications Using Aurelia.js and the MVVM Pattern - *Ben Hoelting*	**TH15** Using Cognitive Services in Business Applications - *Michael Washington*	**TH16** Extend and Customize the Visual Studio Environment - *Walt Ritscher*
2:45 PM	4:00 PM	**TH17** Take the Tests: Can You Evaluate Good and Bad Designs? - *Billy Hollis*	**TH18** Tools for Modern Web Development - *Ben Hoelting*	**TH19** Introduction to Azure Machine Learning Studio (for the Non-Data Scientist) - *Michael Washington*	**TH20** Windows Package Management with NuGet and Chocolatey - *Walt Ritscher*

Speakers and sessions subject to change

vslive.com/dcmsdn

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

REDMOND
AUGUST 14-18, 2017
MICROSOFT HEADQUARTERS

JOIN US AT MICROSOFT HEADQUARTERS THIS SUMMER

SUNDAY, AUG 13: PRE-CON HANDS-ON LABS

Choose From:

- Angular
- Dev Ops with ASP.NET Core/EF Core
- SQL Server 2016

NEW! Only $595!

SPACE IS LIMITED

- Rub elbows with blue badges
- Experience life on campus
- Enjoy lunch in the Commons and visit the Company Store
- And More!

Scott Hanselman, Keynote Speaker

MICROSOFT SET LIST: DETAILS DROPPING SOON!

With all of the announcements sure to come out of Build, we'll be finalizing the FULL Track of Microsoft-led sessions shortly.

Microsoft Speakers are noted with a

Be sure to check vslive.com/redmondmsdn for session updates!

ROCK YOUR CODE TOUR 2017

Register NOW and Save $400!

Use promo code VSLRED2

REGISTER NOW

START TIME	END TIME
8:00 AM	9:00 AM
9:00 AM	6:00 PM
START TIME	**END TIME**
7:00 AM	8:00 AM
8:00 AM	12:00 PM
12:00 PM	2:00 PM
2:00 PM	5:30 PM
7:00 PM	9:00 PM
START TIME	**END TIME**
7:30 AM	8:00 AM
8:00 AM	9:15 AM
9:30 AM	10:45 AM
10:45 AM	11:15 AM
11:15 AM	12:15 PM
12:15 PM	1:30 PM
1:30 PM	2:45 PM
3:00 PM	4:15 PM
4:15 PM	5:45 PM
START TIME	**END TIME**
7:30 AM	8:00 AM
8:00 AM	9:15 AM
9:30 AM	10:45 AM
11:00 AM	12:00 PM
12:00 PM	1:30 PM
1:30 PM	2:45 PM
2:45 PM	3:15 PM
3:15 PM	4:30 PM
6:15 PM	8:30 PM
START TIME	**END TIME**
7:30 AM	8:00 AM
8:00 AM	9:15 AM
9:30 AM	10:45 AM
11:00 AM	12:15 PM
12:15 PM	2:15 PM
2:15 PM	3:30 PM
3:45 PM	5:00 PM
START TIME	**END TIME**
7:30 AM	8:00 AM
8:00 AM	12:00 PM
1:00 PM	5:00 PM

EVENT PARTNER Microsoft

GOLD SPONSOR Syncfusion

SUPPORTED BY Visual Studio msdn magazine Visual Studio MAGAZINE

PRODUCED BY 1105 MEDIA YOUR GROWTH. OUR BUSINESS.

REDMOND AGENDA AT-A-GLANCE

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

NEW Full Day Hands-On Labs: Sunday, August 13, 2017 *(Separate entry fee required)*

Pre-Conference HOL Workshop Registration - Coffee and Morning Pastries

HOL01 Full Day Hands-On Lab: Busy Developer's HOL on Angular - *Ted Neward*	**HOL02** Full Day Hands-On Lab: DevOps with ASP.NET Core and EF Core - *Benjamin Day & Brian Randell*	**HOL03** Full Day Hands-On Lab: Developer Dive into SQL Server 2016 - *Leonard Lobel*

Visual Studio Live! Pre-Conference Workshops: Monday, August 14, 2017 *(Separate entry fee required)*

Pre-Conference Workshop Registration - Coffee and Morning Pastries

M01 Workshop: Modern Security Architecture for ASP.NET Core - *Brock Allen*	**M02** Workshop: Distributed Cross-Platform Application Architecture - *Jason Bock & Rockford Lhotka*	**M03** Workshop: Big Data, BI and Analytics on The Microsoft Stack - *Andrew Brust*

Lunch @ The Mixer - Visit the Microsoft Company Store & Visitor Center

M01 Workshop Continues - *Brock Allen*	**M02** Workshop Continues - *Jason Bock & Rockford Lhotka*	**M03** Workshop Continues - *Andrew Brust*

Dine-A-Round Dinner

Visual Studio Live! Day 1: Tuesday, August 15, 2017

Registration - Coffee and Morning Pastries

T01 Go Mobile With C#, Visual Studio, and Xamarin - *James Montemagno*	**T02** Angular 101: Part 1 - *Deborah Kurata*	**T03** New SQL Server 2016 Security Features for Developers - *Leonard Lobel*	**T04** What's New Visual Studio 2017 - *Robert Green*	**T05** Microsoft Set List: Details Dropping Soon
T06 Building Connected and Disconnected Mobile Apps - *James Montemagno*	**T07** Angular 101: Part 2 - *Deborah Kurata*	**T08** No Schema, No Problem! Introduction to Azure DocumentDB - *Leonard Lobel*	**T09** Getting to the Core of .NET Core - *Adam Tuliper*	**T10** Microsoft Set List: Details Dropping Soon

Sponsored Break - Visit Exhibitors

KEYNOTE: To Be Announced – *Scott Hanselman, Principal Community Architect for Web Platform and Tools, Microsoft*

Lunch - Visit Exhibitors

T11 Take the Tests: Can You Evaluate Good and Bad Designs? - *Billy Hollis*	**T12** Assembling the Web - A Tour of WebAssembly - *Jason Bock*	**T13** Unit Testing & Test-Driven Development (TDD) for Mere Mortals - *Benjamin Day*	**T14** To Be Announced	**T15** Microsoft Set List: Details Dropping Soon
T16 A Developers Introduction to HoloLens - *Billy Hollis & Brian Randell*	**T17** To Be Announced	**T18** Spans, Memory, and Channels - Making .NET Code Fast - *Jason Bock*	**T19** Entity Framework Core for Enterprise Applications - *Benjamin Day*	**T20** Microsoft Set List: Details Dropping Soon

Microsoft Ask the Experts & Exhibitor Reception – Attend Exhibitor Demos, *Sponsored by* smartsheet

Visual Studio Live! Day 2: Wednesday, August 16, 2017

Registration - Coffee and Morning Pastries

W01 Roll Your Own Dashboard in XAML - *Billy Hollis*	**W02** Migrating to ASP.NET Core - A True Story - *Adam Tuliper*	**W03** Hacker Trix - Learning from OWASP Top 10 - *Mike Benkovich*	**W04** Distributed Architecture: Microservices and Messaging - *Rockford Lhotka*	**W05** Microsoft Set List: Details Dropping Soon
W06 Customizing Your UI for Mobile Devices: Techniques to Create a Great User Experience - *Laurent Bugnion*	**W07** Securing Web APIs in ASP.NET Core - *Brock Allen*	**W08** From Containers to Data in Motion, Tour d'Azure 2017 - *Mike Benkovich*	**W09** Agile: You Keep Using That Word... - *Philip Japikse*	**W10** Microsoft Set List: Details Dropping Soon

General Session: To Be Announced

Birds-of-a-Feather Lunch - Visit Exhibitors

W11 Building Cross-platform App. Dev. with CLSA.NET - *Rockford Lhotka*	**W12** User Authentication for ASP.NET Core MVC Applications - *Brock Allen*	**W13** Tactical DevOps with VSTS - *Brian Randell*	**W14** Agile Failures: Stories from The Trenches - *Philip Japikse*	**W15** TypeScript and the Future of JavaScript - *Jordan Matthiesen & Bowden Kelly*

Sponsored Break - Exhibitor Raffle @ 2:55 pm (Must be present to win)

W16 Building Truly Universal Applications with Windows, Xamarin and MVVM - *Laurent Bugnion*	**W17** Integrating AngularJS & ASP.NET MVC - *Miguel Castro*	**W18** Get Started with Git and GitHub - *Robert Green*	**W19** SOLID – The Five Commandments of Good Software - *Chris Klug*	**W20** Using Angular 2, JavaScript, and TypeScript to Build Fast and Secure Mobile Apps - *Jordan Matthiesen*

Set Sail! VSLive!'s Seattle Sunset Cruise - Advanced Reservation & $10 Fee Required

Visual Studio Live! Day 3: Thursday, August 17, 2017

Registration - Coffee and Morning Pastries

TH01 Lessons Learned from Real World Xamarin.Forms Projects - *Nick Landry*	**TH02** Build Real-Time Websites and Apps with SignalR - *Rachel Appel*	**TH03** "Aurelia vs "just Angular" a.k.a "The Framework Formerly Known as Angular 2" - *Chris Klug*	**TH04** Go Serverless with Azure Functions - *Eric D. Boyd*	**TH05** Microsoft Set List: Details Dropping Soon
TH06 Creating Great Looking Android Applications Using Material Design - *Kevin Ford*	**TH07** Hard Core ASP.NET Core - *Rachel Appel*	**TH08** Database Lifecycle Management and the SQL Server Database - *Brian Randell*	**TH09** Breaking Down Walls with Modern Identity - *Eric D. Boyd*	**TH10** Microsoft Set List: Details Dropping Soon
TH11 Software Engineering in an Agile Environment - *David Corbin*	**TH12** Enriching MVC Sites with Knockout JS - *Miguel Castro*	**TH13** Power BI: Analytics for Desktop, Mobile and Cloud - *Andrew Brust*	**TH14** Bots are the New Apps: Building Bots with ASP.NET WebAPI & Language Understanding - *Nick Landry*	**TH15** Microsoft Set List: Details Dropping Soon

Lunch @ The Mixer - Visit the Microsoft Company Store & Visitor Center

TH16 Classic Software Design Principles and Why They Are Still Important - *David Corbin*	**TH17** Getting Started with Aurelia - *Brian Noyes*	**TH18** Big Data with Hadoop, Spark and Azure HDInsight - *Andrew Brust*	**TH19** Extend and Customize the Visual Studio Environment - *Walt Ritscher*	**TH20** Microsoft Set List: Details Dropping Soon
TH21 End-to-End Dependency Injection & Testable Code - *Miguel Castro*	**TH22** Securing Client Apps with IdentityServer - *Brian Noyes*	**TH23** Continuous Integration and Deployment for Mobile using Azure Services - *Kevin Ford*	**TH24** Windows Package Management with NuGet and Chocolatey - *Walt Ritscher*	**TH25** Microsoft Set List: Details Dropping Soon

Visual Studio Live! Post-Conference Workshops: Friday, August 18, 2017 *(Separate entry fee required)*

Post-Conference Workshop Registration - Coffee and Morning Pastries

F01 Workshop: Building Modern Web Apps with Azure - *Eric D. Boyd*	**F02** Workshop: Data-Centric Single Page Apps with Aurelia, Breeze, and Web API - *Brian Noyes*
F01 Workshop Continues - *Eric D. Boyd*	**F02** Workshop Continues - *Brian Noyes*

Speakers and sessions subject to change

vslive.com/redmondmsdn

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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

Optimize Telemetry with Application Insights

Sergey Kanzhelev and Victor Mushkatin

The importance of monitoring your service is self-evident. In this article, we'll focus on fundamental techniques to make your monitoring investments manageable. For the purpose of this discussion, "manageable" means that whatever telemetry you collect about your service, while actionable, doesn't consume an unreasonable amount of resources.

When everything works smoothly, you don't necessarily care about terabytes of log data collected for the service execution. You only care about the general trends. However, when the service goes down or performs poorly, you need everything—and then some—to diagnose the issues. How do you keep the balance between data that's required to detect the problem and needs to be collected all the time, from the data that's required to troubleshoot the problem and needs to be collected, well, when it needs to be collected?

This article discusses:

- Application Insights data model and extensibility points
- Four main techniques for telemetry data reduction
- How to improve the monitoring data ROI for high-volume applications
- Ways to collect statistically representable subsets of application telemetry

Technologies discussed:

Application Insights, Web Applications, Monitoring, Tracing

Code download available at:

bit.ly/2nhwfxn

To illustrate the techniques, we'll use the Microsoft Azure Application Insights Service and its highly extensible SDK model. While concepts we cover are universally applicable, our goal is to make you familiar with Application Insights SDK out-of-the-box capabilities and extensibility points that let you make your "telemetry exhaust" manageable. After reading this article, you'll be able to understand the Application Insights domain model, how telemetry is collected, and what coding techniques are available to decrease the amount of telemetry, while preserving monitoring capabilities, analytical accuracy, and diagnosing depth.

Unbound Volume of Telemetry

Take a service that processes 1 billion transactions a day. If you log all details about every transaction, you'll be able to answer all sorts of questions—for example, questions about transaction latency 95th percentile from a particular geolocation, or failure rate for users running a particular browser. In addition to these monitoring metrics, you'll be able to support users when they call and ask specific questions about their failed transaction by looking into logs. The more data you collect, the wider range of questions you can answer by analyzing application telemetry.

But it all comes with a price. Even with low prices for the telemetry collection (bit.ly/2nNhp3c), do you really need to collect 1 billion data points? Or worse, if every transaction makes one call to SQL Database, do you really need to collect an additional 1 billion data points for all SQL Database calls? If you add up all possible telemetry that might be needed in order to monitor, troubleshoot and analyze your service execution, you might find the infrastructure to support it to be quite expensive, significantly affecting ROI of the monitoring.

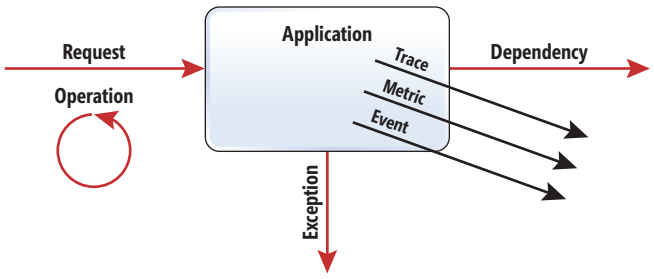


Figure 1 **Application Telemetry Data Model**

Typically, for monitoring purposes, people use various service key performance indicators (KPIs)—for example, service load, transaction latency and failure rate. In order to expose these KPIs, the monitoring system has to have an understanding of the telemetry data structure. It should be able to differentiate events that represent transaction execution from, let's say, events that represent SQL calls. With the defined semantic of telemetry, you can efficiently convert the unbound volume of telemetry into a manageable set of KPIs that'll be cheap to collect and store, and have enough data to answer your monitoring questions.

The Application Insights SDK offers the model that enables the Application Insights Service to render effective and intuitive UI to support your monitoring, troubleshooting and analytical needs, as shown in **Figure 1**.

We'll focus on two application models as part of this review—applications with an endpoint that receive external requests, typical for Web Applications, and applications that periodically "wake up" to process data stored somewhere, typical for WebJobs or Functions. In both cases, we'll call unique execution an operation. Operation succeeds or fails through exception or it might depend on other services/storage to carry its business logic. To reflect these concepts, Application Insights SDK defines three telemetry types: request, exception and dependency. For every one of these types, Telemetry Data Model defines fields used to construct common KPIs–name, duration, status code and correlation. It also lets you extend every type with the custom properties. Here are some typical fields for each of the event types:

- Request (operation id, name, URL, duration, status code, […])
- Dependencies (parent operation id, name, duration, […])
- Exception (parent operation id, exception class, call stack, […])

Typically, these types are defined by the application framework and are automatically collected by the SDK. For example, ASP.NET MVC defines the notion of a request execution in its model-view-controller plumbing–it defines when request starts and stops, dependency calls to SQL are defined by System.Data, and calls to HTTP endpoints are defined by System.Net. However, there are cases where you might need to expose telemetry unique to your application. For example, you might want to implement diagnostics logging using a familiar-to-you instrumentation framework, such as Log4Net or System.Diagnostics, or you might want to capture user interaction with your service to analyze usage patterns. Application Insights recognizes three additional data types to assist with such a need—Trace, Event and Metric:

- Trace (operation id, message, severity, […])
- Metrics (operation id, name, value, […])
- Event (operation id, name, user id, […])

In addition to data collection, Application Insights will automatically correlate all telemetry to the operation of which it's a part. For example, if while processing a request application you make some SQL Database calls, Web Services calls and recorded diagnostics info, it all will be automatically correlated with request by placing a unique auto-generated operation id into the respective telemetry payload.

The Application Insights SDK has a layered model where the previously stated telemetry types, extensibility points and data reduction algorithms are defined in the Application Insights API NuGet package (bit.ly/2n48klm). To focus discussion on core principles, we'll use this SDK to reduce the number of technology-specific data collection concepts as much as possible.

Reduction Techniques

There are four data reduction techniques available in the Application Insights SDK. As a developer, you might utilize them using a built-in extensibility API. We'll demonstrate usage of those APIs later in this article.

Metrics extraction and aggregation is a technique that lets you locally reduce data by aggregating metrics from telemetry data and sending only aggregated values, instead of the events themselves. Imagine you have 100 requests per minute. If the only thing you care about is the number of requests per minute, this technique would let you locally count the number of requests and send the value once a minute, instead of sending each request and calculating counts from the raw telemetry.

Sampling is a technique that selectively collects subsets of telemetry that lets you estimate the characteristics of the service. For most services you might collect every "n-th" request to get well-distributed statistical representation of service behavior. This technique, on the one hand, lets you reduce the volume of collection by "n" times, and on the other hand, preserves with certain accuracy statistical validity of the metrics derived from such telemetry. For better accuracy, a sophisticated algorithm and data model must be used.

Exemplification is the ability to collect samples of interest without invalidating sampling statistical accuracy. For example, you might want to always collect request failures regardless of sampling configuration. This way, while you reduce telemetry load with sampling, you can preserve useful troubleshooting data.

Filtering is the ability to reduce data by filtering out telemetry you don't care about. For example, you might want to ignore all telemetry related to traffic generated by synthetic monitoring or search bots. This way, your metrics will reflect true user interaction with the service.

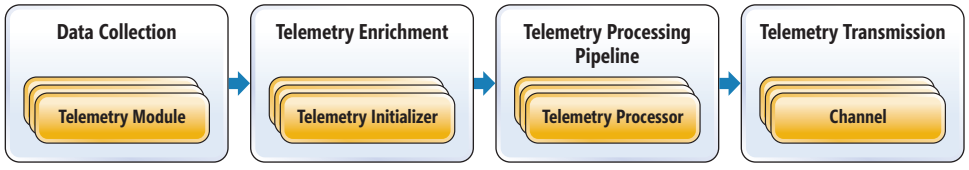


Figure 2 **How the Application Insights SDK Processes Telemetry**

TEXT CONTROL

THE TOP 5 EMR/EHR HEALTHCARE SOFTWARE VENDORS INTEGRATED TEXT CONTROL 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 seamlessly integrated into your business application.

www.textcontrol.com


Visual Studio
Partner
WE'RE
CHANGING
THE WAY
YOU LOOK AT
REPORTING
Microsoft
.NET
WINDOWS FORMS
WPF
ASP.NET MVC
ASP.NET AJAX
CLOUD WEB API

Figure 3 **Typical Code Instrumentation for the Monitoring of Loop Processing**

```
var client = new TelemetryClient(configuration);

var iteration = 0;
var http = new HttpClient();

while (!token.IsCancellationRequested)
{
  using (var operation = client.StartOperation<RequestTelemetry>("Process item"))
  {
    client.TrackEvent("IterationStarted",
      properties: new Dictionary<string, string>(){{"iteration",
      iteration.ToString()}});
    client.TrackTrace($"Iteration {iteration} started", SeverityLevel.Information);

    try
    {
      await http.GetStringAsync("http://bing.com");
    }
    catch (Exception exc)
    {
      // This call will not throw
      client.TrackException(exc);
      operation.Telemetry.Success = false;
    }

    client.StopOperation(operation);
    Console.WriteLine($"Iteration {iteration}. Elapsed time:
      {operation.Telemetry.Duration}");
    iteration++;
  }
}
```

Application Insights SDK

In order to demonstrate these reduction techniques, it's important to understand how the Application Insights SDK processes telemetry. It can be logically grouped into four stages, as shown in **Figure 2**.

Data collection is implemented as a set of telemetry modules, each responsible for particular data sets. For example, there's a telemetry module to collect dependency, exceptions, performance counters and so on.

During **telemetry enrichment**, each item is augmented with useful telemetry. For example, the Application Insights SDK will automatically add the server name as one of the properties for each telemetry item. There are sets of predefined telemetry initializers; however, developers can add any number of additional initializers to include properties that help with monitoring, troubleshooting and analytical processes. For example, for geo-distributed services, you might want to add geolocation to analyze traffic processed by each datacenter separately. Essentially, during this step you increase the payload of the telemetry items.

The telemetry processing pipeline is the place where you define logic for reducing the amount of telemetry sent to the service. The Application Insights SDK provides sampling telemetry processors to automatically reduce collected telemetry data without compromising statistical accuracy.

Telemetry transmission is a final step of the telemetry processing where all telemetry data processed by an application is queued, batched, zipped, and periodically sent to one or more destinations. The Application Insights SDK supports transmission to the Application Insights Service and other channels, such as Event Hub, out of the box.

In this article, we concentrate on techniques available to the developer to configure out-of-the-box sampling and additional telemetry processors to fine-tune data collection to service needs. All examples in this article build the monitoring configuration in code from scratch. However, in many production environments, most of the mentioned parameters are exposed as configuration settings that can be fine-tuned without the application recompilation.

Metrics Aggregation

Before going further, we want to discuss telemetry type concepts. Generally speaking, you can split all telemetry into two buckets—metrics and events.

A metric is defined as a time-series data, pre-aggregated over specified intervals. For example, say you want to count the number of invocations of a function. This is a simple metric that gets incremented each time when call to function occurs. The value of the metrics itself gets aggregated over a period of time—for example, one minute—and at the end of that time is sent out.

An event is a single record of an occurrence that's sent out every time. In many cases, events have very specific structure or type. In the example of Application Insights, the domain model event of a type Request has different properties than event of a type Exception.

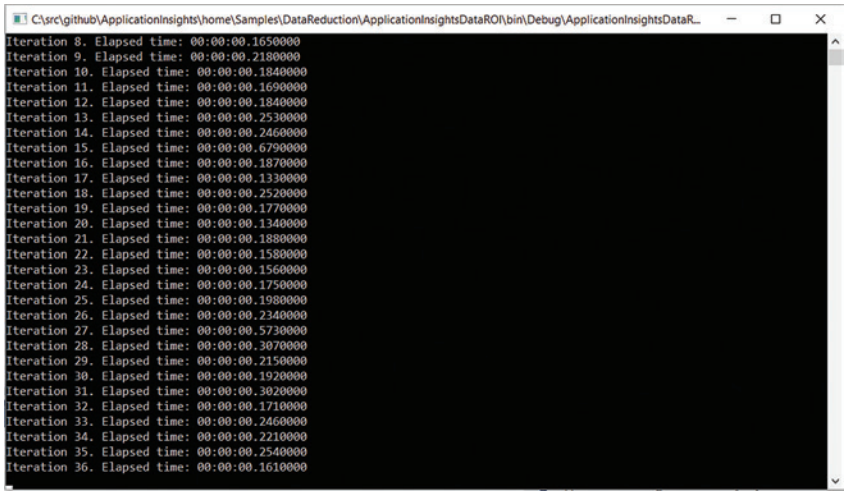


Figure 4 **Loop Processing Telemetry Output—Iteration Number and the Duration of Every Cycle**

Going back to the previous example, in case you want to capture every function execution, you might send event with function name and function parameters every time it gets executed. These events let you answer all sorts of questions about function execution. For example, with raw event telemetry, you can calculate how many times this function has been called with a particular parameter value. Notice that with more data fidelity in addition to simple analysis, such as count of function execution, you can now analyze count of execution grouped by function parameter.

While raw telemetry is much richer and lets you provide better insights, there's a drawback related to the processing and storage costs associated with that. One way to address this is to create as many metrics up front as you think you'll need to analyze your application. The problem with

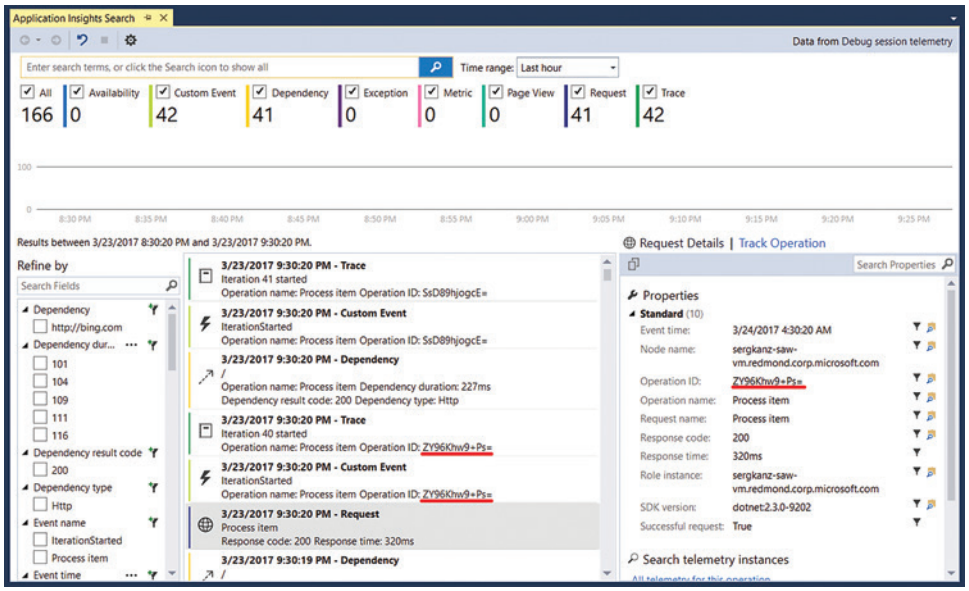


Figure 5 **Loop Processing Telemetry Output in Application Insights**

this approach is that your business is far more dynamic and it's not always possible to know what metrics you might need in advance. The Application Insights SDK addresses this issue by providing a balance between aggregated and raw telemetry—it aggregates key application performance indicators and sends sampled raw application telemetry. This sampling approach lets the SDK minimize the overhead of raw data collection and increases the ROI of the collected data.

Sampling

There are two out-of-the-box sampling telemetry processors provided by the Application Insights SDK—fixed sampling and adaptive sampling (bit.ly/2mNiDHS).

Figure 6 **Telemetry Processor for Item Size Calculation**

```
internal class SizeCalculatorTelemetryProcessor : ITelemetryProcessor
{
  private ITelemetryProcessor next;
  private Action<int> onAddSize;

  public SizeCalculatorTelemetryProcessor(ITelemetryProcessor next,
    Action<int> onAddSize)
  {
    this.next = next;
    this.onAddSize = onAddSize;
  }

  public void Process(ITelemetry item)
  {
    try
    {
      item.Sanitize();

      byte[] content =
        JsonSerializer.Serialize(new List<ITelemetry>() { item }, false);
      int size = content.Length;
      string json = Encoding.Default.GetString(content);

      this.onAddSize(size);
    }
    finally
    {
      this.next.Process(item);
    }
  }
}
```

Fixed-rate sampling reduces the volume of per-node telemetry. For example, you might want to collect only 20 percent of all telemetry from each node.

Adaptive sampling automatically adjusts the volume of per-node telemetry. For example, you might want to decrease collection when the load is greater than 5 eps/node.

Note: There's, also ingestion sampling that discards telemetry, which arrives from your app at the service ingestion endpoint. We aren't going to cover this technique in this article, but documentation can be found at bit.ly/2mNiDHS.

Both sampling telemetry processors use a common algorithm that lets you mix and match those processors without affecting statistical accuracy of the data. In order to decide if the telemetry item has to be sampled in or out, the SDK uses a stateless hash function and compares returned value with configuration ratio. It means that regardless of which thread,

Figure 7 **Building a Telemetry Processing Chain with Items Sampling**

```
// Initialize state for the telemetry size calculation
var collectedItems = 0;
var sentItems = 0;

// Build telemetry processing pipeline
configuration.TelemetryProcessorChainBuilder

  // This telemetry processor will be executed
  // first for all telemetry items to calculate the size and # of items
  .Use((next) => { return new SizeCalculatorTelemetryProcessor(next,
    size => Interlocked.Add(ref collectedItems, size)); })

  // This is a standard fixed sampling processor that'll let only 10%
  .Use((next) =>
  {
    return new SamplingTelemetryProcessor(next)
    {
      IncludedTypes = "Dependency",
      SamplingPercentage = 10,
    };
  })

  // This is a standard adaptive sampling telemetry processor
  // that will sample in/out any telemetry item it receives
  .Use((next) =>
  {
    return new AdaptiveSamplingTelemetryProcessor(next)
    {
      ExcludedTypes = "Event", // Exclude custom events from being sampled
      MaxTelemetryItemsPerSecond = 1, // Default: 5 calls/sec
      SamplingPercentageIncreaseTimeout =
        TimeSpan.FromSeconds(1), // Default: 2 min
      SamplingPercentageDecreaseTimeout =
        TimeSpan.FromSeconds(1), // Default: 30 sec
      EvaluationInterval = TimeSpan.FromSeconds(1), // Default: 15 sec
      InitialSamplingPercentage = 25, // Default: 100%
    };
  })

  // This telemetry processor will be executed ONLY when telemetry is sampled in
  .Use((next) => { return new SizeCalculatorTelemetryProcessor(next,
    size => Interlocked.Add(ref sentItems, size)); })
  .Build();
```

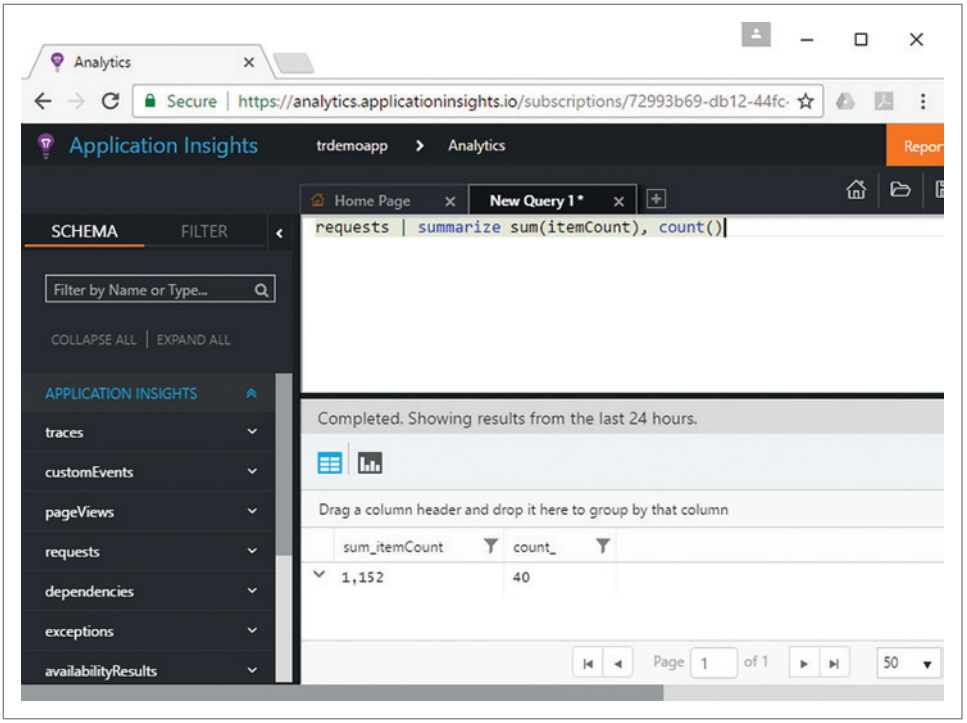


Figure 8 **Number of Telemetry Items in Application Insights and Estimated Number of Originally Collected Items**

process or server processes the data, telemetry with a hash value below threshold will be consistently sampled in. In simplified form you can codify this algorithm like this:

```
If exist(UserID): Hash(UserID) = (returns value [0..100])
ElseIf exist(OperationID): Hash(OperationID) (returns value [0..100])
Else: Random [0..100]
```

As you can see from here, as long as UserID or OperationID is shared among all related telemetry items, it all will have the same hash value and consistently be sampled in or out. Application Insights by default enables adaptive sampling when collecting data. All telemetry stored in the services has a column called itemCount. It represents the sampling ratio at the moment of data collection. Because the hash calculation algorithm is stateless, this number doesn't represent the actual number of sampled out telemetry, it only tells the statistical ratio of telemetry sampled in. To quickly analyze if your telemetry has been sampled, you can execute the following analytical query to compare the number of records stored with the number of records processed by the service:

```
requests | summarize sum(itemCount),
  count()
```

If you see the difference between those two numbers, then sampling has been enabled and your data set has been reduced.

Figure 9 **Marking Slow Dependency Calls for Collection and Exempt Them from Sampling**

```
internal class DependencyExampleTelemetryProcessor : ITelemetryProcessor
{
  private ITelemetryProcessor next;

  public DependencyExampleTelemetryProcessor(ITelemetryProcessor next)
  {
    this.next = next;
  }

  public void Process(ITelemetry item)
  {
    // Check telemetry type
    if (item is DependencyTelemetry)
    {
      var r = item as DependencyTelemetry;
      if (r.Duration > TimeSpan.FromMilliseconds(100))
      {
        // If dependency duration > 100 ms then "sample in"
        // this telemetry by setting sampling percentage to 100
        ((ISupportSampling)item).SamplingPercentage = 100;
      }
    }

    // Continue with the next telemetry processor
    this.next.Process(item);
  }
}
```

Data Reduction in Action

Let's review all these techniques. We're going to use a console application to highlight main concepts. The application pings bing.com in a loop and stores telemetry data in Application Insights. It treats each loop execution as a request telemetry, automatically collects dependency data and correlates all telemetry to the appropriate "request" to which it belongs.

To initialize the Application Insights SDK, you need to perform three simple steps. First, you have to initialize configuration with the Instrumentation Key. Instrumentation Key is an identifier used to associate telemetry data with the Application Insights resource and can be obtained in the Azure Portal when creating it:

```
// Set Instrumentation Key
var configuration = new TelemetryConfiguration();
configuration.InstrumentationKey = "fb8a0b03-235a-4b52-b491-307e9fd6b209";
```

Next, you need to initialize the Dependency Tracking module to automatically collect dependency information:

```
// Automatically collect dependency calls
var dependencies = new DependencyTrackingTelemetryModule();
dependencies.Initialize(configuration);
```

Last, you have to add Telemetry Initializer that adds common correlation id to all related telemetry:

```
// Automatically correlate all telemetry data with request
configuration.TelemetryInitializers.Add(new
  OperationCorrelationTelemetryInitializer());
```

At this point, the Application Insights SDK is fully initialized and you can access all APIs via TelemetryClient object and code the main loop, as shown in **Figure 3**.

When you execute this application, you'll see the screen shown in **Figure 4** in a console window.

All the telemetry will be sent to the cloud and can be accessed using the Azure Portal. During the development, it's easier to analyze telemetry in Visual Studio. So if you run the code in Visual Studio under debugger, you'll see telemetry right away in the Application Insights Search tab. It will look like what's shown in **Figure 5**.

Figure 10 **Filtering of Fast Dependency Calls Telemetry**

```
internal class DependencyFilteringTelemetryProcessor : ITelemetryProcessor
{
  private readonly ITelemetryProcessor next;

  public DependencyFilteringTelemetryProcessor(ITelemetryProcessor next)
  {
    this.next = next;
  }

  public void Process(ITelemetry item)
  {
    // Check telemetry type
    if (item is DependencyTelemetry)
    {
      var d = item as DependencyTelemetry;
      if (d.Duration < TimeSpan.FromMilliseconds(100))
      {
        // If dependency duration > 100 ms then stop telemetry
        // processing and return from the pipeline
        return;
      }
    }

    this.next.Process(item);
  }
}
```

Analyzing this log, for each request you can see trace, event and dependency telemetry with the same operation id. At this point, you have an app that sends various Application Insights telemetry types, automatically collects dependency calls and correlates them all to the appropriate requests. Now, let's reduce telemetry volume utilizing out-of-the-box sampling telemetry processors.

As previously stated, the Application Insights SDK defines **the telemetry processing pipeline** that's used to reduce the amount of telemetry sent to the portal. All collected telemetry enters the pipeline and every telemetry processor decides whether to pass it further along. As you'll see, configuring of sampling with the out-of-the box telemetry processors is as easy as registering them in the pipeline and requires just a couple lines of code. But in order to demonstrate the effect of those processors, we'll slightly modify the program and introduce a helper class to showcase the reduction ratio.

Let's build the Telemetry Processor that'll calculate the size of the telemetry items going through, as shown in **Figure 6**.

Now you're ready to build the telemetry processing pipeline. It will consist of four telemetry processors. The first one will calculate the size and count of telemetry sent into the pipeline. Then, you'll use the fixed sampling telemetry processor to sample only 10 percent of dependency calls (in this case, ping to bing.com). In addition, you'll enable adaptive sampling to all telemetry types, except Events. It means that all events will be collected. The last telemetry processor will calculate the size and count of the telemetry items that'll be sent to the channel for subsequent transmission to the service, as shown in **Figure 7**.

Finally, you'll slightly modify the console output to see the collected and sent telemetry and the ratio for the reduction:

```
Console.WriteLine($"Iteration {iteration}. " +
  $"Elapsed time: {operation.Telemetry.Duration}. " +
  $"Collected Telemetry: {collectedItems}. " +
  $"Sent Telemetry: {sentItems}. " +
  $"Ratio: {1.0 * collectedItems / sentItems}");
```

When executing the app you can see that the ratio may be as high as three times!

Now, if you go to the Application Insights Analytics page and execute the query mentioned here, you might see the stats shown in **Figure 8**, proving that sampling worked. You see only a few requests representing many telemetry items.

Exemplification and Filtering

So far we've talked about sampling and you've learned how to build a custom telemetry processing pipeline and simple telemetry processor. With this knowledge, you can explore two other techniques—filtering and exemplification. We made a couple of examples to showcase what you can do.

First, let's take a look at the exemplification. Let's say your application is dependent on a third-party service and it guarantees a certain performance SLA for processing requests. With the existing approach, you can collect samples of dependency calls. But what if you want to collect all evidences where that service was out

Figure 11 **Filtering of Fast Dependency Calls Telemetry with Metrics Pre-Aggregation**

```
internal class DependencyFilteringWithMetricsTelemetryProcessor
                                        : ITelemetryProcessor, IDisposable
{
  private readonly ITelemetryProcessor next;
  private readonly ConcurrentDictionary<string, Tuple<Metric, Metric>> metrics
    = new ConcurrentDictionary<string, Tuple<Metric, Metric>>();
  private readonly MetricManager manager;

  public DependencyFilteringWithMetricsTelemetryProcessor(
    ITelemetryProcessor next, TelemetryConfiguration configuration)
  {
    this.next = next;
    this.manager = new MetricManager(new TelemetryClient(configuration));
  }

  public void Process(ITelemetry item)
  {
    // Check telemetry type
    if (item is DependencyTelemetry)
    {
      var d = item as DependencyTelemetry;

      // Increment counters
      var metrics = this.metrics.GetOrAdd(d.Type, (type) =>
      {
        var dimensions = new Dictionary<string, string> { { "type", type } };
        var numberOfDependencies =
          this.manager.CreateMetric("# of dependencies", dimensions);
        var dependenciesDuration =
          this.manager.CreateMetric("dependencies duration (ms)", dimensions);
        return new Tuple<Metric, Metric>(
          numberOfDependencies, dependenciesDuration);
      });

      // Increment values of the metrics in memory
      metrics.Item1.Track(1);
      metrics.Item2.Track(d.Duration.TotalMilliseconds);

      if (d.Duration < TimeSpan.FromMilliseconds(100))
      {
        // If dependency duration > 100 ms then stop telemetry
        // processing and return from the pipeline
        return;
      }
    }

    this.next.Process(item);
  }

  public void Dispose()
  {
    this.manager.Dispose();
  }
}
```

of compliance with its SLA? For this demo purpose, we've created an exemplification telemetry processor that collects all dependency calls that are out of compliance with 100 ms SLA, as shown in **Figure 9**.

Unlike exemplification, which, in fact, increases the volume of the collected telemetry for the purpose of more precise data fidelity, filtering is more radical as it drops telemetry items on the floor, making them completely invisible to the service. For demo purposes, we've created an exemplification telemetry processor that drops all dependency calls that are faster than 100 ms, as shown in **Figure 10**.

Telemetry filtering is effective to reduce the amount of telemetry and increase its quality. When you know that the telemetry item isn't actionable, you don't want to see it in analytics. Using the telemetry processor in the previous example, you'll only see dependency calls faster than 100 ms. So if you try to calculate the average duration of the dependency processing based on dependency record, you'll get incorrect results.

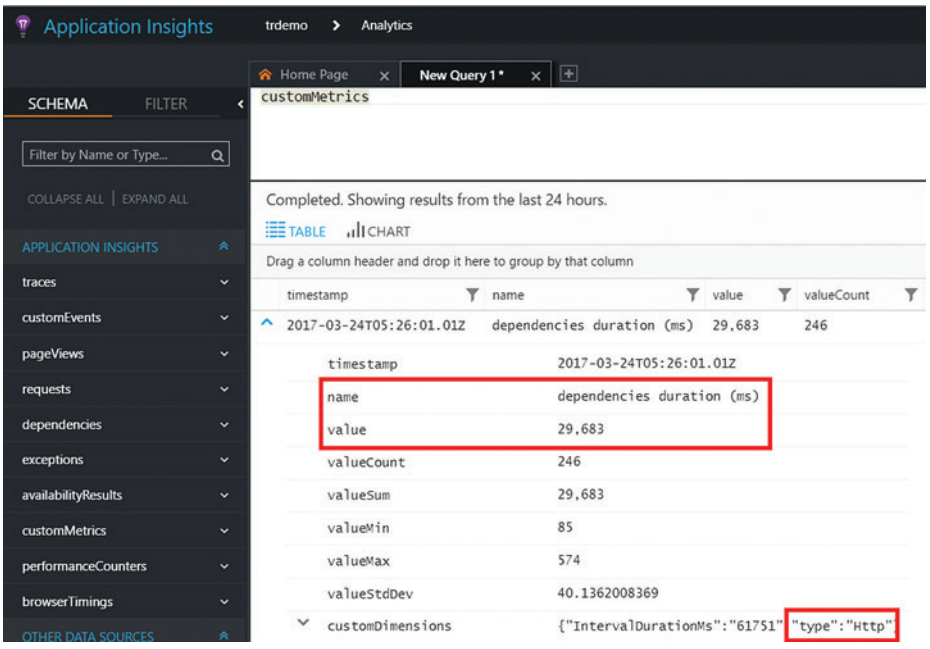


Figure 12 **Pre-Aggregated Metric Collected by Application Insights**

Let's try to address this by locally aggregating dependency call telemetry and sending "true" metrics to the service. To do so, we're going to use a new metrics API and modify the telemetry processor to expose metrics before dropping telemetry, as shown in **Figure 11**.

As you can see, we're creating two metrics—"# of dependencies" and "dependencies duration (ms)"—with dimensionality of a dependency type. In our case, all dependency calls are tagged with HTTP type. If you go to Analytics, you can see the information collected for your custom metrics, as shown in **Figure 12**.

This example lets you calculate the total number of calls and duration your app is spending while calling to dependencies. Name contains the name of the metrics, that is, dependency duration (ms); value is the sum of all http calls to bing.com; and customDimensions contains a custom dimension called type with value HTTP. There were a total of 246 calls to the Track API call; however, only one record was stored per minute for each metric. Both processing efficiency and cost are strong cases to expose app telemetry using the MetricsManager API. The challenge with this approach is that you have to define all your metrics and dimensions up front. When it's possible, it's a recommended way; however, in some cases, it's either not possible or the cardinality of the dimension is too high. In such cases, relying on sampled raw telemetry is the reasonable compromise between accuracy and telemetry volume.

Wrapping Up

Controlling the volume of monitoring telemetry is an important aspect of making good return on your monitoring investments. Over collecting will cost you too much; under collecting will prevent you from being able to effectively detect, triage and diagnose your production issues. This article discussed techniques that help you manage the data collection footprint using the Application Insights SDK and its extensibility model. Using data reduction techniques, such as sampling, filtering, metrics aggregation, and exemplification, it was demonstrated how to significantly decrease the volume of data while preserving monitoring accuracy, analytical correctness, and diagnostics depth.

The Application Insights approach is being adopted by many new Microsoft services and frameworks, such as Azure Functions and Service Fabric (this support will be announced at this year's Microsoft Build 2017 conference) and community through OSS contribution on GitHub (bit.ly/2n1GFzF). In addition to the .NET Framework, there are other SDKs available, including JavaScript, Java and Node.js (Node.js Application Insights SDK improvements like better telemetry collection and correlation, as well as easier enablement in Azure, will be announced at Build 2017). Through a consistent, extensible, cross-platform data collection SDK, you can take control and "manage" your telemetry across your heterogeneous application environment. ■

Victor Mushkatin *is a group program manager on the Application Insights team. His main area of expertise is application performance monitoring technologies and DevOps practices. Reach him at victormu@microsoft.com.*

Sergey Kanzhelev *is a principal software developer on the Application Insights team. His career has been entirely dedicated to application monitoring and diagnostics. He's passionate about connecting with customers, as well as an avid blog author and GitHub contributor. Reach him at sergkanz@microsoft.com.*

Thanks to the following technical experts for reviewing this article: *Mario Hewardt, Vance Morrison and Mark Simms*

Working with Raspberry Pi and Windows 10

Bruno Sonnino

Although I've been working with software for a long time, I've never interacted directly with hardware. I've developed a lot of software that works near the hardware, but I've never worked with a physical board where I have complete control of what's being done. Therefore, when I had the opportunity to work with the Raspberry Pi, especially using Windows 10 and Visual Studio, I jumped at the opportunity.

The Raspberry Pi, in versions 2 and 3, can use Windows 10 as its OS (though it's not the full version, it lets you execute Universal Windows Platform [UWP] apps to control its devices). This is a cheap computer—you can get one for less than $35—and it's powerful. The Raspberry Pi 3 has a Quad-Core, 64-bit ARM processor, HDMI video, Ethernet and Wi-Fi networking, Bluetooth, and four USB ports. You can definitely do many things with it.

This article discusses:

- How to develop a small project with the Raspberry Pi
- How to use the Universal Windows Platform to develop a program for the Raspberry Pi
- How to use Windows 10 programming with the Raspberry Pi

Technologies discussed:

Internet of Things, Windows 10, Universal Windows Platform, Raspberry Pi

Code download available at:

bit.ly/2lQJfaT

The Hardware

To start, you can use the Raspberry Pi board alone, but that's somewhat limiting. If you use only the board, that would be the same as developing for a computer or a smartphone. You also need to use some extra hardware. Some manufacturers have created kits to complement it: prototype boards, resistors, LEDs, potentiometers, sensors and a memory card. You can buy a case for the computer, but that's not necessary, as you should keep the computer open to make the connections.

> This is a very cheap computer—you can get one for less than $35—and it's powerful.

Knowing the Hardware

Once you have the board and the kit, it's time to get to know the hardware. Initially, you should explore the Raspberry Pi and see what it has to offer. **Figure 1** shows the board.

On the right side of the board in **Figure 1**, you can see the four USB ports (1) and the Ethernet connector (2). At the bottom, from left to right, you have the power jack in the form of a mini USB (3), the HDMI video (4), the camera port (5) and the sound output (6). On the left side of the board, you have the micro SD card slot (7) and a connector for an LCD display (8). You can also see the

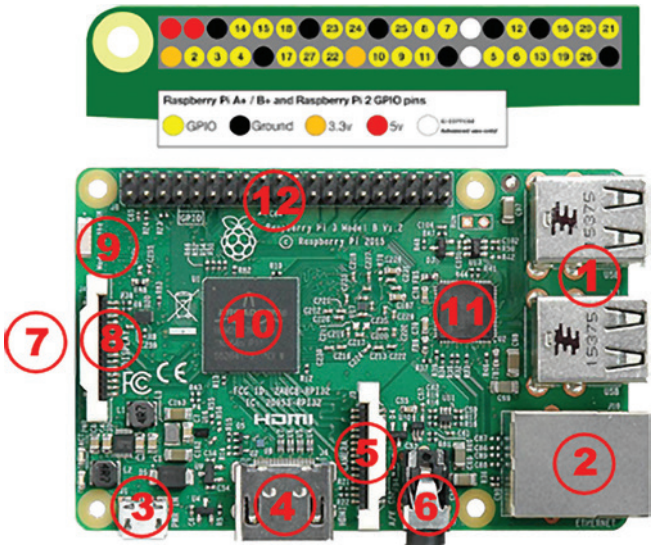


Figure 1 **Raspberry Pi 3 Model B with GPIO**

Wi-Fi and Bluetooth adapter (9). In the middle of the board, you can see the processor (10) and the network controller (11). On the upper side, you have the General Purpose Input/Output (GPIO) block (12), where you make all connections. Every pin has a different purpose, as you can see at the top of the figure.

The Raspberry Pi uses two supply tensions: 5V and 3.3V. The black pins are ground and the yellow ones are the GPIO pins that you'll use in your programming. Note that the pin numbering isn't ordered. Therefore, unless you have a perfect memory, keep a diagram like that nearby (there's one available at bit.ly/1WcBUS2).

The second step is to study the kit. I won't discuss all the content because that can change a lot, depending on the manufacturer (or from what you intend to buy). For this project, you'll need a breadboard, three LEDs, resistors and wires. To learn more about these components and how to interact with them, see Frank La Vigne's April 2016 Modern Apps column, "Writing UWP Apps for the Internet of Things," at msdn.com/magazine/mt694090.

Mounting the First Circuit

Knowing the board and these simple components, you can mount the first circuit. Usually, the "Hello World" for a system like that is a program that makes the LED blink. To make it extra simple, you'll start by creating a circuit that lights up the LED, without blinking. For that, you don't need any kind of program, only to understand the circuit you're going to build.

If you connect the LED directly to the 3.3V pin of the Raspberry Pi, you'd probably burn the LED, as it wouldn't support the current passing by it. By using Ohm's law (V = R*I), you need to add a 220 Ω (Red/Red/Black) resistor in the circuit. If you don't have a 220 Ω resistor available, you can use a larger one–with a larger resistor, there's less current in the circuit, so the LED isn't damaged. The resistor can't be much larger because if the current is too small, the LED doesn't turn on. In my case, I used a 330 Ω with no problems.

To see what the montage in the breadboard looks like, see **Figure 2**. The image was created with an open source program called "Fritzing," which can be downloaded at fritzing.org.

After mounting the circuit (this should be done with the Raspberry Pi power source off to not burn any component), connect the power source. If you mounted it correctly, the LED should turn on. If the LED doesn't turn on, check if you put the poles of the LED correctly—the positive pole (longer wire) and the connection to the 3.3V pin in the Raspberry Pi should be in the same horizontal line. The negative pole and the resistor (in this case, there is no polarization) must be in the same line. The second wire of the resistor must be connected to the line that goes to the ground pin in the Raspberry Pi. If everything is right, check if the LED is burned, and replace it with another one. When the LED turns on, you can go to the next step: creating a program that controls the LED.

Installing and Using Windows 10

Until now, you didn't need an OS because you didn't need to program the Raspberry Pi, but you'll need some programming to go on with your exploration. For that, you'll use Windows 10. You can download and install Windows 10 for the Raspberry Pi free of cost and, although it isn't exactly the same version that runs on desktops and tablets, it lets you execute programs for the UWP with no change.

> To install Windows 10 on the Raspberry Pi, you must have a compatible micro SD card with at least 8GB.

The first step is to download and install Windows 10 in the SD Card. For that, download and install the Windows 10 Internet of Things (IoT) Core Dashboard tool, located at bit.ly/2IPXrRc.

To install Windows 10 on the Raspberry Pi, you must have a compatible micro SD card with at least 8GB. Next, select the option "Setup

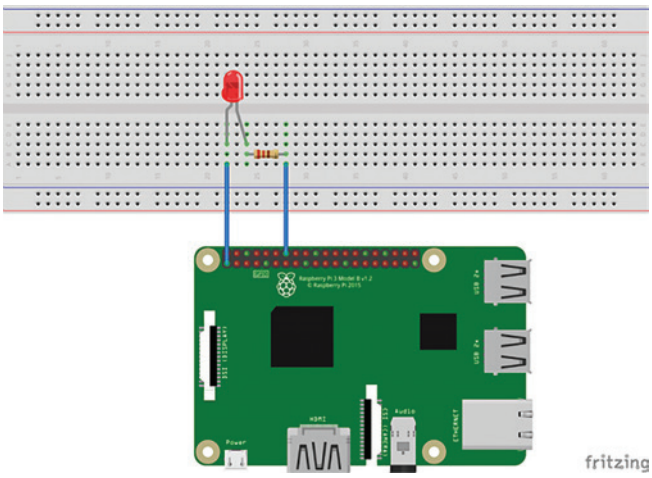


Figure 2 **Mounted Circuit**

a new device" in the dashboard to download and install Windows 10 on the SD card. You must have some way to write to this card on your computer. If you don't have a card reader available, you can buy a USB card reader.

> Once you install it and run the Windows IoT Remote Control app, you can control and interact with the device remotely.

Select the type of device, the OS and the drive where the SD card is located. Give a name for the computer and select an administrator password. Click on the box to accept the license terms and click on the Install button. After downloading and recording the data on the card, you have Windows 10 installed and ready to use. Remove it from the computer card reader and put it in the slot of the Raspberry Pi. Connect it to the network using an Ethernet cable or Wi-Fi if you're using the Raspberry Pi 3 or 2 with a Wi-Fi dongle. Turn on the device.

Once Windows 10 has booted, you can see the device connected under My Devices in the IoT Core Dashboard.

You can open the device portal in the browser using the IP Address shown for the connected device, on port 8080. In my case, I can open it with the address http://192.168.1.199:8080. It will ask you the admin password you've set before to open the portal, as shown in **Figure 3**.

Here, you can configure the device, check the installed apps, and verify its performance and storage. The last option, Remote, lets you enable remote control for the device. This option is useful if the device doesn't have a monitor attached to it, as you can control it remotely from your computer. Check the box labeled "Enable Windows IoT Remote Server" to enable the remote control on the device and download the remote control app for Windows 10 from the store.

Once you install it and run the Windows IoT Remote Control app, you can control and interact with the device remotely.

Now, you can start developing for the Raspberry Pi using Windows 10.

Developing for the Raspberry Pi Using Visual Studio

To develop for the Raspberry Pi using Visual Studio, you must make sure that you installed the tools. You can check this by selecting "Custom installation" and checking the Universal Windows App Development Tools in the Features selection.

Once you do that, you'll have the tools installed and you can start developing for the Raspberry Pi using Windows 10. Create a new project and select the "Blank" UWP app.

This will create a blank app and you'll create an app that shows the name of the current machine in the main screen. In MainPage.xaml, add the following code:

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
  <TextBlock FontSize="32" x:Name="MachineText"
    HorizontalAlignment="Center"
    VerticalAlignment="Center"/>
</Grid>
```

Then, in MainPage.xaml.cs, put this code to get and display the machine name:

```
public MainPage()

  this.InitializeComponent();
  Windows.Security.ExchangeActiveSyncProvisioning.EasClientDeviceInformation eas =
    new Windows.Security.ExchangeActiveSyncProvisioning.EasClientDeviceInformation();
  MachineText.Text = eas.FriendlyName;
}
```

If you run this app on your local machine, it will show a window with the name of your machine.

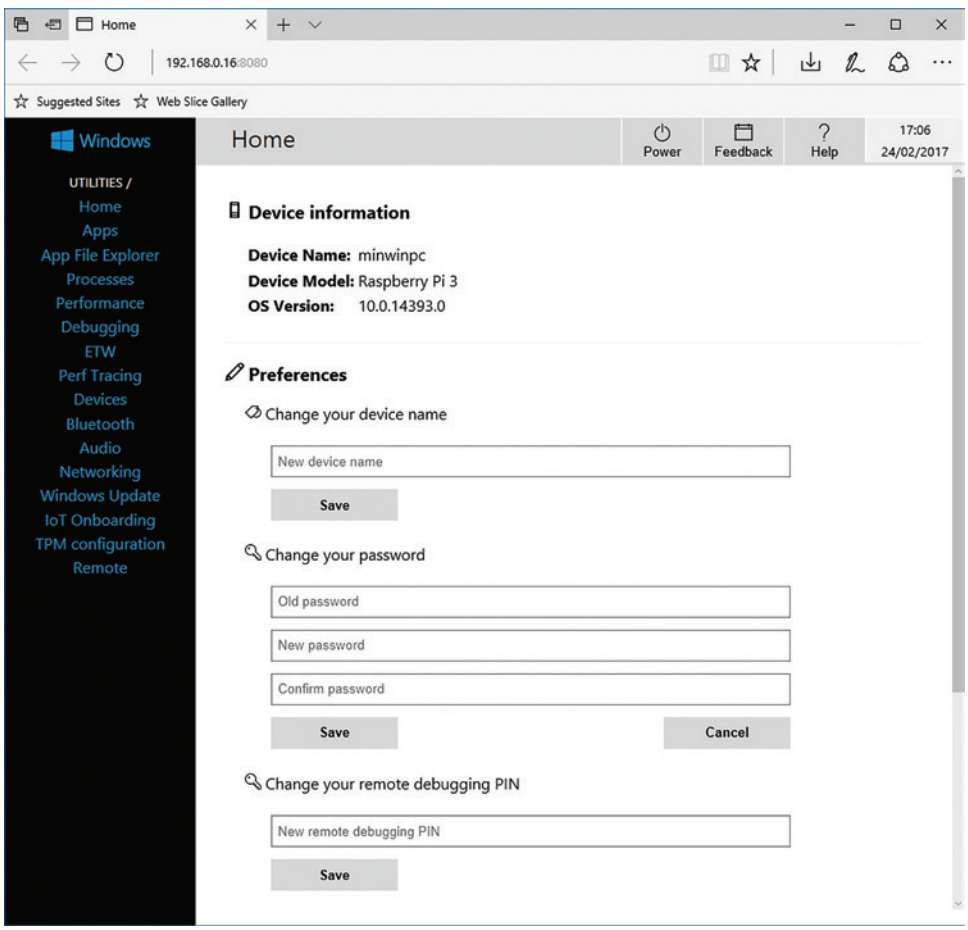


Figure 3 **Device Portal**

GROUPDOCS
Document Manipulation APIs

Manipulating Files?

APIs to view, export, annotate, compare, sign, automate and search documents in your applications.

GroupDocs.Total

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

Try for Free

.NET Libraries

Java Libraries

Cloud APIs

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

Visit us at www.groupdocs.com

GROUPDOCS
Document Manipulation APIs

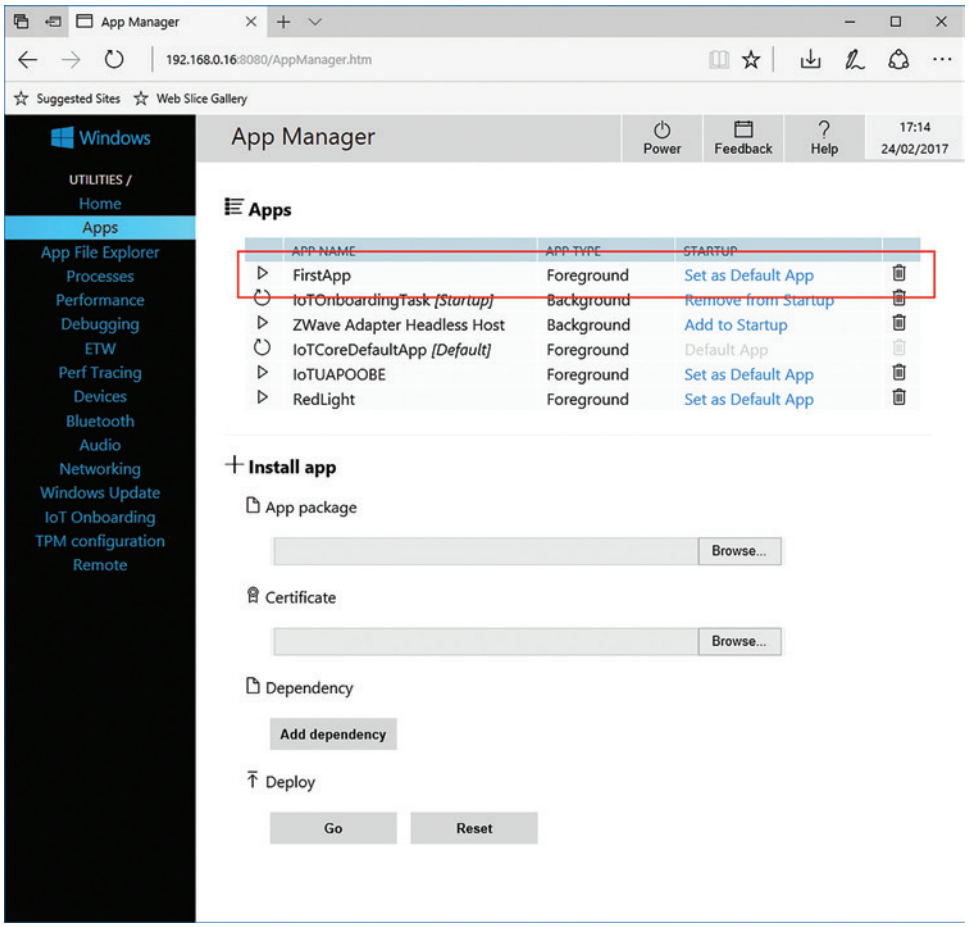


Figure 4 **App Portal Showing the Installed App**

Then, run the Raspberry Pi. On the Solution Platform dropdown, select ARM, and on the Device dropdown, select Remote Machine. A dialog box opens to select the remote machine.

Select the Raspberry Pi device and run the application. The app will be deployed to the Raspberry Pi and you can see it running in the remote control window. Note that the machine name shown in the window should be the one you have set when you formatted the SD card and installed Windows 10 on it.

You can debug this app the same way you do with local apps—set breakpoints, analyze variables and so on. If you terminate the app in Visual Studio, you'll see that the app closes and the main screen appears in the Raspberry Pi. If you go to the browser portal, you'll see that the app is still installed and can be run by using the Run button, as shown in **Figure 4**.

I'm amazed at the level of compatibility achieved with UWP apps. To show the potential, I'll use an app that wasn't made for the Raspberry Pi. I'll use the sample app for the UWP Community Toolkit, a toolkit of components developed by Microsoft and the community, which definitely is worth checking out at bit.ly/2b1PAJY.

If you download the package and compile it, you can deploy it to the Raspberry Pi and run it (yes, you can run the program in the exact same way you would on a desktop). By the way, you should try using the controls in the device—they work fine.

Interacting with the Board

Once you have your programs running, you must start interacting with the board. You'll create a traffic light controller. It will have three LEDS (red, yellow and green) and there can be different timings for each light.

To operate on the LEDs in the board, you must get the GPIO Controller and open the pin you want to control and set it the way you want. In **Figure 1**, you see that the eighth pin in the GPIO block (second row) is pin 22. You'll use pins 22, 9, and 19, and the resulting circuit will be like the one shown in **Figure 5**.

With this circuit in place, create a new blank UWP app. In MainPage.xaml, input the code shown in **Figure 6**.

You'll see the traffic lights both on the board and on the display, so you can also see what's happening by viewing the remote display. The source code in Mainpage.xaml.cs is shown in **Figure 7**.

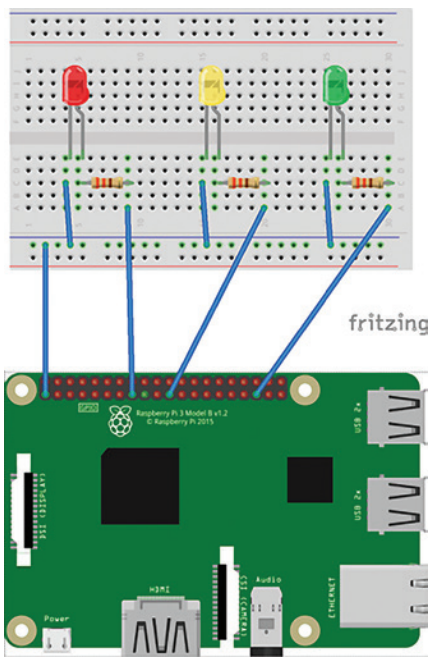


Figure 5 **Circuit for the Traffic Lights**

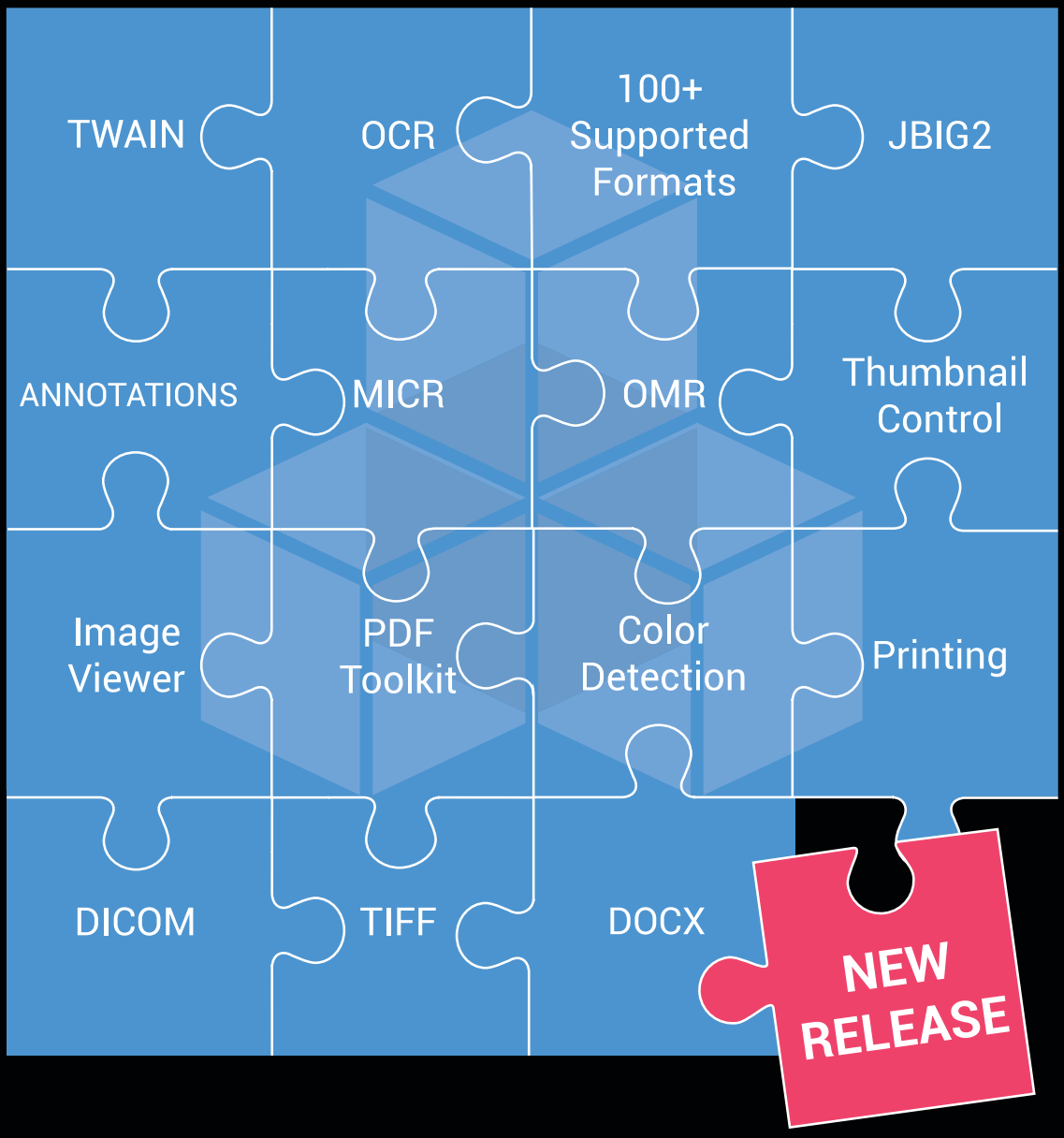
> You can run the program in the exact same way you would on a desktop.

To run this code, you must add a reference to the IoT extensions. Right-click in the References node in the Solution Explorer, click on Add References, then go to Extensions and add the Windows IoT Extensions for UWP.

GdPicture.NET 14

100% ROYALTY FREE

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



Leverage your apps. with **GdPicture.NET** Imaging Toolkit

DOWNLOAD YOUR FREE TRIAL **www.gdpicture.com**

GdPicture.NET is an ORPALIS imaging technologies product

You create a timer that will turn on each LED in an interval set by the intervals array (in your case, 6s, 2s and 6s). The ellipses in the screen have their opacity set to 0.5, so they appear dimmed and each one will be set to 1 when the light is on. The timer will only be set if you can set the GPIO for the board, in the InitGPIO function, as shown in **Figure 8**.

Figure 6 **Main Page xaml Code, Showing Traffic Lights**

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
  <Border BorderBrush="Black" BorderThickness="3" HorizontalAlignment="Center"
      VerticalAlignment="Center" CornerRadius="5">
    <StackPanel HorizontalAlignment="Center" VerticalAlignment="Center">
      <Ellipse Width="50" Height="50" Fill="Red" Opacity="0.5"
        Margin="20,20,20,10" x:Name="RedLed" Stroke="Black"
        StrokeThickness="1"/>
      <Ellipse Width="50" Height="50" Fill="Yellow" Opacity="0.5"
        Margin="20,10,20,10" x:Name="YellowLed" Stroke="Black"
        StrokeThickness="1"/>
      <Ellipse Width="50" Height="50" Fill="LimeGreen" Opacity="0.5"
         Margin="20,10,20,20" x:Name="GreenLed" Stroke="Black"
         StrokeThickness="1"/>
    </StackPanel>
  </Border>
</Grid>
```

Figure 7 **Source Code to Turn on Traffic Lights at Specified Intervals**

```
private int _currentLight;
private DispatcherTimer _timer;
private int[] _pinNumbers = new[] { 22, 9, 19 };
private GpioPin[] _pins = new GpioPin[3];

public MainPage()
{
  this.InitializeComponent();
  if (InitGPIO())
    InitTimer();
}

private void InitTimer()
{
  var intervals = new[] { 6000, 2000, 6000 };
  var lights = new[] { RedLed, YellowLed, GreenLed };
  _currentLight = 2;
  _timer = new DispatcherTimer { Interval = TimeSpan.FromMilliseconds(500) };
  _timer.Tick += (s, e) =>
  {
    _timer.Stop();
    lights[_currentLight].Opacity = 0.5;
    _pins[_currentLight].Write(GpioPinValue.High);
    _currentLight = _currentLight == 2 ? 0 : _currentLight + 1;
    lights[_currentLight].Opacity = 1.0;
    _pins[_currentLight].Write(GpioPinValue.Low);
    _timer.Interval = TimeSpan.FromMilliseconds(intervals[_currentLight]);
    _timer.Start();
  };
  _timer.Start();
}
```

Figure 8 **Code to Initialize GPIO and Set LED Pins for Output**

```
private bool InitGPIO()
{
  var gpio = GpioController.GetDefault();

  if (gpio == null)
    return false;
  for (int i = 0; i < 3; i++)
  {
    _pins[i] = gpio.OpenPin(_pinNumbers[i]);
    _pins[i].Write(GpioPinValue.High);
    _pins[i].SetDriveMode(GpioPinDriveMode.Output);
  }
  return true;
}
```

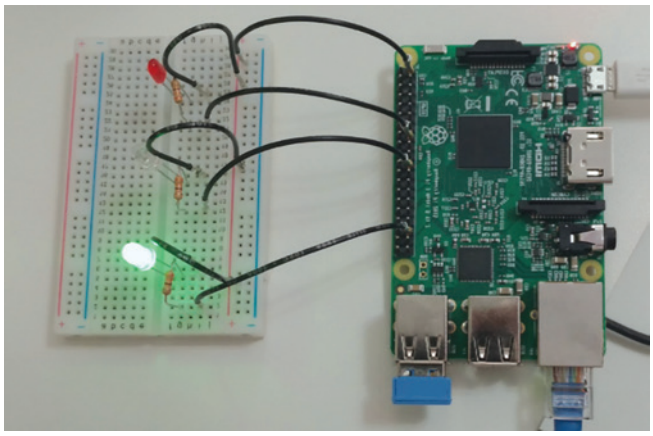


Figure 9 **Traffic Lights Board with Program Running**

You open the three pins for output and set them to High so the LEDs are turned off. When you set a pin to Low, the current will flow by the circuit and the LED will turn on. When you run the program, you'll see a screen with a traffic light, where the lights go on and off like a real one, and the board will look like the photo in **Figure 9**.

> Knowing Windows 10 programming, you already have all the knowledge you need to program a Raspberry Pi.

Wrapping Up

As you can see, it's simple to create programs that interact with the Raspberry Pi. Knowing Windows 10 programming, you already have all the knowledge you need to program a Raspberry Pi (yes, interacting with the board is a different story, but it's already halfway done). You can create your programs the same way you create any Windows 10 program (in fact, UWP programs run with no change in the Raspberry Pi). The only difference is that you have the GPIO controller to set data and get data. If you want to extend your knowledge and try other projects, there are many samples to try at bit.ly/2IlecFZ. This will open a multitude of opportunities and you'll be able to combine a powerful hardware with great and productive software. That's an unbeatable combination. ■

Bruno Sonnino *has been a Microsoft MVP since 2007. He's a developer, consultant, and author, having written many books and articles about Windows development. You can follow him on Twitter: @bsonnino or read his blog at blogs.msmvps.com/bsonnino.*

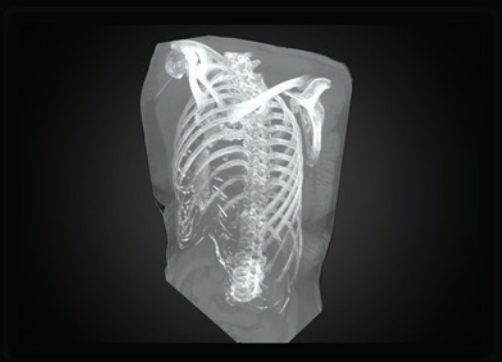
Thanks to the following Microsoft technical expert who reviewed this article: *Rachel Appel*

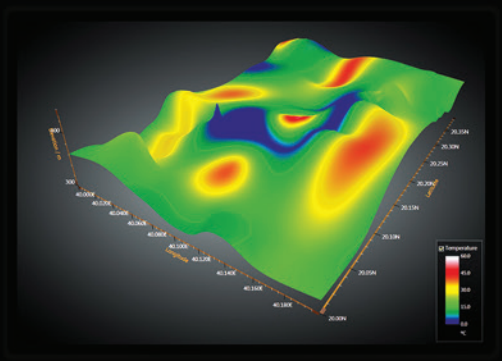
Arction

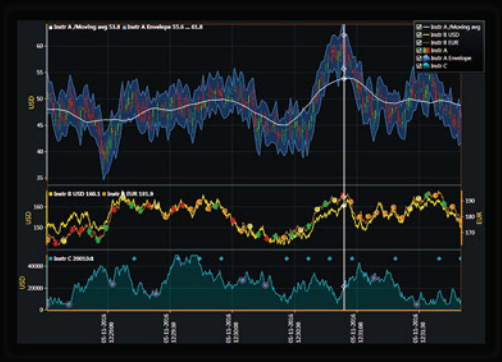
[WPF]
[Windows Forms]
[Free Gauges]
[Data Visualization]
[Volume Rendering]
[3D / 2D Charts] [Maps]

LightningChart®

The fastest and most advanced charting components







Create eye-catching and powerful charting applications for engineering, science and trading

- DirectX GPU-accelerated
- Optimized for real-time monitoring
- Supports gigantic datasets
- Full mouse-interaction
- Outstanding technical support
- Hundreds of code examples

NEW
- Now with Volume Rendering extension
- Flexible licensing options

Get free trial at
LightningChart.com/ms

Use Modern C++ to Access the Windows Registry

Giovanni Dicanio

The Windows OS exposes a series of C-interface APIs to give developers access to the registry. Some of these APIs are fairly low level and require programmers to pay attention to many details. Starting with Windows Vista, a kind of higher-level API was added to the mix: the RegGetValue function (bit.ly/2jXtfpJ). Before this API was introduced, to read a value from the registry, you first had to open the desired registry key containing the value calling RegOpenKeyEx. Then, you had to call the RegQueryValueEx API, dealing with many complex details. For example, if you read a string value with RegQueryValueEx, the returned string is *not* guaranteed to be properly NUL-terminated, which can cause a series of dangerous security bugs in your code. To prevent that from happening, you have to pay attention to check if there's a NUL terminator in the returned string; if there isn't any, you have to add it. Moreover, you have to make sure you properly close the open key, calling RegCloseKey.

Of course, opening the registry key could fail, so you have to add code to handle that, as well. The RegGetValue API simplifies this workflow, as it automatically opens the desired registry key, closes it after use, and it properly NUL-terminates strings before returning them to the caller. Despite this simplification, the RegGetValue function is still a low-level C-interface function; moreover, the fact that it can handle several different types of registry values (from DWORDs to strings to binary data), makes its interface complicated for you to program.

Thankfully, you can use modern C++ to properly build higher-level abstractions around this RegGetValue Win32 API, offering a simplified interface to read values of different types from the registry.

This article discusses:

- Wrapping some Win32 registry C API functions in modern C++ code
- Wrapping Win32 return codes in C++ exception classes
- Interfacing the STL std::wstring class with Win32 C APIs that operate with raw C character pointers and raw C buffers
- Wrapping the Win32 C RegGetValue API in higher-level C++ code to read values from the registry

Technologies discussed:

C++, Win32, STL, Registry, Exceptions

Code download available at:

msdn.com/magazine/0517magcode

Representing Errors Using Exceptions

The RegGetValue API is a C-interface API and as such it signals error conditions to the caller using return codes. In particular, this function returns a value of type LONG: ERROR_SUCCESS (that is, zero) in the case of success and a different value in the case of errors. For example, if the output buffer provided by the caller isn't large enough for the API to write its data, the function returns ERROR_MORE_DATA. For building a higher-level C++ interface around this C API, you can define a C++ exception class to represent errors. This class can be derived from the standard std::runtime_error class and you can embed the LONG error code returned by RegGetValue inside it:

```
class RegistryError
  : public std::runtime_error
{
public:
  ...
private:
  LONG m_errorCode;
};
```

In addition, other information pieces can be embedded in the exception object; for example, the HKEY and the name of the sub key. This is just a basic implementation.

You can add a constructor to create an instance of this exception class using an error message and the return code from the failed RegGetValue call:

```
RegistryError(const char* message, LONG errorCode)
  : std::runtime_error{message}
  , m_errorCode{errorCode}
{}
```

And the error code can be exposed to clients using a read-only accessor (getter):

```
LONG ErrorCode() const noexcept
{
  return m_errorCode;
}
```

Now that you've built this exception class, you can proceed to wrap the RegGetValue C API in a higher-level C++ interface that's easier to use and less bug-prone.

Reading a DWORD Value from the Registry

Let's start with a simple operation: using the RegGetValue API to read a DWORD value from the registry. The usage pattern in this case is pretty simple. But first, let's see what kind of interface can be defined in C++ for managing this case.

Here's the RegGetValue API prototype:

```
LONG WINAPI RegGetValue(
  _In_        HKEY    hkey,
  _In_opt_    LPCTSTR lpSubKey,
  _In_opt_    LPCTSTR lpValue,
  _In_opt_    DWORD   dwFlags,
  _Out_opt_   LPDWORD pdwType,
  _Out_opt_   PVOID   pvData,
  _Inout_opt_ LPDWORD pcbData
);
```

As you can see, this C-interface function takes highly generic data, like a void* output buffer (pvData) and an input/output buffer size parameter (pcbData). Moreover, there are C-style strings (lpSubKey and lpValue) that identify the registry key and the specific value name under that key. You can massage this C function prototype a bit, making it simpler for C++ callers.

First, as you're going to signal error conditions throwing C++ exceptions, the DWORD value read from the registry can just be returned by the C++ wrapper as a return value. This automatically eliminates the need of the raw void* output buffer parameter (pvData) and the associated size parameter (pcbData).

Moreover, as you're using C++, it's better to represent Unicode (UTF-16) strings using the std::wstring class instead of C-style raw pointers. So, you can define this much simpler C++ function to read a DWORD value from the registry:

```
DWORD RegGetDword(
  HKEY hKey,
  const std::wstring& subKey,
  const std::wstring& value
)
```

As you can see, there are no PVOID and LPDWORD parameters; the input strings are passed via const references to std::wstring objects and the value read from the registry is returned as a DWORD by this C++ function. This is definitely a much simpler and higher-level interface.

Now let's dive into the implementation. As mentioned, the invoke pattern for RegGetValue in this case is fairly simple. You just have to declare a DWORD variable that will store the value read from the registry:

```
DWORD data{};
```

Then you need another DWORD variable that represents the size (in bytes) of the output buffer written by RegGetValue. Note that the output buffer in this simple case is just the previous "data" variable, and its size is constantly the size of a DWORD:

```
DWORD dataSize = sizeof(data);
```

However, please note that you can't mark dataSize as "const" because it's both an input and an output parameter for RegGetValue.

Then you can invoke the RegGetValue API:

```
LONG retCode = ::RegGetValue(
  hKey,
  subKey.c_str(),
  value.c_str(),
  RRF_RT_REG_DWORD,
  nullptr,
  &data,
  &dataSize
);
```

The input wstring objects are converted to raw C-style string pointers using the wstring::c_str method. The RRF_RT_REG_DWORD flag restricts the type of the registry value to DWORD. If the registry value you're attempting to read is of a different type, the RegGetValue function call safely fails.

The last two parameters represent the address of the output buffer (in this case, the address of the data variable) and the address of a variable that stores the size of the output buffer. In fact, on return, RegGetValue reports the size of the data written to the output buffer. In this case of reading a simple DWORD, the size of data is always 4 bytes, that is, sizeof(DWORD). However, this size parameter is more important for variable-size values such as strings; I'll discuss that later in this article.

After invoking the RegGetValue function, you can check the return code and throw an exception in case of error:

```
if (retCode != ERROR_SUCCESS)
{
  throw RegistryError{"Cannot read DWORD from registry.", retCode};
}
```

Note that the error code (retCode) returned by RegGetValue is embedded in the exception object and can be later retrieved by the code that will process the exception.

Or, on success, the DWORD data variable can just be returned to the caller:

```
return data;
```

That's it for the function implementation.

The caller can simply invoke this C++ wrapper function with code like this:

```
DWORD data = RegGetDword(HKEY_CURRENT_USER, subkey, L"MyDwordValue");
```

Note how simple this code is when compared to the original RegGetValue C API call. You just pass a handle to an open registry key (in this example, the HKEY_CURRENT_USER predefined key), a string containing the sub key, and the value name. On success, the DWORD value is returned to the caller. On the other hand, on error, a custom exception of type RegistryError is thrown. This kind of code is higher level and much simpler than invoking RegGetValue. In fact, the complexity of RegGetValue has been hidden inside this custom RegGetDword C++ wrapper function.

You can use the same pattern to read a QWORD (64-bit data) value from the registry; in this case, you just have to substitute the DWORD type for the registry value with the 64-bit ULONGLONG.

Reading a String Value from the Registry

Reading a DWORD value from the registry is fairly simple: just one call to the RegGetValue Win32 API is sufficient. That's mainly because a DWORD value is of fixed size—four bytes—the size of a DWORD. On the other hand, reading strings from the registry introduces another layer of complexity because strings are *variable-size* data. The idea in this case is to call the RegGetValue API twice: In the first call, you request this API to return the desired size for the

output string buffer. Next, you dynamically allocate a proper size buffer. Finally, you make a second call to RegGetValue to actually write the string data into the previously allocated buffer. (This pattern was discussed in detail in my previous article, "Using STL Strings at Win32 API Boundaries," at msdn.com/magazine/mt238407).

First, take a look at the prototype of the C++ higher-level wrapper function:

```
std::wstring RegGetString(
  HKEY hKey,
  const std::wstring& subKey,
  const std::wstring& value
)
```

As in the DWORD case, this is much simplified with respect to the original complex RegGetValue C API prototype. The string value is returned from the function as a std::wstring instance. Instead, in case of errors, an exception is thrown. The sub-key name and the value name are passed as input wstring const reference parameters, as well.

Now I'll discuss the implementation code.

As I wrote, the idea is to first call the RegGetValue API to get the size of the output buffer to store the string value:

```
DWORD dataSize{};
LONG retCode = ::RegGetValue(
  hKey,
  subKey.c_str(),
  value.c_str(),
  RRF_RT_REG_SZ,
  nullptr,
  nullptr,
  &dataSize
);
```

You can see a call syntax similar to the previous DWORD value case. The wstring objects are converted to C-style string pointers invoking the wstring::c_str method. The RRF_RT_REG_SZ flag in this case restricts the valid registry type to the string type (REG_SZ). On success, the RegGetValue API will write the desired output buffer size (expressed in bytes) in the dataSize variable.

On failure, you have to throw an exception of the custom RegistryError class:

```
if (retCode != ERROR_SUCCESS)
{
  throw RegistryError{"Cannot read string from registry", retCode};
}
```

Now that you know the desired output buffer size, you can allocate a wstring object of the required size for the output string:

```
std::wstring data;
data.resize(dataSize / sizeof(wchar_t));
```

Note that the dataSize value returned by RegGetValue is expressed in bytes, but the wstring::resize method expects a size expressed in wchar_t count. So, you have to scale from bytes to wchar_t, dividing the former byte size value by sizeof(wchar_t).

Now that you have a string with enough room allocated, you can pass a pointer to its internal buffer to the RegGetValue API, which this time will write the actual string's data into the provided buffer:

```
retCode = ::RegGetValue(
  hKey,
  subKey.c_str(),
  value.c_str(),
  RRF_RT_REG_SZ,
  nullptr,
  &data[0],
  &dataSize
);
```

The &data[0] is the address of the wstring internal buffer that will be written by the RegGetValue API.

As usual, it's important to verify the result of the API call and throw an exception in case of error:

```
if (retCode != ERROR_SUCCESS)
{
  throw RegistryError{"Cannot read string from registry", retCode};
}
```

Note that on success, RegGetValue writes the actual result string size (in bytes) in the dataSize variable. You must resize the wstring object according to this size. As dataSize is expressed in bytes, it's better to convert it to the corresponding wchar_t count when dealing with wstrings:

```
DWORD stringLengthInWchars = dataSize / sizeof(wchar_t);
```

Moreover, dataSize includes the terminating NUL character for the output string. However, wstring objects are already NUL-terminated, so you must pay attention to avoid a spurious and bogus double-NUL termination for the read string. You have to chop off the NUL-terminator written by RegGetValue:

```
stringLengthInWchars--; // Exclude the NUL written by the Win32 API
data.resize(stringLengthInWchars);
```

Note that the RegGetValue API guarantees a NUL-terminated string on success, even if the original string stored in the registry wasn't NUL-terminated. This is a much safer behavior than the older RegQueryValueEx API, which didn't guarantee NUL-termination for the returned strings. So, the caller had to write additional code to properly take that case into account, increasing the overall code complexity and bug surface area.

Now that the data variable contains the string value read from the registry, you can return it to the caller on function exit:

```
return data;
```

Once you have this convenient RegGetString C++ wrapper around the RegGetValue low-level C API, you can invoke it like this:

```
wstring s = RegGetString(HKEY_CURRENT_USER, subkey, L"MyStringValue");
```

As in the DWORD case, you've raised the level of abstraction from the RegGetValue Win32 API, providing an easy-to-use and hard-to-misuse C++ wrapper function to read a string value from the registry. All the details and complexity of dealing with the RegGetValue API are safely hidden inside the body of this custom RegGetString C++ function.

Reading Multi-String Values from the Registry

Another type of registry value is the so-called "multi-string": Basically, this is a set of double-NUL-terminated strings packed in a single registry value. This double-NUL-terminated string data structure consists of a series of C-style NUL-terminated strings that occupy adjacent memory locations. The end of the sequence is marked by an additional NUL terminator, so the whole structure is terminated by two NULs. For more details on this data structure, see the blog post, "What Is the Format of a Double-Null-Terminated String with No Strings?" (bit.ly/2jCqg2u).

The usage pattern of the RegGetValue Win32 API in this case is very similar to the previous case of single strings. That is, first the RegGetValue API is invoked to get the size of the whole destination buffer containing the desired data (in this case, the whole sequence of adjacent strings terminated with a double-NUL). Then, a buffer of such size is dynamically allocated. And, finally, the RegGetValue function is called for the second time, passing the address of the previously allocated buffer so the API can write the actual multi-string data into that buffer.

In this case, you have to pay attention to the data structure storing the double-NUL-terminated string. In fact, while a std::wstring can properly contain embedded NULs, it could be *potentially* used

TECHMENTOR

IN-DEPTH TRAINING FOR IT PROS

AUGUST 7 - 11, 2017

MICROSOFT HEADQUARTERS

REDMOND, WA

PLUG IN TO NEW KNOWLEDGE @ THE SOURCE

WHAT SETS TECHMENTOR APART?

+ Immediately usable IT education
+ Training you need today, while preparing you for tomorrow
+ Zero marketing-speak, a strong emphasis on doing more with the technology you already own, and solid coverage of what's just around the corner
+ Intimate setting, where your voice is heard, making it a viable alternative to huge, first-party conferences
+ Experience life @ Microsoft Headquarters for a full week

YOU OWE IT TO YOURSELF, YOUR COMPANY AND YOUR CAREER TO BE AT TECHMENTOR REDMOND 2017!

HOT TRAINING TOPICS INCLUDE:

+ Windows Server + Hyper-V + Windows PowerShell + DSC + DevOps + Azure + Security + And More! +

++

REGISTER NOW

SAVE $400
USE PROMO CODE TMMAY1

[TECHMENTOREVENTS.COM/REDMOND]

EVENT SPONSOR: Microsoft

SUPPORTED BY: Redmond · Redmond Channel Partner · Virtualization Review

GOLD SPONSOR: GOVERLAN

PRODUCED BY: 1105 MEDIA

to store a double-NUL-terminated string structure, but I prefer to raise the level of abstraction and parse the double-NUL-terminated string into a higher-level, more convenient vector<wstring>.

So, the prototype of your C++ wrapper function to read multi-string values from the registry can look like this:

```
std::vector<std::wstring> RegGetMultiString(
  HKEY hKey,
  const std::wstring& subKey,
  const std::wstring& value
)
```

On success, the multi-string will be returned to the caller as a nice vector<wstring>. On the other hand, on error, an exception in the usual RegistryError form will be thrown.

Inside the body of your C++ wrapper function, first you invoke the RegGetValue API to get the size of the desired output buffer to store the multi-string:

```
DWORD dataSize{};
LONG retCode = ::RegGetValue(
  hKey,
  subKey.c_str(),
  value.c_str(),
  RRF_RT_REG_MULTI_SZ,
  nullptr,
  nullptr,
  &dataSize
);
```

Note the use of the RRF_RT_REG_MULTI_SZ flag this time to specify the multi-string registry value type.

As usual, in case of error, an exception is thrown, embedding the error code in the RegistryError object:

```
if (retCode != ERROR_SUCCESS)
{
  throw RegistryError{"Cannot read multi-string from registry", retCode};
}
```

On success, you allocate a buffer of proper size, to store the whole multi-string:

```
std::vector<wchar_t> data;
data.resize(dataSize / sizeof(wchar_t));
```

I consider a vector<wchar_t> much clearer than a wstring to represent the multi-string raw buffer. Note also that the size value returned by the RegGetValue API is expressed in bytes, so it must be properly converted to a wchar_t count before passing it to the vector::resize method.

Then, the RegGetValue API can be invoked for the second time, to write the actual multi-string data into the buffer previously allocated:

```
retCode = ::RegGetValue(
  hKey,
  subKey.c_str(),
  value.c_str(),
  RRF_RT_REG_MULTI_SZ,
  nullptr,
  &data[0],
  &dataSize
);
```

The &data[0] argument points to the beginning of the output buffer.

Again, you have to check the API return code and signal errors throwing a C++ exception:

```
if (retCode != ERROR_SUCCESS)
{
  throw RegistryError{"Cannot read multi-string"
    from registry", retCode};
}
```

It's also good to properly resize the data buffer with the dataSize value returned as an output parameter by the RegGetValue API:

```
data.resize( dataSize / sizeof(wchar_t) );
```

At this point, the data variable (which is a vector<wchar_t>) stores the double-NUL-terminated string sequence. The last step is to parse this data structure and convert it into a higher-level, more convenient vector<wstring>:

```
// Parse the double-NUL-terminated string into a vector<wstring>
std::vector<std::wstring> result;
const wchar_t* currStringPtr = &data[0];
while (*currStringPtr != L'\0')
{
  // Current string is NUL-terminated, so get its length with wcslen
  const size_t currStringLength = wcslen(currStringPtr);

  // Add current string to result vector
  result.push_back(std::wstring{ currStringPtr, currStringLength });
  // Move to the next string
  currStringPtr += currStringLength + 1;
}
```

Finally, the result vector<wstring> object can be returned to the caller:

```
return result;
```

This RegGetMultiString C++ wrapper can simply be invoked, like this:

```
vector<wstring> multiString = RegGetMultiString(
  HKEY_CURRENT_USER,
  subkey,
  L"MyMultiSz"
);
```

Again, all the complexity of the Win32 RegGetValue API has been hidden behind a high-level convenient C++ interface.

Enumerating Values Under a Registry Key

Another common Windows registry operation is the enumeration of the values under a given registry key. Windows provides the RegEnumValue API (bit.ly/2jB4kaV) for this purpose. Here, I'll show how to use this API to get a list of the names and types of the values located under a given registry key, wrapping the enumeration process in a convenient higher-level C++ function. Your custom C++ function can take as input a valid HKEY handle associated to the key you want to enumerate. On success, this custom C++ wrapper function will return a vector of pairs: The first item in the pair will be a wstring containing the value name and the second item a DWORD representing the value type. So, the prototype of this C++ wrapper function will look like this:

```
std::vector<std::pair<std::wstring, DWORD>> RegEnumValues(HKEY hKey)
```

Now I'll discuss the details of the enumeration process. The idea is to first call the RegQueryInfoKey (bit.ly/2jraw2H) API to get some useful pre-enumeration information, like the total value count and the maximum length of value names under the given registry key, as shown in **Figure 1**.

Figure 1 **Invoking the RegQueryInfoKey API**

```
DWORD valueCount{};
DWORD maxValueNameLen{};
LONG retCode = ::RegQueryInfoKey(
  hKey,
  nullptr,    // No user-defined class
  nullptr,    // No user-defined class size
  nullptr,    // Reserved
  nullptr,    // No subkey count
  nullptr,    // No subkey max length
  nullptr,    // No subkey class length
  &valueCount,
  &maxValueNameLen,
  nullptr,    // No max value length
  nullptr,    // No security descriptor
  nullptr     // No last write time
);
```

Note that I passed nullptr for the pieces of information in which I'm not interested. Of course, you have to check the return value and throw an exception if something went wrong when calling the aforementioned API:

```
if (retCode != ERROR_SUCCESS)
{
  throw RegistryError{"Cannot query key info from"
    the registry", retCode};
}
```

According to the Windows Dev Center RegQueryInfoKey function page (bit.ly/2lctUDt),

INFRAGISTICS

Write Fast, Run Fast

New Release

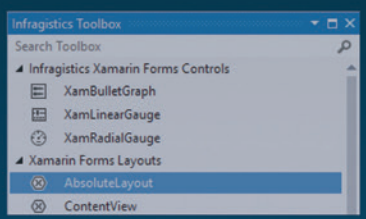
Infragistics Ultimate UI for Xamarin

Lightning-fast controls with a RAD WYSIWYG design-time experience

Get started today with a free trial, reference apps, tutorials, and eBooks at **Infragistics.com/Xamarin**

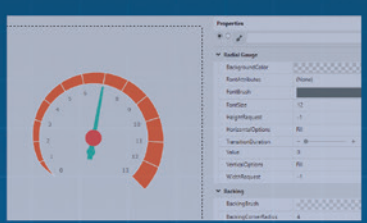
AppMap

Visually map out the entire flow of your app. Automatically generate all the Views, ViewModels, and Navigation code.



Xamarin.Forms Toolbox

Use the world's first NuGet-powered Toolbox to design your app views by dragging & dropping widgets directly from the Toolbox onto the XAML editor – no previous XAML knowledge required.



Control Configurators

Visually configure rich controls like data grids and charts right in the XAML editor and shave hours off your development time.

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

the size returned for the maximum length of value names (stored in the maxValueNameLen variable in the previous code) does *not* include the terminating NUL; so, let's adjust this value, adding one to take the terminating NUL into account when you allocate a buffer for reading value names:

```
maxValueNameLen++;
```

Then you can allocate a buffer of proper size to read the value names at each enumeration step; an efficient low-overhead std::unique_ptr<wchar_t[]> can be used for that purpose:

```
auto nameBuffer = std::make_unique<wchar_t[]>(maxValueNameLen);
```

The result of the enumeration, in the form of pairs of value name and value type, can be stored in a std::vector:

```
std::vector<std::pair<std::wstring, DWORD>> values;
```

You'll progressively add content to this vector during the enumeration process and then "values" will be returned to the caller when the enumeration is complete.

Then you can use a for loop, calling the RegEnumValue API repeatedly and enumerating a new value at each iteration step:

```
for (DWORD index = 0; index < valueCount; index++)
{
  // Call RegEnumValue to get data of current value ...
}
```

Note that you got valueCount from the initial pre-enumeration RegQueryInfoKey call.

Inside the body of the for loop, the RegEnumValue API can be called to get the desired information for the current value. In this context, you're interested in the value's name and value's type. The value's name will be read in the nameBuffer previously allocated; the value's type will be stored in a simple DWORD. So, inside the body of the for loop, you can write code like this:

```
DWORD valueNameLen = maxValueNameLen;
DWORD valueType{};
retCode = ::RegEnumValue(
  hKey,
  index,
  nameBuffer.get(),
  &valueNameLen,
  nullptr,    // Reserved
  &valueType,
  nullptr,    // Not interested in data
  nullptr     // Not interested in data size
```

As usual, it's good practice to check the API return value and throw an exception on error:

```
if (retCode != ERROR_SUCCESS)
{
  throw RegistryError{"Cannot get value info from the registry", retCode};
}
```

On success, the RegEnumValue API will write the value's name in the provided nameBuffer, and the value's type in the valueType variable. So, you can build a pair<wstring, DWORD> with these two pieces of information and add this information pair to the enumeration result vector:

```
values.push_back(std::make_pair(
  std::wstring{ nameBuffer.get(), valueNameLen },
  valueType
));
```

After the for loop, the result "values" vector can be returned to the caller:

```
return values;
```

The caller can then enumerate all the values under a registry key by just calling the C++ wrapper function like this:

```
auto values = RegEnumValues(hKey);

// For each value
for (const auto& v : values)
{
  // Process v.first (value's name) and v.second (value's type)
  // ...
}
```

A similar coding pattern can be used to enumerate the sub-keys under a given registry key; in this case, the Win32 RegEnumKeyEx (bit.ly/2k3VEX8) API must be used instead of the previously discussed RegEnumValue. The code of such sub-key enumeration function is provided in the download associated with this article.

A Safe Resource Manager for Raw HKEY Handles

Registry keys represented by the raw HKEY Win32 handle type can be safely and conveniently wrapped in a C++ resource manager class. The class destructor will properly call the RegCloseKey API on the wrapped raw handle to automatically close the handle. Moreover, move semantics operations like a move constructor and a move assignment operator can be defined to efficiently transfer ownership of the wrapped handle between different instances of the C++ resource manager class. For efficiency, all the class methods that don't throw exceptions are marked as noexcept, letting the C++ compiler emit more optimized code. This convenient key resource manager C++ class, named RegKey, is implemented in the Registry.hpp file accompanying this article. In this reusable header-only file, you'll also find the implementations of a couple of helper functions: RegOpenKey and RegCreateKey, that respectively wrap the Win32 APIs RegOpenKeyEx and RegCreateKeyEx, returning an HKEY handle safely wrapped in the aforementioned C++ resource manager class. In case of errors, those C++ functions throw a RegistryError exception, wrapping the error code returned by the raw C-interface Win32 APIs.

Wrapping Up

The RegGetValue Win32 API provides a relatively higher-level interface for reading values from the Windows registry when compared to lower-level APIs such as RegQueryValueEx. RegGetValue also offers a safer interface that, for example, guarantees the returned strings are properly NUL-terminated. Nonetheless, RegGetValue is still a C-interface low-level API that requires the programmer to pay attention to many details, and programming against it can lead to bug-prone complex code. This article showed how a convenient, easy-to-use and hard-to-misuse modern C++ interface can be built to hide the complexities of the RegGetValue API, while simplifying access to the Windows registry. Moreover, the RegEnumValue API was wrapped in a convenient higher-level C++ function to enumerate all the values under a given registry key. The source code containing the implementation of the functions and classes discussed in this article can be found in a reusable header-only form (in the Registry.hpp file) in the article's download. ■

Giovanni Dicanio *is a computer programmer specializing in C++ and the Windows OS, a Pluralsight author (bit.ly/GioDPS) and a Visual C++ MVP. Besides programming and course authoring, he enjoys helping others on forums and communities devoted to C++. Reach him via e-mail at giovanni.dicanio@gmail.com. He also blogs on msmvps.com/gdicanio.*

Thanks to the following technical experts for reviewing this article: *David Cravey and Marc Gregoire*

Extreme Performance Linear Scalability

NCache™

Distributed Cache

- In-Memory App Data Caching
- ASP.NET Sessions & View State
- Runtime Data Sharing

NosDB

NoSQL Database

- Schema-Free JSON Documents
- Multiple Shards & Data Replication
- Powerful SQL, LINQ, ADO.NET

100% Native .NET

Open Source

sales@alachisoft.com

alachisoft.com

US: +1 (925) 236 3830

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

AUSTIN, TX
MAY 15-18, 2017
HYATT REGENCY

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

HOT TOPICS INCLUDE:

- Visual Studio / .NET
- JavaScript / HTML5
- Angular
- Native Mobile & Xamarin
- Software Practices
- Database and Analytics
- ASP.NET Core
- Web API
- ALM / DevOps
- Cloud Computing
- UWP
- Unit Testing

FRIDAY, MAY 19: POST-CON HANDS-ON LABS

Choose From:

- Angular
- ASP.NET Core MVC/ Entity Framework

NEW! Only $695!

SPACE IS LIMITED

Register by May 15 and Save $200*

Must use discount code AUEB01 *Savings Based on 3-Day and 4-Day Packages Only.

REGISTER NOW

EVENT PARTNER: Microsoft

GOLD SPONSORS: Progress, Syncfusion

SUPPORTED BY: Visual Studio, msdn magazine, Visual Studio Magazine

PRODUCED BY: 1105 MEDIA — YOUR GROWTH, OUR BUSINESS

AUSTIN AGENDA AT-A-GLANCE

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

START TIME	END TIME	Visual Studio Live! Pre-Conference Workshops: Monday, May 15, 2017 *(Separate entry fee required)*		
9:00 AM	6:00 PM	**M01** Workshop: Distributed Cross-Platform Application Architecture - *Jason Bock & Rockford Lhotka*	**M02** Workshop: Practical ASP.NET DevOps with VSTS or TFS - *Brian Randell*	**M03** Workshop: Building for the Internet of Things: Hardware, Sensors & the Cloud - *Nick Landry*
6:45 PM	9:00 PM	Dine-A-Round		

START TIME	END TIME	Visual Studio Live! Day 1: Tuesday, May 16, 2017			
8:00 AM	9:00 AM	**KEYNOTE:** Slow Cook Your Serverless Application – Building Modem Applications in a Serverless World - *Yochay Kiriaty, Principal Program Manager, Microsoft*			
9:15 AM	10:30 AM	**T01** Go Mobile With C#, Visual Studio, and Xamarin - *James Montemagno*	**T02** Angular 2 – The 75-Minute Crash Course - *Chris Klug*	**T03** What's New in Azure IaaS v2 - *Eric D. Boyd*	**T04** Tour of Visual Studio 2017 - *Jason Bock*
10:45 AM	12:00 PM	**T05** Building Connected and Disconnected Mobile Apps - *James Montemagno*	**T06** Enriching MVC Sites with Knockout JS - *Miguel Castro*	**T07** Bots are the New Apps: Building Bots with ASP.NET WebAPI & Language Understanding - *Nick Landry*	**T08** Getting to the Core of .NET Core - *Adam Tuliper*
12:00 PM	1:30 PM	Lunch - Visit Exhibitors			
1:30 PM	2:45 PM	**T09** Lessons Learned from Real World Xamarin.Forms Projects - *Nick Landry*	**T10** Explore Web Development with Microsoft ASP.NET Core 1.0 - *Mark Rosenberg*	**T11** To Be Announced	**T12** Tactical DevOps with VSTS - *Brian Randell*
3:00 PM	4:15 PM	**T13** Xamarin vs. Cordova - *Sahil Malik*	**T14** Assembling the Web - A Tour of WebAssembly - *Jason Bock*	**T15** Cloud Oriented Programming - *Vishwas Lele*	**T16** Exploring C# 7 New Features - *Adam Tuliper*
4:15 PM	5:30 PM	Welcome Reception			

START TIME	END TIME	Visual Studio Live! Day 2: Wednesday, May 17, 2017			
8:00 AM	9:15 AM	**W01** To Be Announced	**W02** Introduction to Writing Custom Angular (Not 2.0) Directives - *Miguel Castro*	**W03** Microservices with Azure Container Service & Service Fabric - *Vishwas Lele*	**W04** Database Continuous Integration - *Steve Jones*
9:30 AM	10:45 AM	**W05** Write Once, Run Everywhere - Cordova, Electron, and Angular2 - *Sahil Malik*	**W06** Integrating AngularJS & ASP.NET MVC - *Miguel Castro*	**W07** Go Serverless with Azure Functions - *Eric D. Boyd*	**W08** Continuous Integration and Deployement for Mobile Using Azure Services - *Kevin Ford*
11:00 AM	12:00 PM	**GENERAL SESSION:** .NET in 2017 - *Jeffrey T. Fritz, Senior Program Manager, Microsoft*			
12:00 PM	1:30 PM	Birds-of-a-Feather Lunch - Visit Exhibitors			
1:30 PM	2:45 PM	**W09** Building Cross-Platform Business Apps with CSLA. NET - *Rockford Lhotka*	**W10** Aurelia vs "just Angular" a.k.a "the framework formerly known as Angular 2" - *Chris Klug*	**W11** Using Cognitive Services in Business Applications - *Michael Washington*	**W12** .NET Deployment Strategies: The Good, The Bad, The Ugly - *Damian Brady*
3:00 PM	4:15 PM	**W13** Creating Great Looking Android Applications Using Material Design - *Kevin Ford*	**W14** Use Docker to Develop, Build, and Deploy Applications, A Primer - *Mark Rosenberg*	**W15** Distributed Architecture: Microservices and Messaging - *Rockford Lhotka*	**W16** PowerShell for Developers - *Brian Randell*
4:30 PM	5:45 PM	**W17** SOLID – The Five Commandments of Good Software - *Chris Klug*	**W18** JavaScript for the C# (and Java) Developer - *Philip Japikse*	**W19** Introduction to Azure Machine Learning Studio (for the non-Data Scientist) - *Michael Washington*	**W20** Brownfields DevOps in Practice - *Damian Brady*
7:00 PM	9:00 PM	Rollin' On the River Bat Cruise			

START TIME	END TIME	Visual Studio Live! Day 3: Thursday, May 18, 2017			
8:00 AM	9:15 AM	**TH01** A Developers Introduction to HoloLens - *Billy Hollis*	**TH02** Extend and Customize the Visual Studio Environment - *Walt Ritscher*	**TH03** Entity Framework Core for Enterprise Applications - *Benjamin Day*	**TH04** Agile Failures: Stories From The Trenches - *Philip Japikse*
9:30 AM	10:45 AM	**TH05** Unity for .NET Developers - The Time is Now! - *John Alexander*	**TH06** SASS and CSS for Developers - *Robert Boedigheimer*	**TH07** A Tour of SQL Server 2016 Security Features - *Steve Jones*	**TH08** Busy Developer's Guide to the Clouds - *Ted Neward*
11:00 AM	12:15 PM	**TH09** Take the Tests: Can You Evaluate Good and Bad Designs? - *Billy Hollis*	**TH10** Debugging Your Website with Fiddler and Chrome Developer Tools - *Robert Boedigheimer*	**TH11** Busy Developer's Guide to NoSQL - *Ted Neward*	**TH12** Top 10 Ways to Go from Good to Great Scrum Master - *Benjamin Day*
12:15 PM	1:30 PM	Lunch			
1:30 PM	2:45 PM	**TH13** Using Visual Studio to Scale your Enterprise - *Richard Hundhausen*	**TH14** Building Single Page Web Applications Using Aurelia.js and the MVVM Pattern - *Ben Hoelting*	**TH15** Azure Data Lake - *Michael Rys*	**TH16** Care and Feeding of Your Product Owner - *John Alexander*
3:00 PM	4:15 PM	**TH17** Windows Package Management with NuGet and Chocolatey - *Walt Ritscher*	**TH18** Tools for Modern Web Development - *Ben Hoelting*	**TH19** U-SQL Killer Scenarios: Performing Advanced Analytics and Big Cognition at Scale with U-SQL - *Michael Rys*	**TH20** Stop the Waste and Get Out of (Technical) Debt - *Richard Hundhausen*

START TIME	END TIME	Full Day Hands-On Labs: Friday, May 19, 2017 *(Separate entry fee required)*	
8:00 AM	5:00 PM	**HOL01** Full Day Hands-On Lab: Busy Developer's HOL on Angular - *Ted Neward*	**HOL02** Full Day Hands-On Lab: Develop an ASP.NET Core MVC/Entity Framework Core App in a Day - *Philip Japikse*

Speakers and sessions subject to change

vslive.com/austinmsdn

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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

TED NEWARD

How To Be MEAN: Angular CRUD

Welcome back, MEANers.

Last month I explored Angular components—modules, components, services and others—in preparation for a deeper dive into how to create some of those components (msdn.com/magazine/mt795191).

So let's start applying these component concepts to something more practical.

Command-Line Tooling

Before I get into the component concepts, there's just one side note I want to make. Some readers e-mailed me to ask if the Git project was the only way to get started with an Angular project; they expressed dubiousness at the idea that this was the only way to get started. As they well suspect, that's hardly the case. Angular makes available an npm package called angular-cli that can get the bare-bones structure in place. Using it is straightforward: install, run the generator, answer a few questions and go:

```
npm install -g @angular/cli
ng new my-app
cd my-app
ng serve
```

This will create a new application in the my-app subdirectory and ng serve will effectively do an npm start and kick off a long-running process to run the Web server locally and transpile files as they're edited.

> The starting point for a new component is to create it.

Whichever approach you use, you'll end up with a scaffolded "hello world" application; but keep in mind, by the time your project starts to see serious coding, much of that scaffolding will get modified/replaced. So, the actual path you use to get started won't make much difference in the long run. Use whichever approach feels more comfortable.

Components

When thinking about how to structure a visual application—be it a traditional GUI desktop, a server-based Web application or an SPA application—it helps to think about the application in terms of the identifiable "parts" that can come to mind. Most applications, for example, have a discernible header or footer, usually some kind of menubar or menu structure, and a more generalized content area. Then, within that, there are smaller breakdowns, depending on the goal of the application and the constituent domain entities. One such breakdown is the traditional "master-detail" UI approach in which a user first navigates to a large (usually searchable) overview list of domain entities, selects one and moves into a detailed view of that domain entity. Often, that detailed view can be flipped between read-only and editable states through some kind of button or switch, usually defaulting to a read-only view to start.

The key to all of this is to think in component terms: Where are the convenient boundaries? In the case of an application that wants to display the speakers giving talks and let users of the application offer their feedback, the components are fairly obvious. (As with most written examples, they're designed to be.) A few components will be trivial to write—and thus make good first steps—owing to the fact that they involve no user input or editable state, and a few will be a bit more complex, as they'll likely require both input and reaction to changes in state. The key across all components is a high degree of coherence: The component should basically be a tightly fitting "whole" that represents a "thing," whatever that "thing" might be.

I'll start by defining that "thing" as being a simple footer component that will display a copyright message containing the current year.

Copyright

Now, it's time to start working on the first component. Assuming you've got a cleanly scaffolded project (meaning you've thrown away the GreetingsComponent you did last time), you want to build out a new UI component to display a footer at the bottom of the application's main page. Ideally, this component should display "Copyright (c) (current year), (your company)" across the page, wherever it appears. The assumption is that the users of this component will put it at the bottom of the page, but it's entirely possible that some odd UI/UX quirk will require something to go below it (perhaps legal disclaimers?), so let's not make any strong assumptions about where it will appear on the page.

The starting point for a new component is to create it. While there's always the option of creating files and directories by hand, I like tools that will do the repetitive stuff for me. So I'll lean on the angular-cli ng tool again, and this time, ask it to generate a component for me:

```
$ ng generate component footer
installing component
  create src/app/footer/footer.component.css
  create src/app/footer/footer.component.html
  create src/app/footer/footer.component.spec.ts
  create src/app/footer/footer.component.ts
  update src/app/app.module.ts
```

Spreadsheets Made Easy.

Fastest Calculations

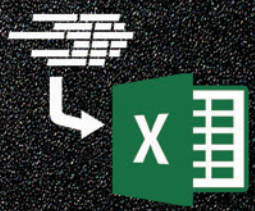
Evaluate complex Excel-based models and business rules with the fastest and most complete Excel-compatible calculation engine available.

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

Figure 1 **CopyrightComponent**

```
import { Component, Input, OnInit } from '@angular/core';

@Component({
  selector: 'footer',
  templateUrl: './footer.component.html',
  styleUrls: ['./footer.component.css']
})
export class FooterComponent implements OnInit {

  currentYear: string;
  @Input() owner: string;

  constructor() { }

  ngOnInit() {
    var d = new Date();
    this.currentYear = d.getFullYear().toString();
    console.log("Current year is ", this.currentYear);
  }

}
```

The generated files aren't anything magical, and you've seen them before: The footer directory contains the FooterComponent scaffolded code, and includes a TypeScript source file, a CSS and HTML file for the UI, and a .spec.ts file specifically for testing this component. It has also updated the application module file (and, sure enough, if you look in that file, it's been updated to include the footer files and import directive). If the server is still running and you have a browser window open to it, then there will be a brief flicker of "Loading …" and then the application will show up again. (Nothing new is displayed because the UI hasn't changed, but the flicker is the application being reloaded because of the modifications to the app.module.ts file.) Because it's easiest to imagine how you'll want to use this particular component, let's start with the HTML template in the footer.component.html file. And, in fact, let's start with how you want to use it.

Fundamentally, I'm thinking that the footer will want to display something along the lines of "Copyright (c) 2017, Neward & Associates, LLC," which suggests two variabilities in the message displayed: the current year and the copyright owner. One will be generated internally within the component (by creating a new Date object and extracting the year) and one will be passed in from the outside, like so (from app.component.html):

```
<h1>
  {{title}}
</h1>

<footer owner="Neward &amp; Associates, LLC" />
```

This suggests that the FooterComponent code needs an input variable (as I discussed in last month's column) and then also an internal currentYear variable that will be reflected in the HTML (footer.component.html):

```
<p>Copyright (c) {{currentYear}} {{owner}}</p>
```

So now, all that remains is to populate the right variables in the FooterComponent itself (footer.component.ts), as shown in **Figure 1**.

Note that the "selector" value in the @Component decorator defaulted to app-footer and I've changed it to footer so as to match the HTML usage I want; this is obviously an aesthetic argument in terms of what the tags should look like. (Some may prefer the app- prefix, but I find it a little redundant.) Whatever your preference, the selector value will define the syntax used in the HTML templates used by other components, so a consistent "style" will go a long way against confusion over time as more and more components are developed, debugged and deployed.

Testing, Testing

Speaking of debugging, one of the things scaffolded out is the .spec.ts file that defines a basic set of unit tests for the component, invoked as part of the test suite that gets executed when running either npm test or ng test. (The test rig is set up to automatically pick up all .spec.ts files under the app subdirectory, so it picks up FooterComponent tests the moment the file is introduced.) Without going into too much detail (yet) about the tests, **Figure 2** shows what FooterComponent tests look like.

To those readers who have spent some time writing unit tests in other frameworks, this won't seem too entirely indecipherable; to wit:

- describe establishes a unit test suite, which doubles as a scope/context in which all the tests defined therein will run. It provides a place, for example, to hang fields used as part of each test. (Never try to use these across tests; always assume—and ensure—that the fields are reset before and after each test.)
- beforeEach, as its name implies, is a block of code that runs before each test.

Figure 2 **FooterComponent Test**

```
import { async, ComponentFixture, TestBed } from '@angular/core/testing';
import { By } from '@angular/platform-browser';
import { DebugElement } from '@angular/core';

import { FooterComponent } from './footer.component';

describe('FooterComponent', () => {
  let component: FooterComponent;
  let fixture: ComponentFixture<FooterComponent>;
  let de: DebugElement;
  let el: HTMLElement;

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [ FooterComponent ]
    })
    .compileComponents();
  }));

  beforeEach(() => {
    fixture = TestBed.createComponent(FooterComponent);
    component = fixture.componentInstance;
    de = fixture.debugElement.query(By.css('p'));

    component.owner = "ACME, Inc";
    fixture.detectChanges();

    el = de.nativeElement;
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });

  it('should know the current year', () => {
    const currentYear = "" + new Date().getFullYear();
    expect(component.currentYear).toEqual(currentYear);
  });

  it('should know the owner', () => {
    expect(component.owner).toEqual("ACME, Inc");
  });

  it('should render message in a p tag', async(() => {
    expect(el.textContent).toContain("Copyright (c)");
    expect(el.textContent).toContain(new Date().getFullYear().toString());
    expect(el.textContent).toContain("ACME, Inc");
  }));
});
```

- it defines a test, consisting of a description and a block of code (which, not surprisingly, is the test in question) to run.
- expect is a fluent interface, defining a set of expectations that should hold true; if they don't, the test is considered to have failed.

When run (using either npm test or ng test), a browser window will open and indicate the passing nature—or not—of the code under test. If this process is left to run in the background, by the way, every time a file is changed after the transpiling is done (by the npm server or ng serve process), the tests will automatically run again, giving near-instant feedback on whether the code passes muster.

In this case, the two beforeEach blocks do some standard boilerplate setup, including passing in the owner data that would normally come through the template, and then testing to ensure that the owner data was successfully passed in, that currentYear was correctly calculated, and that the <p> tag in the template, when rendered, will contain the right parts of the copyright message.

And then, of course, the browser window that contains the whole application should also display the new message.

Wrapping Up

At a certain point during every would-be Angular developer's journey through the opening steps of Angular, there comes a point where the cognitive overhead seems so much higher than the benefits; "After all," they might muse, "all I really wanted to do was put a copyright message on the bottom of the page!" This is where Angular, like many environments, can be deceiving, in that it might seem to make simple things difficult. Truthfully, that's not entirely true—you could easily have just dropped the footer on the bottom of the AppComponent in raw HTML and static text, and then you'd have to change it next year, then again when the company gets acquired, and the year after that when the date changes again, and the year after that and so on.

The act of constructing a component is never as freely undertaken as the act of simply "dropping some HTML" onto the page. The pain will come in time, however, as the complexity of the application scales up and developers are forced to remember every page on which the footer appears; by encapsulating the footer into a component, the developer can just use it without having to worry about how the footer comes to decide what message will be displayed. And, just the same, if some later legal scenario requires a more comprehensive message, such as including the license model under which this page is offered to the world, you can change it in one—and only one—place, and that change will silently ripple through the rest of the system without requiring manual intervention.

There's still much more to explore. In the meantime, happy coding! ■

Ted Neward *is a Seattle-based polytechnology consultant, speaker and mentor, currently working as the director of Developer Relations at Smartsheet.com. He has written more than 100 articles, authored and coauthored a dozen books, and works all over the world. Reach him at ted@tedneward.com or read his blog at blogs.tedneward.com.*

Thanks to the following technical expert for reviewing this article: *Ward Bell*

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

ANAHEIM, CA
OCT 16-19, 2017
HYATT REGENCY
A Disneyland® Good Neighbor Hotel

California CODIN'

EVENT PARTNER
Microsoft

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!

Development Topics include:

- Visual Studio / .NET
- JavaScript / HTML5
- Angular
- Native Mobile & Xamarin
- Software Practices
- Database and Analytics
- ASP.NET Core
- Web API
- ALM / DevOps
- Cloud Computing
- UWP
- Unit Testing

Register Now and Save $300!*

Use promo code VSLAN2 *SAVINGS BASED ON 4-DAY PACKAGES ONLY.

REGISTER NOW

ROCK YOUR CODE TOUR 2017

ANAHEIM
OCT 16-19

"Good speakers; comfortable environment; I really liked the food and venue. Everything ran very smoothly!" *- Vickie Bertini, Roosevelt University*

"Content was up to date, plus no cookie-cutter sessions here! That made listening to lectures more enjoyable. The friendly atmosphere and networking was of great value, too." *- James McConnell, Anexinet*

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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

vslive.com/anaheimmsdn

FRANK LA VIGNE

Deep Dive into the Map Control

In my column last month, I demonstrated the basic features of the Bing Maps Control for Universal Windows Platform (UWP) apps and how easy it is to add a map to your UWP apps. This month I'll show you how to leverage some of the more advanced features that will make your apps really stand out and provide great experiences for your users. I'll also explore how to embed the rich imagery and 3D experience of Bing Maps directly into your apps.

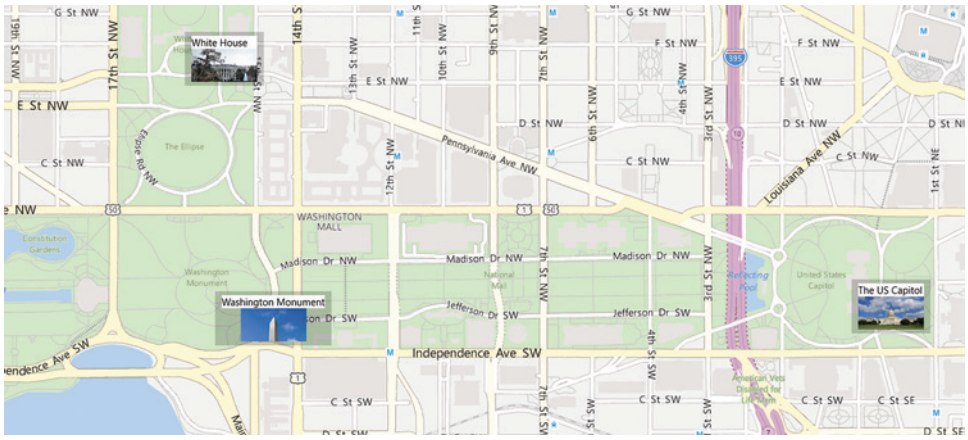


Figure 1 **Map with Customized Button Controls Placed at Specified Location**

Getting Started

For an in-depth look at how to add the Map control and obtain a MapServiceToken, please refer to my column last month (msdn.com/magazine/mt797655). For the purposes of this column, create a new blank UWP project in Visual Studio by choosing New Project from the File menu. Then expand to Installed Templates | Windows | Blank App (Universal Windows). Name the project DCTourismMap and then click OK. Immediately afterward, a dialog box will appear asking you which version of Windows the app should target. For this project, the default options will be fine, so you can simply click OK. In the MainPage.xaml file, add the following namespace declaration to the Page tag:

```
xmlns:maps="using:Windows.UI.Xaml.Controls.Maps"
```

Next, add the Map control to the page, by adding the following XAML in the Grid control on the page (be sure to insert the MapServiceToken value you received from the Bing Maps Portal at bingmapsportal.com):

```
<maps:MapControl x:Name="mapControl" MapServiceToken="{Insert Key Here}" >
</maps:MapControl>
```

You may now run the solution and you should see a Map control covering the entire screen of your app. With this control in place, you can now begin to take advantage of all that the Bing Maps control for UWP has to offer.

Custom Map Markers

In my last column, I wrote about adding markers to the map. The markers simply pinpoint a location on a map. However, the UWP Map control also lets developers place customized XAML controls onto a map and, along with XAML controls, comes enhanced interactivity. The goal is to create a map that showcases various landmarks in Washington, D.C. The markers on the map will each have an image of the location and will bring up a Web site about it when you click on it. In the end, the map should look something like **Figure 1**.

To accomplish this, the Map control will need a MapItemsControl, a DataTemplate, and a few other attributes added. First, modify the Map control XAML in the MainPage.xaml to look like what's shown in **Figure 2**.

Figure 2 **Modified Map Control XAML in MainPage.xaml**

```
<maps:MapControl x:Name="mapControl" Loaded="mapControl_Loaded"
  MapServiceToken="[Insert Key Here]">
  <maps:MapItemsControl x:Name="sitesMapItemsControl" >
    <maps:MapItemsControl.ItemTemplate>
      <DataTemplate>
        <Button Click="itemButton_Click"
          maps:MapControl.Location="{Binding Location}" >
          <StackPanel>
            <Border Background=
              "{ThemeResource ApplicationPageBackgroundThemeBrush}">
              <TextBlock Text="{Binding Name}"/>
            </Border>
            <Image Source="{Binding ImageUri}" Width="100" Height="50"
              Stretch="UniformToFill">
            </Image>
          </StackPanel>
        </Button>
      </DataTemplate>
    </maps:MapItemsControl.ItemTemplate>
  </maps:MapItemsControl>
</maps:MapControl>
```

Code download available at Bit.ly/MapControl.

Figure 3 **Event Handler to Center Map on Washington, D.C., and Create List to Populate Points of Interest**

```
private void mapControl_Loaded(object sender, RoutedEventArgs e)
{
  this.mapControl.Center = new Geopoint(
    new BasicGeoposition() { Latitude = 38.889906, Longitude = -77.028634 });
  this.mapControl.ZoomLevel = 16;
  sitesMapItemsControl.ItemsSource = LoadPointsOfInterest();
}

private List<POI> LoadPointsOfInterest()
{
  List<POI> pointsOfInterest = new List<POI>();

  pointsOfInterest.Add(new POI()
  {
    Name ="Washington Monument",
    ImageUri= new Uri(
      "https://upload.wikimedia.org/wikipedia/commons/e/ea/
      Washington_October_2016-6_%28cropped%29.jpg"),
    InformationUri = new Uri("https://www.nps.gov/wamo/index.htm"),
    Location = new Geopoint(
      new BasicGeoposition() { Latitude = 38.8895, Longitude = -77.0353 })
  });

pointsOfInterest.Add(new POI()
{
  Name = "White House",
  ImageUri = new Uri(
    "http://www.publicdomainpictures.net/pictures/20000/nahled/white-house.jpg"),
  InformationUri = new Uri("https://www.whitehouse.gov"),
  Location = new Geopoint(
    new BasicGeoposition() { Latitude = 38.897734, Longitude = -77.036535 })
});


pointsOfInterest.Add(new POI()
{
  Name = "The US Capitol",
  ImageUri = new Uri(
    "https://upload.wikimedia.org/wikipedia/commons/thumb/4/4f/US_
    Capitol_west_side.JPG/320px-US_Capitol_west_side.JPG"),
  InformationUri = new Uri("https://www.capitol.gov/"),
  Location = new Geopoint(
    new BasicGeoposition() { Latitude = 38.889892, Longitude = -77.009341 })
});

return pointsOfInterest;

}
```

Figure 4 **XAML to Create the Interface for the 3D Map App**

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
  <Grid.RowDefinitions>
    <RowDefinition Height="30*"/>
    <RowDefinition Height="80*"/>
    <RowDefinition Height="549*"/>
  </Grid.RowDefinitions>
  <StackPanel Orientation="Horizontal"  Grid.Row="0">
    <Button Name="btn3DView" Content="3D View" Click="btn3DView_Click" />
  </StackPanel>
  <StackPanel Orientation="Vertical" Grid.Row="1">
    <TextBlock FontWeight="Bold">Heading: </TextBlock>
    <TextBlock Text="{Binding ElementName=mapControl,Path=Heading,
      Mode=OneWay}"></TextBlock>
    <TextBlock FontWeight="Bold">Pitch: </TextBlock>
    <TextBlock Text="{Binding ElementName=mapControl,Path=ActualCamera.Pitch,
      Mode=OneWay}"></TextBlock>
  </StackPanel>
  <maps:MapControl x:Name="mapControl" Grid.Row="2"
    MapServiceToken="[Insert Token Here]">
  </maps:MapControl>
</Grid>
```

Figure 5 **Code to Switch to a 3D Map View Focused on Lower Manhattan**

```
private async void btn3DView_Click(object sender, RoutedEventArgs e)
{
  if (this.mapControl.Is3DSupported)
  {
    this.mapControl.Style = MapStyle.Aerial3DWithRoads;

    BasicGeoposition nycDowntownLatLong = new BasicGeoposition()
    {
      Latitude = 40.712929,
      Longitude = -74.018291
    };

    double radius = 550;
    double heading = 90;
    double pitch = 80;

    MapScene nycDowntownScene = MapScene.CreateFromLocationAndRadius(
      new Geopoint(nycDowntownLatLong), radius, heading, pitch );

    await mapControl.TrySetSceneAsync(nycDowntownScene);
  }
}
```

Read the XAML carefully and note that the MapItemsControl DataTemplate binds to an object with several properties. The next step is to create a model that contains those properties. In Solution Explorer, right-click on the project, click on Add, and then click on New Folder in the subsequent menu. Name the folder Models and hit enter. Right-click on the Models folder, click Add, and then choose New Item. In the subsequent dialog box, choose Class from the list of templates. Name the new class file POI.cs and click OK (POI stands for points of interest).

Modify the contents of the POI.cs file to resemble the following code:

```
using System;
using Windows.Devices.Geolocation;

namespace DCTourismMap.Model
{
  public class POI
  {
    public string Name { get; set; }
    public Geopoint Location { get; set; }
    public Uri ImageUri { get; set; }
    public Uri InformationUri { get; set; }
  }
}
```

Next, open the MainPage.xaml.cs file and add the event handler shown in **Figure 3** to center the map on Washington, D.C., and set a zoom level that focuses on the area around the National Mall. The LoadPointsOfInterest method creates and populates a List<POI>. The last line of the mapControl_Loaded method sets the ItemsSource of the MapItemsControl to this list.

Adding Interactivity

Now, add the following event handler code for the button control specified in the DataTemplate:

```
private void itemButton_Click(object sender, RoutedEventArgs e)
{

var buttonSender = sender as Button;
POI poi = buttonSender.DataContext as POI;
Launcher.LaunchUriAsync(poi.InformationUri);

}
```

The event handler retrieves the POI object associated with the control that triggered the event and hands off the InformationURI property to the LaunchURIAsync method of the Launcher class.

The Ultimate Education Destination

2017 Orlando

ROYAL PACIFIC RESORT AT UNIVERSAL ORLANDO
NOVEMBER 12-17

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

- 6 FULL Days of Training
- 5 Co-Located Conferences
- 1 Low Price
- Create Your Own Agenda from Hundreds of Sessions
- Expert Education and Training
- Knowledge Share and Networking

CONNECT WITH LIVE! 360

twitter.com/live360
@live360

facebook.com
Search "Live 360"

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

EVENT PARTNERS

Microsoft Magenic

PLATINUM SPONSOR

SMARTBEAR

SUPPORTED BY

Visual Studio msdn magazine Redmond Channel Partner

LIVE! 360

TECH EVENTS WITH PERSPECTIVE

NEW: HANDS-ON LABS

Join us for full-day, pre-conference hands-on labs Sunday, November 12.

Only $595 through August 11

5 GREAT CONFERENCES, 1 GREAT PRICE

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

Visual Studio Live!: Code in Paradise at VSLive!™, featuring unbiased and practical development training on the Microsoft Platform.

SQL Server LIVE!

TRAINING FOR DBAs AND IT PROS

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

TECHMENTOR

IN-DEPTH TRAINING FOR IT PROS

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

Office & SharePoint LIVE!

ON-PREMISE, CLOUD & CROSS-PLATFORM TRAINING

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

Modern Apps LIVE!

MOBILE, CROSS-DEVICE & CLOUD DEVELOPMENT

Modern Apps Live!: Presented in partnership with Magenic, this unique conference delivers how to architect, design and build a complete Modern App.

REGISTER NOW

REGISTER BY AUGUST 11 AND SAVE $500!*

Use promo code L360MAY2

*Savings based on 5-day packages only. See website for details.

PRODUCED BY

Redmond MAGAZINE | VIRTUALIZATION & Cloud Review | Visual Studio MAGAZINE | 1105 MEDIA

LIVE360EVENTS.COM

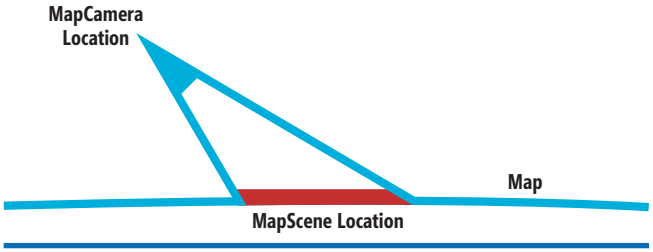


Figure 6 **Conceptual Diagram of a MapScene**

For URIs of Web sites, this will launch the default browser on the system and load the URI sent to it. The Launcher class resides in the Windows.System namespace. You have the option of adding the namespace manually with a using statement or letting Visual Studio handle that detail. The Launcher class can do much more than simply open a browser. You can read more about the Launcher class at bit.ly/2n4ZxOF.

Run the solution now and it should look like **Figure 1**. Clicking on any of the markers will bring up a Web site about the landmark.

Maps in Three Dimensions

One of the more appealing aspects of the Bing Maps UWP control is its ability to render 3D images of many locations around the world. To get started, create a new project using the steps detailed in the Getting Started section. This time, however, name the solution 3DMaps. Be sure to add the namespace declaration to the MainPage.xaml file to include the maps namespace. Insert the XAML shown in **Figure 4** in the Page node of MainPage.xaml to the Map control and a series of buttons to control the 3D view of the map.

Manipulating 3D Maps

Viewing a map in 3D is simple and I touched on that briefly in my column last month. However, all the sample app from that column did was switch to 3D view. Now, let's go over how to programmatically manipulate a 3D map. Add the event handler for the btn3Dview button, as shown in **Figure 5**.

Immediately, you'll notice the code to switch the Map control Style property to Aerial3DWithRoads. Beyond that, I want to be able to control specific details about the view. In this example, I want to show the skyline of Lower Manhattan. In order to do this, I need to "frame a shot" very much in the same way a photographer would by picking a location in 3D space to set up a camera and point in a certain position and specified angle. The diagram in **Figure 6** provides an overview.

To set up the desired picture, start with a latitude and a longitude to define the location. Then define variables for radius, heading and pitch to control the view of the camera. Next, create a MapScene using the MapScene.CreateFromLocationAndRadius method. This creates a scene centered on the specified latitude and longitude from a certain distance away in meters (radius). Heading refers to the direction in which the camera faces. This value can be anywhere from zero to 360, representing the compass points. For reference, a value of 90 represents East. A value of zero is North, 180 is South, and 270 is West. Pitch refers to the angle at which the camera is facing. Zero would be a top-down view and 90 would be perpendicular.

Figure 7 **Event Handlers for UI Buttons that Control the MapScene Camera**

```
private async void btnRotateCameraLeft_Click(object sender, RoutedEventArgs e)
{
  await mapControl.TryRotateAsync(-30);
}

private async void btnRotateCameraRight_Click(object sender, RoutedEventArgs e)
{
  await mapControl.TryRotateAsync(30);
}

private async void btnTiltCameraUp_Click(object sender, RoutedEventArgs e)
{
  await mapControl.TryTiltAsync(15);
}

private async void btnTiltCameraDown_Click(object sender, RoutedEventArgs e)
{
  await mapControl.TryTiltAsync(-15);
}
```

Once the MapScene object is created, it's applied to the Map control by using the TrySetSceneAsync method. Run the project now. Click on 3D View and your app should show a 3D view of lower Manhattan. Note that you can control the map via mouse and keyboard. If your device supports touch, you may also use touch gestures to control the heading, zoom level and pitch of the Map control.

Although having the mouse, keyboard and gesture control built right into the Map control is good, it would be nice to manipulate the 3D viewport of the map programmatically.

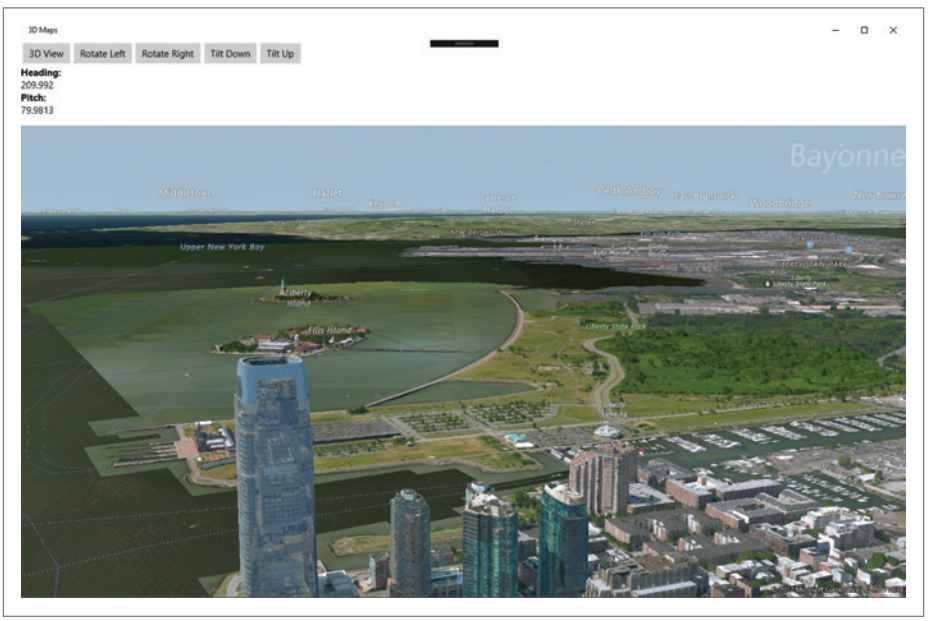


Figure 8 **The App After Clicking Rotate Right Several Times**

Close the app and return to the code in the MainPage.xaml file in Visual Studio. Add the following XAML into the StackPanel control after the btn3DView button:

```
<Button Name="btnRotateCameraLeft" Content="Rotate Left"
  Click="btnRotateCameraLeft_Click" />
<Button Name="btnRotateCameraRight" Content="Rotate Right"
  Click="btnRotateCameraRight_Click" />
<Button Name="btnTiltCameraDown" Content="Tilt Down"
  Click="btnTiltCameraDown_Click" />
<Button Name="btnTiltCameraUp" Content="Tilt Up"
  Click="btnTiltCameraUp_Click" />
```

This XAML will create four buttons to control the viewport of the map. Two are for controlling heading: one to turn right and the other to turn left. Two are for controlling the pitch: one to tilt up and the other to tilt down. Now, add the event handlers, as shown in **Figure 7**, to the MainPage.xaml.cs file.

The methods TryRotateAsync and TryTiltAsync both take a parameter of type double that represents the angle to rotate or tilt, respectively. Negative values rotate left and tilt down. Positive values rotate right and tilt up. Run the app now. Click on 3D View and try out the control you now have over the map through the buttons. Your app should look something like **Figure 8**.

StreetSide View

One of the other powerful features of Bing Maps is the StreetSide View, which lets users explore a location through a series of interactive panoramas. With the UWP Map control, your apps can now have this feature embedded into them.

If the app is already running, stop it and return to Visual Studio. Add the following XAML in the MainPage.xaml file to add a button for enabling StreetSide View:

```
<Button Name="btnStreetSide" Content="StreetSide" Click="btnStreetSide_Click" />
```

Now add the event handler in **Figure 9** to enable StreetSide View.

The first step is to check whether the app is running on a device that supports StreetSide View, as not every device is capable of supporting it. The next step is to create a GeoPoint from a latitiude/longitude coordinate. Now, pass that GeoPoint to the FindNearbyAsync method to find a nearby location that has StreetSide imagery available. If no imagery near that location exists, the method will return null.

All that's left to do is to create a StreetSide Experience by passing the StreetSide Panorama object to the StreetSide Experience constructor. To switch the Map control over to StreetSide View, set the Map control Custom Experience property to the newly created StreetSide Experience object.

Run the solution now and click on the StreetSide button. Click on Tilt Up and Tilt Down and the app should show the StreetSide View of the World Financial Center and Freedom Tower. Notice that the control responds to keyboard, mouse and touch. The control will even let you zoom in and explore the environment. Below the StreetSide View, there's an overview map that lets users explore the area even further. In the process of using the app, you might have noticed that the Turn Left and Turn Right buttons no longer work.

Close the app and return to Visual Studio and modify the event handlers for the turn buttons. In order to change the heading of the StreetSide view, the Heading property of the Map control needs to be changed directly. To detect whether StreetSide View is enabled, check to see if the Custom Experience property isn't null. Modify the event handlers for the Turn Left and Turn Right buttons so that they look like the code in **Figure 10**.

Run the solution again and click on the StreetSide button to enable StreetSide View. The Turn Left and Turn Right buttons now work. It's worth noting that, in 3D View, setting the Heading property of the Map control does, indeed, rotate the view of the camera. However, it does so without animating the points in between, making for a rough transition.

Figure 9 **Setup and Display the StreetSide View of Lower Manhattan**

```
if (mapControl.IsStreetSideSupported)
{
  BasicGeoposition nycDowntownLatLong = new BasicGeoposition()
  {
    Latitude = 40.712929,
    Longitude = -74.018291
  };

  Geopoint nycDowntownPoint = new Geopoint(nycDowntownLatLong);
  StreetSidePanorama panoramaNearDowntownNYC =
    await StreetSidePanorama.FindNearbyAsync(nycDowntownPoint);

  if (panoramaNearDowntownNYC != null)
  {
    var nycSSE = new StreetSideExperience(panoramaNearDowntownNYC);
    mapControl.CustomExperience = nycSSE;
  }
  }
}
```

Figure 10 **Updated Rotate Event Handlers to Support StreetSide View**

```
private async void btnRotateCameraLeft_Click(object sender, RoutedEventArgs e)
{
  await mapControl.TryRotateAsync(-30);
  if (mapControl.CustomExperience != null)
  {
    mapControl.Heading = mapControl.Heading - 30;
  }
}

private async void btnRotateCameraRight_Click(object sender, RoutedEventArgs e)
{
  await mapControl.TryRotateAsync(30);
  if (mapControl.CustomExperience != null)
  {
    mapControl.Heading = mapControl.Heading + 30;

  }
}
```

Wrapping Up

As mentioned in my column last month, maps are truly one of the more indispensable features of mobile devices. With the transition to digital maps, maps can become not just customizable, but interactive and immersive. In this column, you saw how the Map control included with the UWP provides these rich interactive features and 3D imagery. In most cases, access to the mapping services and imagery come at no cost to you, the developer. ■

Frank La Vigne *is chief evangelist at DataLeader.io, where he helps customers leverage data science in order to seek actionable insights. He blogs regularly at FranksWorld.com and has a YouTube channel called Frank's World TV (FranksWorld.TV).*

Thanks to the following Microsoft technical expert for reviewing this article: *Rachel Appel*

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

Development Topics Include:

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

REGISTER NOW

Register Today and Save $300!

Use promo code VSLCHI2

ROCK YOUR CODE TOUR 2017

"I loved that this conference isn't a sales pitch. It's actual developers giving the sessions - I felt the classes were really geared towards me!"
- Jon Plater, Kantar Retail

"The speakers were clearly experts. Nice breadth of topics. Great hotel; great location."
- Fred Kauber, Tranzact

vslive.com/chicagomsdn

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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

I'm Still Flying

I'm still flying. I just finished the best class I've ever taught in my life. I'm still waking up at night, pumping my fist and saying, "Yeah!" Come, see what my students and I accomplished.

I speak of the Advanced UX class that I taught this past January at Harvard Extension. It was a three-week intensive session, for which students flew in from as far away as Colombia. I decided that we'd all work together on a project, a good, important, challenging project. Where could I find such a thing?

I found a good one from Isaac Kohane, M.D., Ph.D., the chair of Biomedical Informatics at Harvard Medical School. I'd seen him speak at a Grand Rounds (a major hospital-wide presentation), decrying the innumeracy of the medical establishment, a significant impediment in this data-driven age. You might think that a guy with these credentials would be haughty and unapproachable, but exactly the opposite is true. I always found him friendly, open, interested and interesting. He goes by the nickname of Zak. And he liked the copy of "Why Software Sucks" that I gave him. When I pinged him last fall, he had a serious bee in his bonnet, and wondered if my class and I could extract it for him.

It was a three-week intensive session, for which students flew in from as far away as Colombia. I decided that we'd all work together on a project, a good, important, challenging project.

Zak's mother lives independently at age 92. She has congestive heart failure (CHF), a condition in which the heart's pumping action declines. Untreated CHF patients essentially drown in their own body fluids. Traditional maintenance treatment is a diuretic pill called Lasix, which causes patients to excrete their excess fluid, reducing the volume that their weakened hearts need to handle. Lasix is well understood, well tolerated and cheap ($10 for a 90-day supply at Walmart).

Patients encounter trouble when they start retaining more fluid than their regular Lasix dose can handle. Their hearts pump even less well, leading to more fluid retention, starting a vicious cycle. If you detect this situation quickly, you can reverse it with more oral Lasix. But if you don't catch it within a day or two, the oral route stops working. The patient then needs hospital admission to get dried out with a Lasix intravenous drip over four or five days. Naturally, the patients hate that, leaving their comfortable homes for needles and catheters, antibiotic-resistant infections, and roommates who snore. Or scream. Their loved ones hate it, too, having to drop their own lives and hang out in hospitals for a week.

In addition to being unpleasant, these hospital episodes cost about $25,000 each. About 6 million adults in the United States have CHF (see bit.ly/2msAnbh). The admission rate is stubbornly stuck around 1 million per year (see bit.ly/2mJ9sbU). Do the math and you'll see that the cost of this one problem is approaching 1 percent of the total U.S. health care budget.

Micro or macro, these CHF hospital admissions really suck. Could my class help?

After his mother needed two hospital admissions in 2015, Zak wondered if he could detect excess fluid retention by monitoring her weight, intervening in time to correct any imbalance orally. He gave her a Fitbit Aria bathroom scale (fitbit.com/aria), which uploads its measurements to a user's Web account. She stands on it every day, usually in the morning, and Zak checks the value sometime during the day. If she's up a pound or more, he tells her to take another Lasix, and checks the next day to make sure that she's back down. Following this regimen, he managed to avoid any admissions in 2016.

Now Zak had a proof of concept. Not-too-invasive monitoring on consumer-grade hardware and software could produce large benefits. How much better would a dedicated program be? He wanted to view the patient's weight without the other distracting items in the full Fitbit UI. He wanted to scroll back and forth to see the patient's weight history graphically. He wanted to record when and how much medication he prescribed. And, of course, he wanted it to run on his iPhone. Could we design one for him quickly?

Watch this space next month for the exciting conclusion. Be there! Aloha. ■

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

Looking for a powerful distributed cache?

Get the easiest, most powerful in-memory data grid designed to scale your .NET applications.

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

Go with the industry leader in distributed caching

Trusted by hundreds of enterprise customers for more than 12 years, ScaleOut's distributed caching technology delivers rock-solid performance, legendary ease of use, and world-class support. Unlike Redis, it offers a better migration path from AppFabric Caching. Here's why:

	ScaleOut	Redis
Source-code compatible AppFabric Caching APIs for seamless migration	✔	✖
Architected from the ground up for automatic scaling and high availability	✔	✖
Fully coherent data storage and access for mission-critical applications	✔	✖
Advanced .NET features: distributed LINQ query, C# MapReduce, and more	✔	✖

Replacing AppFabric Caching? Trouble switching to Redis?
Check out our AppFabric-compatible drop-in: **www.scaleoutsoftware.com/appfabric**

ScaleOut Software

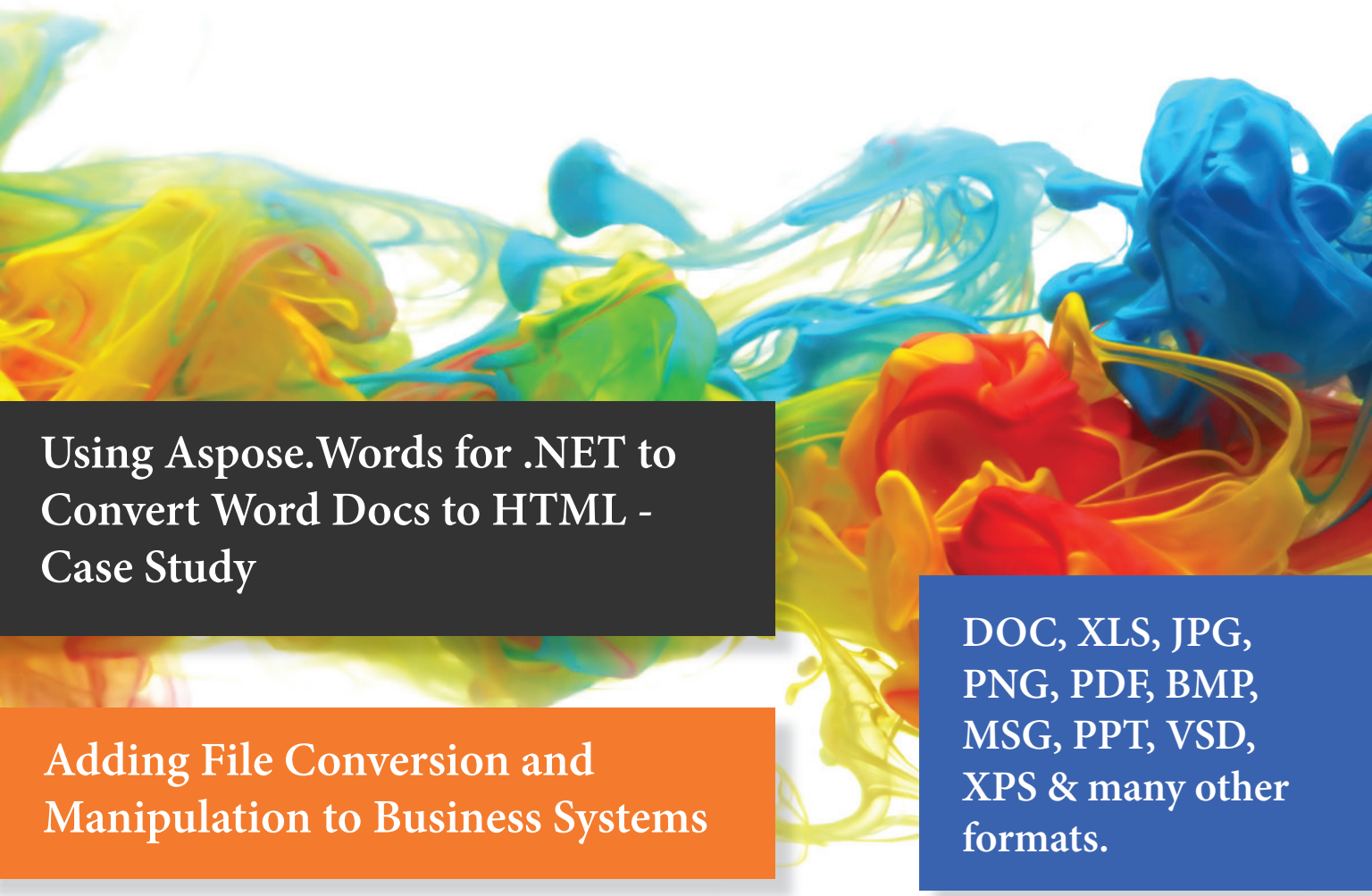
Please visit **scaleoutsoftware.com** or call us at **+1.503.643.3422** to learn how ScaleOut's technology can help accelerate your business.

ASPOSE

File Format APIs

Powerful File APIs that are easy and intuitive to use

Native APIs for .NET, Java & Cloud



Using Aspose.Words for .NET to Convert Word Docs to HTML - Case Study

Adding File Conversion and Manipulation to Business Systems

DOC, XLS, JPG, PNG, PDF, BMP, MSG, PPT, VSD, XPS & many other formats.

www.aspose.com

EU Sales: +44 141 628 8900

US Sales: +1 903 306 1676
sales@aspose.com

AU Sales: +61 2 8006 6987

Aspose.Total

Every Aspose API combined in one powerful suite.

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, TIFF, 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...

and many more!

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

ASPOSE
File Format APIs

Working with Files?

Try Aspose File APIs

Convert
Print
Create
Combine
Modify

files from your applications!

Over 15,000 Happy Customers

.NET Java Cloud

Get your FREE evaluation copy at www.aspose.com

Aspose.Cells

Work with spreadsheets and data without depending on Microsoft Excel

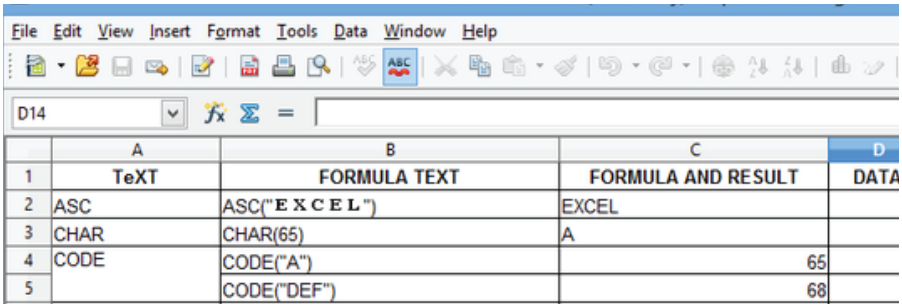
- Solution for spreadsheet creation, manipulation and conversion.
- Import and export data.

ASPOSE.CELLS IS A PROGRAMMING API that allows developers to create, manipulate and convert Microsoft Excel spreadsheet files from within their own applications. Its powerful features make it easy to convert worksheets and charts to graphics or save reports to PDF.

A flexible API for simple and complex spreadsheet programming.

Aspose.Cells speeds up working with Microsoft Excel files. The API is a flexible tool for simple tasks such as file conversion, as well as complex tasks like building models. Developers control page layout, formatting, charts and formulas. They can read and write spreadsheet files and save out to a wide variety of image and text file formats.

Fast, scalable, and reliable, Aspose.Cells saves time and effort compared to using Microsoft Office Automation.



	A	B	C	D
1	TeXT	FORMULA TEXT	FORMULA AND RESULT	DATA
2	ASC	ASC("EXCEL")	EXCEL	
3	CHAR	CHAR(65)	A	
4	CODE	CODE("A")	65	
5		CODE("DEF")	68	

Aspose.Cells lets developers work with data sources, formatting, even formulas.

Common Uses

- Building dynamic reports on the fly.
- Creating Excel dashboards with charts and pivot tables.
- Rendering and printing spreadsheets and graphics with high fidelity.
- Exporting data to, or importing from, Excel and other spreadsheets.
- Generating, manipulating and editing spreadsheets.
- Converting spreadsheets to images or other file formats.

Key Features

- A complete spreadsheet manipulation solution.
- Flexible data visualization and reporting.
- Powerful formula engine.
- Complete formatting control.

Supported File Formats

XLS, XLSX, XLSM, XMPS, XLTX, XLTM, ODS, XPS, SpreadsheetML, tab delim., CSV, TXT, PDF, HTML, and many image formats including SVG, TIFF, JPEG, PNG and GIF.

Format support varies across platforms.

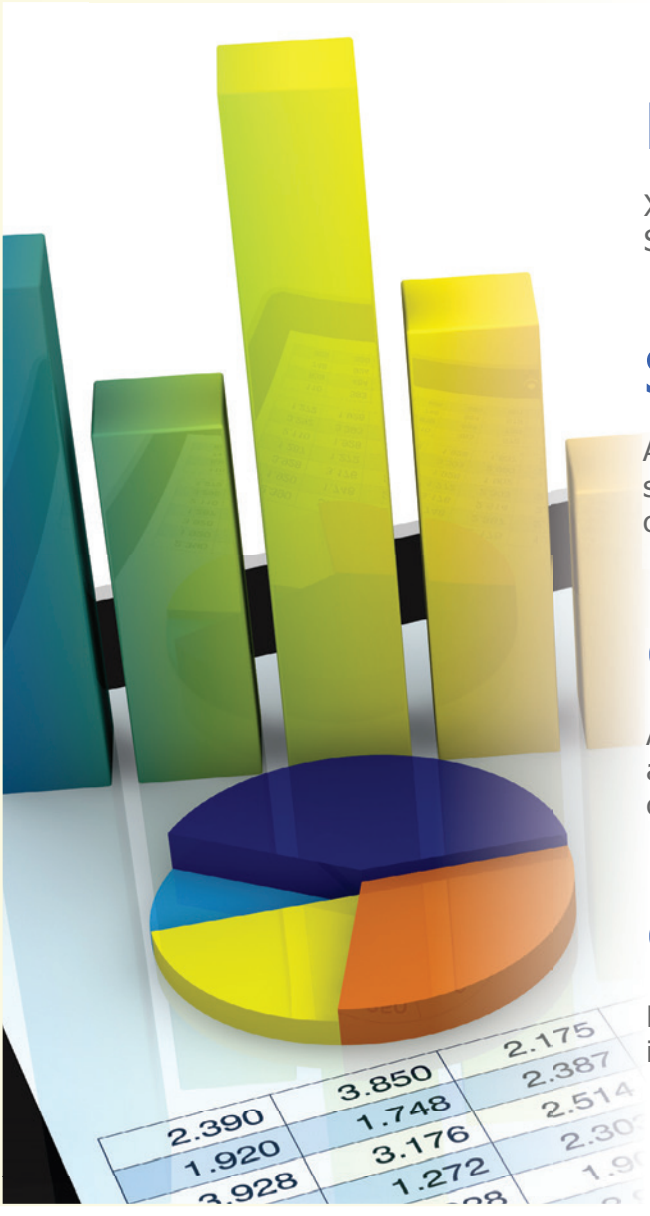
Platforms

Pricing Info					
	Standard	Enhanced		Standard	Enhanced
Developer Small Business	$999	$1498	Site Small Business	$4995	$7490
Developer OEM	$2997	$4494	Site OEM	$13986	$20972

The pricing info above is for .NET: prices for other platforms may differ. For the latest, contact sales.

Aspose.Cells for

.NET, Java, Cloud & more



File Formats

XLS, CSV, ODS, PDF, SVG, HTML, PNG, BMP, XPS, JPG SpreadsheetML and many others.

Spreadsheet Manipulation

Aspose.Cells lets you create, import, and export spreadsheets and also allows you to manipulate contents, cell formatting, and file protection.

Creating Charts

Aspose.Cells comes with complete support for charting and supports all standard chart types. Also, you can convert charts to images.

Graphics Capabilities

Easily convert worksheets to images as well as adding images to worksheets at runtime.

Get your FREE Trial at
http://www.aspose.com

100% Standalone

Aspose.Cells does not require Microsoft Office to be installed on the machine in order to work.

ASPOSE
File Format APIs

Aspose.Words

Program with word processing documents independently of Microsoft Word

- Solution for document creation, manipulation and conversion.
- Advanced mail merge functionality.

ASPOSE.WORDS IS AN ADVANCED PROGRAMMING API that lets developers perform a wide range of document processing tasks with their own applications. Aspose.Words makes it possible to generate, modify, convert, render and print documents without Microsoft Word. It provides sophisticated and flexible access to, and control over, Microsoft Word files.

Generate, modify, convert, render and print documents without Microsoft Word.

Aspose. Words is powerful, user-friendly and feature rich. It saves developers time and effort compared to using Microsoft Office Automation and makes gives them powerful document management tools.

Aspose.Words makes creating, changing and converting DOC and other word processing file formats fast and easy.

	Table			
	Column 1	**Column 2**	**Column 3**	**Column 4**
Row 1	Cell 1	Cell 2	Cell 3	Cell 4
Row 2	Cell 1	Cell 2	Cell 3	
Row 3	Cell 1	Cell 2		

Aspose.Words has sophisticated controls for formatting and managing tables and other content.

Common Uses

- Generating reports with complex mail merging; mail merging images.
- Populating tables and documents with data from a database.
- Inserting formatted text, paragraphs, tables and images into Microsoft Word documents.
- Adding barcodes to documents.
- Inserting diagrams and watermarks into Word documents.
- Formatting date and numeric fields.

Key Features

- A complete Microsoft Word document manipulation solution.
- Extensive mail merge features.
- Complete formatting control.
- High-fidelity conversion, rendering and printing.

Supported File Formats

DOC, DOCX, ODT, OOXML, XML, HTML, XHTML, MHTML, EPUB, PDF, XPS, RTF, and a number of image formats, including TIFF, JPEG, PNG and GIF.

Format support varies across platforms.

Platforms

Pricing Info					
	Standard	Enhanced		Standard	Enhanced
Developer Small Business	$999	$1498	Site Small Business	$4995	$7490
Developer OEM	$2997	$4494	Site OEM	$13986	$20972

The pricing info above is for .NET: prices for other platforms may differ. For the latest, contact sales.

Case Study: Aspose.Words for .NET

ProHire Staffing - Using Aspose.Words for .NET to convert Word Docs to HTML

PROHIRE IS THE WORKFORCE SOLUTIONS LEADER IN THE UNITED STATES AND SPECIALIZE IN THE RECRUITMENT OF SALES AND SALES MANGEMENT PROFESSIONALS. We were founded with the goal of becoming the premier provider of executive search and placement services to the Fortune 500 and Inc. 500 Companies.

Problem

ProHire uses Bullhorn ATS as its Application Tracking System to track the electronic handling of its recruitment needs. We wanted to integrate the Bullhorn API with our new website. Our goal was to convert MS Word Documents resumes into a clean and concise HTML format into our existing .Net Stack. The converted HTML resume version needed to look close to the original.

> "The transitions with Aspose. Words for .NET was very painless to do."

Looking for a Solution

We chose the ASPOSE.Words product because it easily integrated into our existing .Net stack, and provided a quality MS Word to HTML conversion. The product was easy to download, and with a few lines of code we were up and running. We found the primary difference between the Aspose. Words and other products was the obvious conversion quality from MS Word to HTML.

Finding a Solution

We had tested other products that converted Word to HTML. Every one we tested had some problem with the conversion. Some of them lost elements of the resume during the conversion. Most of them changed the format of the resume or changed the color of the text unexpectedly. This is unacceptable when you are sending a resume to a hiring manger. We were very satisfied with the results. We did not need any technical support because documentation was sufficient to us.

Implementation

Once we had the Aspose DLL our developer was able to implement Aspose.Words for .NET in a few hours. The transitions with Aspose. Words for .NET was very painless to do.

Outcome

We are very pleased with the success of our Aspose.Words for .NET implementation. Aspose.Words is a very powerful development tool that is well documented and easy to install. The documentation is easy to understand and use. If you want a product to convert Word Docs to HTML look no further. ProHire is happy to recommend Aspose.

This is an extract from a case study on our website. For the full version, go to: www.aspose.com/corporate/customers/case-studies.aspx

The converted HTML resume version needed to look close to the original.

Open, Create, Convert, Print & Save Files

from within your *own* applications.

ASPOSE.TOTAL

allows you to process these file formats:

- Word documents
- Excel spreadsheets
- PowerPoint presentations
- PDF documents
- Project documents
- Visio documents
- Outlook emails
- OneNote documents

DOC XLS PPT PDF EML
PNG XML RTF HTML VSD
BMP & barcode images.

ASPOSE
File Format APIs

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

Helped over 11,000 companies and over 300,000 users work with documents in their applications.

File Format APIs

GET STARTED NOW

- Free Trial
- 30 Day Temp License
- Free Support
- Community Forums
- Live Chat
- Blogs
- Examples
- Video Demos

Adding File Conversion and Manipulation to Business Systems

How often do people in your organization complain that they can't get information in the file format and layout they want? Converting documents from one format to another without losing layout and formatting should be simple, but it can be frustrating for both users and developers.

EXTRACTING DATA FROM A DATABASE AND DELIVERING IT TO THE SALES TEAM AS A REPORT, complete with charts and corporate branding, is fine. Until the sales team says that they want it as a Microsoft Excel file, and could you add a dashboard?

Using information from online forms in letters that can are printed and posted is easy. But what if you also want to add tracking barcodes and archive a digital copy as a PDF?

> Aspose creates APIs that work independently of Microsoft Office Automation.

Ensuring that your business system supports all the different Microsoft Office file formats your users want can be difficult. Sometimes the native file format support of your system lets you down. When that is the case, use tools that extend that capability. A good tool can save you time and effort.

Document Conversion Options

Building your own solution: Time-consuming and costly, this option is only sensible if the solution you develop is central to your business.

Using Microsoft Office Automation: Microsoft Office Automation lets you use Microsoft Office programs server-side. It is not how the Office products were designed to be used. It can work well but you might notice issues with the stability, security and speed of the system, as well as cost.

Using an API: The API market has lots of free and commercial solutions, some very focused, some feature-rich. An API integrates with your code and gives you access to a range of new features.

Look to Aspose

Aspose are API experts. We create APIs, components and extensions that work independently of Microsoft Automation to extend a platform's native file format manipulation capabilities.

Aspose have developed APIs for .NET, Java, Cloud and Android that lets developers convert, create and manipulate Microsoft Office files – Microsoft Word, Excel, PowerPoint, Visio and Project – and other popular business formats, from PDFs and images to emails. We also have APIs for working with images, barcodes and OCR. The APIs are optimised for stability, speed and ease of use. Our APIs save users weeks, sometimes months, of effort.

Finding the Right Tool

To find the product that's right for you, take a systematic approach:

- List must-have and nice-to-have features.
- Research the market.
- Ask for recommendations.
- Select a few candidates .
- Run trials.
- Evaluate
  - ease of use,
  - support and documentation,
  - performance, and
  - current and future needs.

Aspose.BarCode

A complete toolkit for barcode generation and recognition

- Generate barcodes with customer defined size and color.
- Recognize a large number of barcode types from images.

ASPOSE.BARCODE IS A ROBUST AND RELIABLE BARCODE GENERATION AND RECOGNITION API that allows developers to add barcode generation and recognition functionality to their applications quickly and easily.

Aspose.BarCode supports most established barcode specifications. It can export generated barcodes to multiple image formats, including BMP, GIF, JPED, PNG and TIFF.

Aspose.BarCode gives you full control over every aspect of the barcode image, from background and bar color, through image quality, rotation angle, X-dimension, captions, and resolution.

> Robust and reliable barcode generation and recognition.

Aspose.BarCode can read and recognize most common 1D and 2D barcodes from any image and at any angle. Filters help developers clean up difficult to read images to improve recognition.

Aspose.BarCode offers a large number of symbologies and formatting options.

Common Uses

- Generating and recognizing barcode images.
- Printing barcode labels.
- Enhancing workflow by adding barcode functionality.
- Using recognition functions to drive real-life work processes.

Key Features

- Barcode generation and recognition.
- Comprehensive support for 1D and 2D symbologies.
- Image processing for improved recognition.

Supported File Formats

JPG, TIFF, PNG, BMP, GIF, EMF, WMF, EXIP and ICON.

Format support varies across platforms.

Supported Barcodes

Linear: EAN13, EAN8, UPCA, UPCE, Interleaved2of5, Standard2of5, MSI, Code11, Codabar, EAN14(SCC14), SSCC18, ITF14, Matrix 2 of 5, PZN, Code128, Code39 Extended, Code39 Standard, OPC, Code93 Extended, Code93 Standard, IATA 2 of 5, GS1Code128, ISBN, ISMN, ISSN, ITF6, Pharmacode, DatabarOmniDirectional, VIN, DatabarTruncated, DatabarLimited, DatabarExpanded, PatchCode, Supplement **2D**: PDF417, MacroPDF417, DataMatrix, Aztec, QR, Italian Post 25, Code16K, GS1DataMatrix **Postal**: Postnet, Planet, USPS OneCode, Australia Post, Deutsche Post Identcode, AustralianPosteParcel, Deutsche Post Leticode, RM4SCC, SingaporePost, SwissPostParcel

Platforms

Pricing Info					
	Standard	Enhanced		Standard	Enhanced
Developer Small Business	$599	$1098	Site Small Business	$2995	$5490
Developer OEM	$1797	$3294	Site OEM	$8386	$15372

The pricing info above is for .NET: prices for other platforms may differ. For the latest, contact sales.

Aspose *for* Cloud

The easiest API to Create, Convert & Automate Documents in the cloud.

Convert
Create
Render
Combine
Modify

without installing anything!

Aspose.Words for Cloud Create and convert docs Manipulate text Render documents Annotate	**Aspose.Cells for Cloud** Create spreadsheets Convert spreadsheets Manipulate cells and formulas Render spreadsheets
Aspose.Slides for Cloud Create presentations Manage slides Edit text and images Read and convert	**Aspose.Pdf for Cloud** Create and convert PDFs Manipulate text, images Add pages, split, encrypt Manage stamps
Aspose.Email for Cloud Create, update, and convert messages Extract attachments Use with any language	**Aspose.BarCode for Cloud** Generate barcodes Read barcodes Set attributes Multiple image formats

Free Evaluation at www.aspose.com

- +44 141 628 8900
- +1 903 306 1676
  sales@aspose.com
- +61 2 8006 6987

Aspose.Email

Work with emails and calendars without Microsoft Outlook

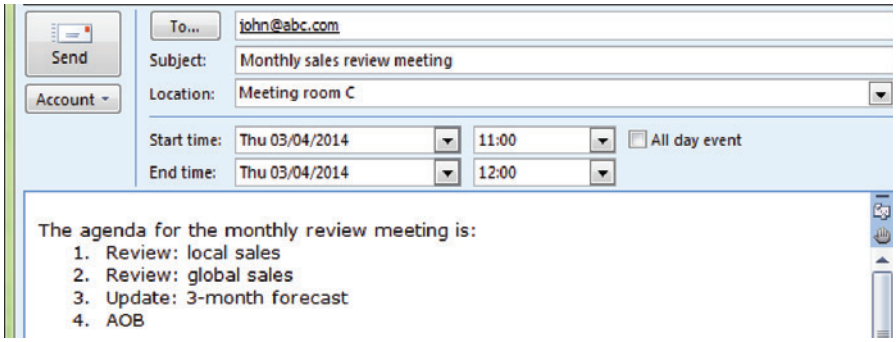
- Complete email processing solution.
- Message file format support.

ASPOSE.EMAIL IS AN EMAIL PROGRAMMING API that allows developers to access and work with PST, EML, MSG and MHT files. It also offers an advanced API for interacting with enterprise mail systems like Exchange and Gmail.

Aspose.Email can work with HTML and plain text emails, attachments and embedded OLE objects. It allows developers to work against SMTP, POP, FTP and Microsoft Exchange servers. It supports mail merge and iCalendar features, customized header and body, searching archives and has many other useful features.

Aspose.Email works with HTML and plain text emails, attachments and embedded OLE objects.

Aspose.Email allows developers to focus on managing email without getting into the core of email and network programming. It gives you the controls you need.



Aspose.Email lets your applications work with emails, attachments, notes and calendars.

Common Uses

- Sending email with HTML formatting and attachments.
- Mail merging and sending mass mail.
- Connecting to POP3 and IMAP mail servers to list and download messages.
- Connecting to Microsoft Exchange Servers to list, download and send messages.
- Create and update tasks using iCalendar.
- Load from and save messages to file or stream (EML, MSG or MHT formats).

Key Features

- A complete email processing solution.
- Support for MSG and PST formats.
- Microsoft Exchange Server support.
- Complete recurrence pattern solution.

Supported File Formats

MSG, MHT, OST, PST, EMLX, TNEF, and EML.

Format support varies across platforms.

Platforms

Pricing Info					
	Standard	Enhanced		Standard	Enhanced
Developer Small Business	$599	$1098	Site Small Business	$2995	$5490
Developer OEM	$1797	$3294	Site OEM	$8386	$15372

The pricing info above is for .NET: prices for other platforms may differ. For the latest, contact sales.

Aspose.Pdf

Create PDF documents without using Adobe Acrobat

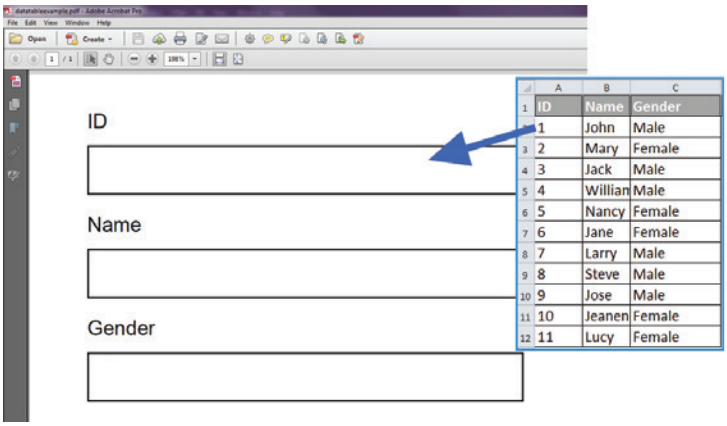
- A complete solution for programming with PDF files.
- Work with PDF forms and form fields.

ASPOSE.PDF IS A PDF DOCUMENT CREATION AND MANIPULATION API that developers use to read, write and manipulate PDF documents without using Adobe Acrobat. Aspose.Pdf is a sophisticated product that integrates with your application to add PDF capabilities.

Aspose.Pdf offers a wealth of features that lets developers compress files, create tables, work with links, add and remove security, handle custom fonts, integrate with external data sources, manage bookmarks, create table of contents, create forms and manage form fields.

Read, write and manipulate PDF documents independently of Adobe Acrobat.

It helps developers add, work with attachments, annotations and PDF form data, add, replace or remove text and images, split, concatenate, extract or inset pages, and print PDF documents.



Aspose.Pdf can be used to automatically complete PDF forms with external data.

Common Uses

- Creating and editing PDF files.
- Inserting, extracting, appending, concatenating and splitting PDFs.
- Working with text, images, tables, images, headers, and footers.
- Applying security, passwords and signatures.
- Working with forms and form fields.

Key Features

- PDF creation from XML or XSL-FO documents.
- PDF form and field support.
- Advanced security and encryption.
- High-fidelity printing and conversion.
- Supported File Formats
- PDF, PDF/A, PDF/A_1b, PCL, XSL-FO, LaTeX, HTML, XPS, TXT and a range of image formats.

Format support varies across platforms.

Platforms

Pricing Info					
	Standard	Enhanced		Standard	Enhanced
Developer Small Business	$799	$1298	Site Small Business	$3995	$6490
Developer OEM	$2397	$3894	Site OEM	$11186	$18172

The pricing info above is for .NET: prices for other platforms may differ. For the latest, contact sales.

Aspose.Pdf

.Net, Java & Cloud

File Formats

PDF DOC XML XSL-FO XPS HTML BMP JPG PNG ePUB & other image file formats.

Create and Manipulate PDFs

Create new or edit/manipualte existing PDFs.

Form Field Features

Add form fields to your PDFs. Import and export form fields data from select file formats.

Table Features

Add tables to your PDFs with formatting such as table border style, margin and padding info, column width and spanning options, and more.

Aspose.Note for .NET

Aspose.Note for .NET is an API that lets developers convert Microsoft OneNote pages to a variety of file formats, and extract the text and document information.

Conversion is fast and high-fidelity. The output looks like the OneNote page, no matter how complex the formatting or layout.

Aspose.Note works independently of Office Automation and does not require Microsoft Office or OneNote to be installed.

Modify, convert, render and extract text and images from Microsoft OneNote files without relying on OneNote or other libraries.

Features

File Formats and Conversion		Rendering and Printing	Document Management
Microsoft OneNote 2010, 2010 SP1, 2013	Load, Save	Save as Image (BMP, GIF, JPG, PNG)	• Extract text • Get the number of pages in a document. • Get page information. • Extract images. • Get image information from a document. • Replace text in document.
PDF	Save	Save as PDF	
Images (BMP, GIF, JPG, PNG)	Save		

Aspose.Imaging

Create Images from scratch.

- Load existing images for editing purposes.
- Render to multiple file formats.

ASPOSE.IMAGING IS A CLASS LIBRARY that facilitates the developer to create Image files from scratch or load existing ones for editing purpose. Also, Aspose.Imaging provides the means to save the created or edited Image to a variety of formats. All of the above mentioned can be achieved without the need of an Image Editor. It works independent of other applications and although Aspose.Imaging allows you to save to Adobe PhotoShop® format (PSD), you do not need PhotoShop installed on the machine.

Aspose.Imaging allows creation and manipulation of images.

Aspose.Imaging is flexible, stable and powerful. It's many features and image processing routines should meet most imaging requirements. Like all Aspose file format components, Aspose.Imaging introduces support for an advanced set of drawing features along with the core functionality. Developers can draw on Image surface either by manipulating the bitmap information or by using the advanced functionality like Graphics and Paths.

Create images from scratch. or load existing ones...

Common Uses

- Create images from scratch.
- Load and Edit existing images.
- Export images to a variety of formats.
- Adding watermark to images.
- Export CAD drawings to PDF & raster image formats.
- Crop, resize & RotateFlip images.
- Extract frames from multipage TIFF image.

Key Features

- Create, edit, and save images
- Multiple file formats
- Drawing features
- Export images

Supported File Formats

BMP, JPG, TIFF, GIF, PNG, PSD, DXF, DWG, and PDF.

Platforms

Pricing Info					
	Standard	Enhanced		Standard	Enhanced
Developer Small Business	$399	$898	Site Small Business	$1995	$4490
Developer OEM	$1197	$2694	Site OEM	$5586	$12572

The pricing info above is for .NET.

Aspose.Slides

Work with presentations without using Microsoft PowerPoint

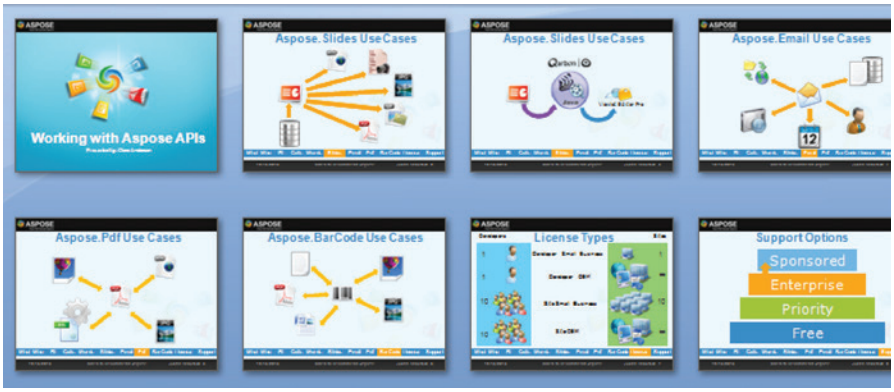
- Complete solution for working with presentation files.
- Export presentations and slides to portable or image formats.

ASPOSE.SLIDES IS A FLEXIBLE PRESENTATION MANAGEMENT API that helps developers read, write and manipulate Microsoft PowerPoint documents. Slides and presentations can be saved to PDF, HTML and image file formats without Microsoft PowerPoint.

Aspose.Slides offers a number of advanced features that make it easy to perform tasks such as rendering slides, exporting presentations, exporting slides to SVG and printing. Developers use Aspose.Slides to build customizable slide decks, add or remove standard graphics and automatically publish presentations to other formats.

> Aspose.Slides gives you the tools you need to work with presentation files.

Aspose.Slides gives developers the tools they need to work with presentation files. It integrates quickly and saves time and money.



Aspose.Slides has advanced features for working with every aspect of a presentation.

Common Uses

- Creating new slides and cloning existing slides from templates.
- Handling text and shape formatting.
- Applying and removing protection.
- Exporting presentations to images and PDF.
- Embedding Excel charts as OLE objects.
- Generate presentations from database.

Key Features

- A complete presentation development solution.
- Control over text, formatting and slide elements.
- OLE integration for embedding external content.
- Wide support for input and output file formats.

Supported File Formats

PPT, HTML, POT, PPS, PPTX, POTX, PPSX, ODP, PresentationML, XPS, PDF and image formats including TIFF and JPG.

Format support varies across platforms.

Platforms

Pricing Info					
	Standard	Enhanced		Standard	Enhanced
Developer Small Business	$799	$1298	Site Small Business	$3995	$6490
Developer OEM	$2397	$3894	Site OEM	$11186	$18172

The pricing info above is for .NET: prices for other platforms may differ. For the latest, contact sales.

Support Services

Get the assistance you need, when you need it, from the people who know our products best.

- Free support for all, even when evaluating
- Get the level of support that suits you and your team

NO ONE KNOWS OUR PRODUCTS AS WELL AS WE DO.
We develop them, support them and use them. Our support is handled through our support forums and is available to all Aspose users.

Support

We are developers ourselves and understand how frustrating it is when a technical issue or a quirk in the software stops you from doing what you need to do. This is why we offer free support. Anyone who uses our product, whether they have bought them or are using an evaluation, deserves our full attention and respect. We have four levels of support that can fit your needs.

> Everyone who uses Aspose products have access to our free support.

Work with the developers that developed and continue to maintain our products.

Support Options

Free

Everyone who uses Aspose products have access to our free support. Our software developers are on stand-by to help you succeed with your project, from the evaluation to roll-out of your solution.

Priority

If you want to know when you'll hear back from us on an issue and know that your issue is prioritized, Priority Support is for you. It provides a more formal support structure and has its own forum that is monitored by our software engineers.

Enterprise

Enterprise customers often have very specific needs. Our Enterprise Support option gives them access to the product development team and influence over the roadmap. Enterprise Support customers have their own, dedicated issue tracking system.

Sponsored

Available to Enterprise customers that would like to request features, this higher prioritized support can ensure your needed features are on our roadmap. A member of our team will produce a feature specification document to capture your requirements and how we intend to fulfill them so the direction development will take is clear up-front.

Pricing Info
To see the Priority and Enterprise support rates, refer to the product price list, or contact our sales team.
Sponsored Support is unique so pricing is specific to each project. Please contact our sales team to discuss.

Aspose has 4 Support Services to best suit your needs

Free Support	Support Forums with no Charge
Priority Support	24 hour response time in the week, issue escalation, dedicated forum
Enterprise Support	Communicate with product managers, influence the roadmap
Sponsored Support	Get the feature you need built now

Technical Support is an issue that Aspose takes very seriously. Software must work quickly and dependably. When problems arise, developers need answers in a hurry. We ensure that our clients receive useful answers and solutions quickly.

Email • Live Chat • Forums

Contact Us

US Sales: +1 903 306 1676
sales@aspose.com

EU Sales: +44 141 628 8900

AU Sales: +61 2 8006 6987

ASPOSE
File Format APIs