THE MICROSOFT JOURNAL FOR DEVELOPERS

OCTOBER 2017 VOL 32 NO 10

msdn magazine

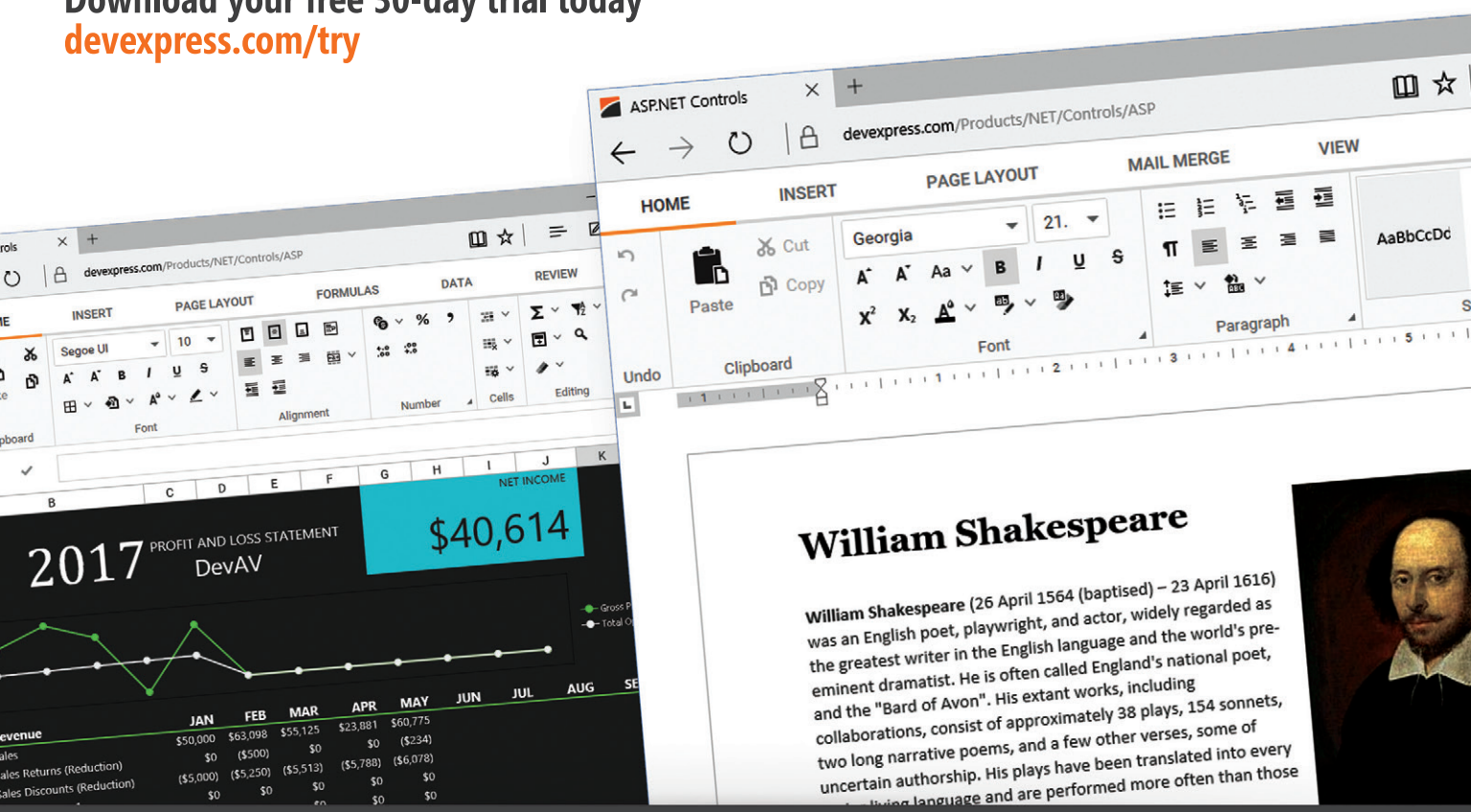
C++

Office-Inspired
ASP.NET & MVC Controls

DevExpress®

Create high-impact line-of-business applications for the web with the DevExpress ASP.NET Subscription.

Download your free 30-day trial today
devexpress.com/try



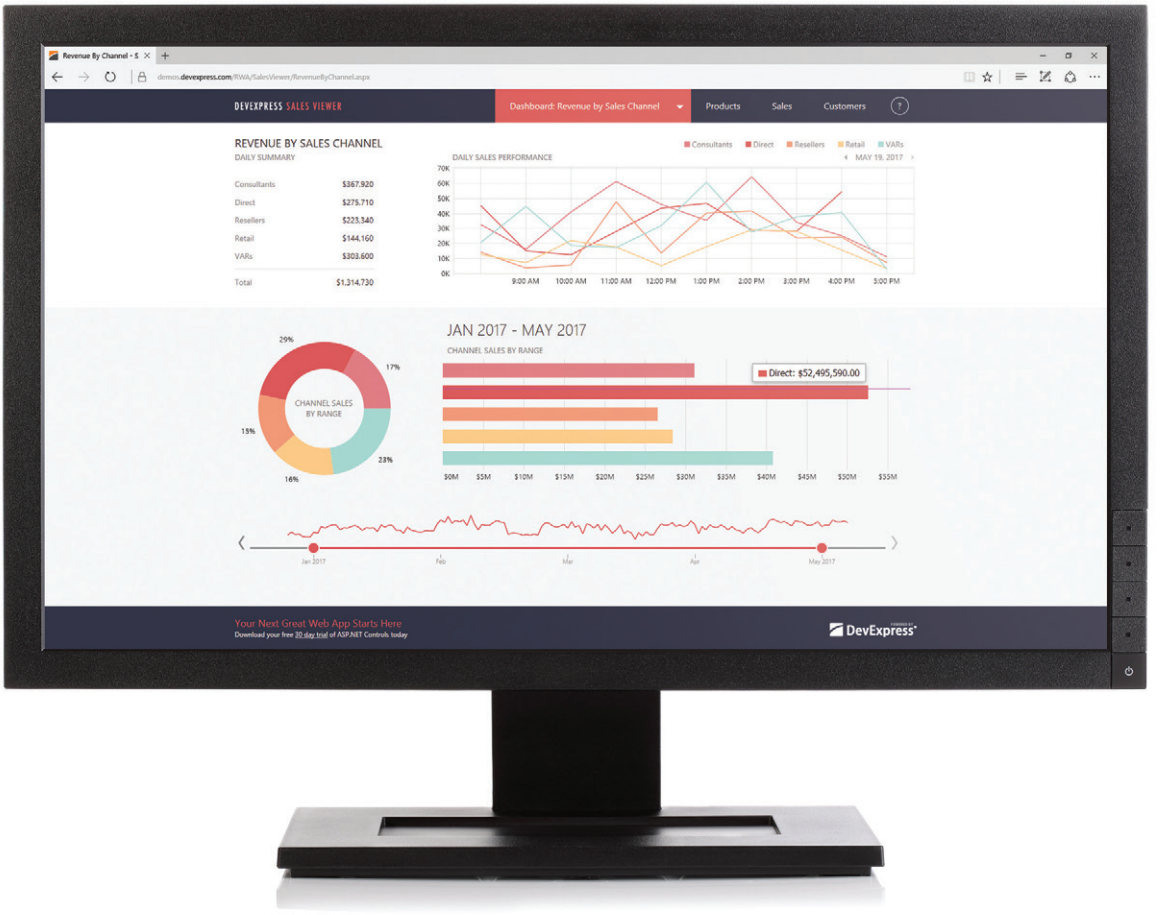
Visual Studio Magazine Readers Choice 2016 2015 2013 2012 Winner

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

DevExpress®

Your Next Great Web App Starts Here

From apps that replicate the look and feel of Microsoft Office® 365, to high-impact decision support systems for your enterprise, DevExpress Web Controls for ASP.NET 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

OCTOBER 2017 VOL 32 NO 10

msdn magazine

C++

Coroutines in C++.....24

COLUMNS

Microsoft

INFRAGISTICS

Write Fast, Run Fast

with **Infragistics Ultimate** Developer Toolkit

Includes 100+ beautiful, fast grids, charts, and other UI controls, plus productivity tools for quickly building high-performing web, mobile, and desktop apps

Featuring

- Xamarin UI controls with innovative, code-generating productivity tools
- JavaScript/HTML5 and ASP.NET MVC components, with support for:

ANGULAR ANGULARJS Knockout. React Bootstrap

Also includes controls for WPF, Windows Forms, and ASP.NET, plus prototyping, remote usability testing, and more.

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

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


Till 3am
Roger Deep
MUSIC BY GENRE
Hip Hop 25%
Country 35%
Rock & Roll 18%
Metal 25%
EDM 25%
Jazz 25%
FINANCE
Example Corporation (EXMPL)
23.18
2/15/2017, 9:55 AM
-0.34 (-0.72%)
OCT
SEP
SEP
SEP
Close
Contains...
22.34
22.288
21.9729
22.12
21.7834
1 - 5 of 10 records
Date
Contains...
Thu Jan 19 2017
Wed
Tue
Mon
Sun
High
Mail App
File
Active
Tab Item
Item Hover
Cut
Copy
Paste
Delete
Clipboard
Text Style
Find and Replace
Editing
inbox@mail.com
Inbox 13
Clutter
Drafts 1
Sent Items
Junk Email
From
Name LastName
Sed ut perspiciatis unde omnis iste natus
This is th
Sometimes it co
Very short
But if it's longer and it doesn't
This is the subject of the email
This is the subject of the email, it could be p
Reply
Reply All
This is the subject of the email
JULY 29
WEEK 31
Expense
100
0
S
M
T
W
Th
F
S
AVG. $45
TODAY
Coffee
@ STARBUCKS
$19.25
Movie
RISE OF GORT
4 75
YESTERDAY
Travel
4 75

msdn magazine

OCTOBER 2017 VOLUME 32 NUMBER 10

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

1105 ENTERPRISE COMPUTING GROUP

President
Henry Allain

Chief Revenue Officer
Dan LaBianca

Chief Marketing Officer
Carmel McDonagh

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

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

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

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

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

ENTERPRISE COMPUTING GROUP EVENTS
Vice President, Events **Brent Sutton**
Senior Director, Operations **Sara Ross**
Senior Director, Event Marketing **Merikay Marzoni**
Events Sponsorship Sales **Danna Vedder**
Senior Manager, Events **Danielle Potts**
Coordinator, Event Marketing **Michelle Cheng**
Coordinator, Event Marketing **Chantelle Wallace**

1105 MEDIA INC
YOUR GROWTH. OUR BUSINESS.

Chief Executive Officer
Rajeev Kapur

Chief Operating Officer
Henry Allain

Chief Financial Officer
Craig Rucker

Chief Technology Officer
Erik A. Lindgren

Executive Vice President
Michael J. Valenti

Chairman of the Board
Jeffrey S. Klein

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

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

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

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

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

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

LIST RENTAL This publication's subscriber list, 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: meritdirect.com/1105

Reaching the Staff

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

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

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

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

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

Microsoft

BPA WORLDWIDE

1906 100 2006 AMERICAN BUSINESS MEDIA

WPA Western Publications Association

LEADTOOLS®

THE WORLD LEADER IN IMAGING SDKs

Build Better Apps with LEADTOOLS SDKs

LEADTOOLS is a family of comprehensive toolkits designed to help programmers integrate raster, document, medical, multimedia, and vector imaging into their desktop, web, server, tablet, and phone applications.

SDKs for WEB

SDKs for MOBILE

SDKs for DESKTOP

SDKs for SERVER

.NET WinRT Windows API Java Linux iOS macOS Android JavaScript

DOWNLOAD OUR 60 DAY EVALUATION
WWW.LEADTOOLS.COM

LEAD TECHNOLOGIES INCORPORATED

SALES@LEADTOOLS.COM
800.637.1840

Editor's Note

MICHAEL DESMOND

Misprint

There are things that send me into immediate rage. Drivers staring at their smartphones when the light turns green. Or when I snag the wire of my earbuds on an object so they get ripped forcefully out of my ears. Or, almost any time I have to print something.

Not to get all "if we can send a man to the moon," but how can it still be this frustrating to shoot a few bits across six feet of USB cable and have them laid out properly on a piece of paper?

I was thinking about this after reading David Platt's Don't Get Me Started column this month, where he concocts a handy metric to determine the difficulty and brokenness of a user experience, inspired by his struggle to connect an iPhone to the Harvard University Wi-Fi network. His column got me to thinking about another mundane-yet-frustrating computing experience, and one that's been around for decades: printing.

In an age of smartphones, autonomous cars and AI-driven assistant technology, it's astonishing how little printing has changed since 1995. Printing a document is like lobbing a really unreliable hand grenade over a wall. Your grenade rolls around in the print queue for a few seconds or forever, depending on the whims of the printer driver, the connectivity between printer and PC, ink and toner levels, and a host of other variables. I've opened the print queue window after failing to see activity, only to discover a mob of failed and forgotten print jobs stuck in there. It's infuriating.

I asked Platt about this, and he was blunt. "I cringe when my wife announces she's gotten a document that she needs printed."

Platt gets especially worked up over the unhelpful error messages from his printer. "Unable to print because printer is offline," said Platt, a professor at the Harvard University Extension: "If one of my UX students threw such a cryptic message, I'd flunk his sorry ass so fast he'd change his major to Sanskrit."

Any IT manager can regale you with stories of undiagnosable printer failures, but even at the home-office level printing is an exercise in fail. When I try to print, my Windows 7 home PC and Windows 10 laptop both often show duplicate printer instances in the Windows Print dialog box. Should I send the job to the printer named Brother HL-2340D series, or to Brother HL-2340D series (Copy 1) or (Copy 2) instead? It's insane.

It doesn't have to be this way. The solution to bad user experiences is better developer training and education. Big ticket events like Microsoft Build and Ignite are incredibly important in this regard, but I'm a particular fan of smaller conferences like Live! 360, which focuses on hands-on experiences and face-to-face interaction with presenters and experts.

> Printing a document is like lobbing a really unreliable hand grenade over a wall.

A few weeks ago I hosted a Web cast with Dr. Doris Chen, a senior developer evangelist at Microsoft is presenting a pair of technical sessions at the Visual Studio Live! event in Anaheim later this month (bit.ly/2wZ5fI7). Her work informing the dev community on Web and JavaScript development is vitally important as we continue to build out and rely on Web-based software and services. Looking forward, the Live! 360 event in Orlando (Nov. 12-17) is another opportunity for developers to advance their craft by working in close quarters to some top minds in the industry. You can learn more about the event at bit.ly/Live360FL.

Look, I'm not saying that attending a hands-on technical conference will spare anyone the indignity of being flummoxed by the Windows Print Queue. Heck, my Windows 7 desktop spawned a Brother HL-2340D (Copy 3) in the printer list just this afternoon. So, the struggle continues. But it's events like Visual Studio Live! and Live! 360—small, accessible and hands-on—that can help move us all to a better place.

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

© 2017 Microsoft Corporation. All rights reserved.

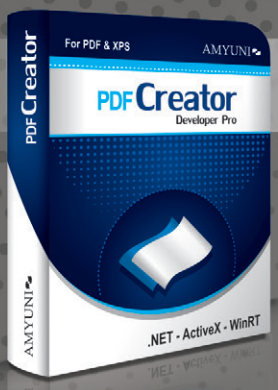
Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, you are not permitted to reproduce, store, or introduce into a retrieval system *MSDN Magazine* or any part of *MSDN Magazine*. If you have purchased or have otherwise properly acquired a copy of *MSDN Magazine* in paper format, you are permitted to physically transfer this paper copy in unmodified form. Otherwise, you are not permitted to transmit copies of *MSDN Magazine* (or any part of *MSDN Magazine*) in any form or by any means without the express written permission of Microsoft Corporation.

A listing of Microsoft Corporation trademarks can be found at microsoft.com/library/toolbar/3.0/trademarks/en-us.mspx. Other trademarks or trade names mentioned herein are the property of their respective owners.

MSDN Magazine is published by 1105 Media, Inc. 1105 Media, Inc. is an independent company not affiliated with Microsoft Corporation. Microsoft Corporation is solely responsible for the editorial contents of this magazine. The recommendations and technical guidelines in *MSDN Magazine* are based on specific environments and configurations. These recommendations or guidelines may not apply to dissimilar configurations. Microsoft Corporation does not make any representation or warranty, express or implied, with respect to any code or other information herein and disclaims any liability whatsoever for any use of such code or other information. *MSDN Magazine*, MSDN and Microsoft logos are used by 1105 Media, Inc. under license from owner.

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

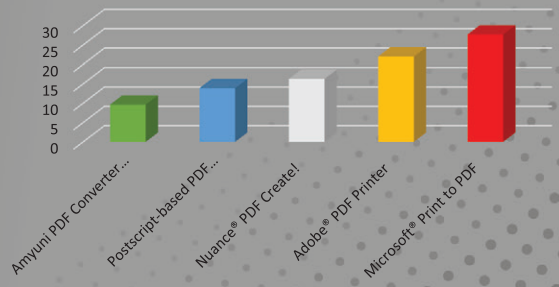
Other Developer Components from Amyuni®

- 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

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



CERTIFIED FOR Windows Server® 2008

Windows Server 2012 Certified

Windows 7

Windows

All trademarks are property of their respective owners. © Amyuni Technologies Inc. All rights reserved.

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

The Engineer's Path: 2 Decisions That Define a Career

Engineers face a host of unique career challenges. In this column, I deal with two specific ones: The choice between the technical track and management track, and a more fundamental decision about how to "show up" at work.

Management Muddle

For some engineers, the opportunity to transition into management comes up as they grow and, often, they feel conflicted about it. Those who *do* make the switch may find they're still haunted by self-doubt, prompting them to regularly consider a return to engineering. Here are five questions you can ask yourself to know if management is a good fit for you:

Do I seek growth through external interactions or internal struggles? Management is a special opportunity to experience conflicts with others. This serves as a mirror into your own strengths and blind spots, and accelerates your personal growth.

Engineering, being more contemplative, demands that you grow through *internal* struggles—unlocking your personal productivity, developing a single-minded focus on technical problems and demanding a deep desire for self-driven growth.

Do I enjoy technical depth or strategic breadth? Managers are "good enough" experts in many areas. They're OK with having just passing knowledge of other areas. They learn to make good decisions with bad data.

Engineers deliver their best value with mastery and expertise. They enjoy the patience and endurance it takes to dive deeper into one or two areas, for weeks or months at a time.

Do I have my act together? As a manager, you're responsible for the careers of many engineers, not just your own. If you feel helpless or out of control in your own career, you can't help but propagate chaos to those you manage.

Engineering, on the other hand, is the perfect opportunity to work through your inner conflicts by yourself. This will help make you a better manager when the time is right.

Do I enjoy discovering calm amidst chaos, or finding beauty in complexity? As a manager, you're hit with demands from all sides. Good managers enjoy fulfilling the right requests and sidestepping the wrong ones. The best managers know that mastering this dance is a lifelong art.

As an engineer, you're responsible for solving open-ended, complex and highly technical problems. You may not even know *what* the problem is, so relishing the scientific method is key.

What would I do if I'm paid the same? You might be driven by a desire for money or status without even knowing it. Yet, externally, you might justify your choices by mentioning anything *but* these things. Ask yourself, "How can I contribute the most to the world with my skills and abilities right now, regardless of how much recognition or money I get? Which option plays to my strengths while helping me deepen my personal growth?"

A Question of Courage

Once you decide which path to take, you might feel some anxiety, inadequacy or fear. For engineers, this feeling might be so intense that there's a temptation to hide and try to be invisible. But invisibility only hinders your ability to contribute at work. So you have a choice: Do you choose to be a *timid engineer* who becomes invisible in the face of a challenge, or a *brave engineer* who stays visible despite the risk?

A brave engineer recognizes her own nervousness and thinks, "I want to tap this energy and turn it into enthusiasm!" A timid engineer sees his own nervousness and thinks, "Ugh, I'm so nervous. I'll put this off until I feel better." Brave engineers experience the feelings and emotions that timid engineers do—they've just practiced interpreting them differently.

Bravery also recognizes the universal nature of humanity. Brave engineers understand that managers have many of the same fears and personal insecurities as a junior engineer. By contrast, timid engineers ascribe super-human qualities to their superiors, imagining them to be perfect while considering themselves to be deeply flawed.

Brave engineers know that the fastest path to success is to commit to embracing and transcending their own imperfections. They view work interactions as fun games they design themselves, whereas timid engineers see work interactions as a maze from which they must escape.

Here's the good news: Timid engineers can become brave engineers quickly, just by making their next action a tiny act of courage. It's possible to undo a decade of fear with just six months of constant practice. That's what I did. ■

KRISHNAN RANGACHARI *is a leadership coach to CTOs and technical leaders. Visit RadicalShifts.com for his free courses for engineers.*

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

JULIE LERMAN

DDD-Friendlier EF Core 2.0, Part 2

In my September column (msdn.com/magazine/mt842503), I laid out the many Entity Framework Core (EF Core) 2.0 features that align nicely with Domain-Driven Design (DDD) principles. In addition to providing great guidance and patterns for software development, DDD principles are also critical if you're designing microservices. In the examples throughout the article, I used simplistic patterns in order to focus on the particular EF Core feature. Doing this meant that the code didn't represent well-designed DDD-guided classes, and I promised that in an upcoming column I'd evolve those classes to look more like what you might write for a real-world implementation using DDD. And that's what I'm going to do in this article. I'll walk you through these better-architected classes and show you how they continue to work well as I use EF Core 2.0 to map them to my database.

The Original Domain Model

I'll begin with a quick refresher on my little domain model. Because it's for a sample, the domain lacks the complex business problems that would generally drive you to lean on DDD, but even without those complicated problems, I can still apply the patterns so you can see them in action, and see how EF Core 2.0 responds to them.

The domain comprises the Samurai characters from the movie "Seven Samurai," where I keep track of their first appearance in the movie and their secret identities.

In the original article, the Samurai was the root of the aggregate and I had constrained the model to ensure the Samurai was responsible for managing its entrances and its secret identity. I demonstrated some of those constraints as follows:

Samurai and Entrance have a one-to-one relationship. Samurai's Entrance field is private. Entrance has a foreign key field, SamuraiId. Because Samurai.Entrance is private, I needed to add a Fluent API mapping in the DbContext class to be sure EF Core was able to comprehend the relationship for retrieving and persisting this data. I evolved the Entrance property to be tied to a backing field, and then modified the mappings to let EF Core know about this, as well.

PersonName_ValueObject (named so elaborately for your benefit) is a value object type without its own identity. It can be used as a property in other types. Samurai has a PersonName_ValueObject property called SecretIdentity. I used the new EF Core Owned Entity feature to make SamuraiContext know to treat the SecretIdentity the same as earlier versions of EF would handle a ComplexType, storing the properties of the value object in columns of the same table to which the Samurai type maps.

The Enhanced Domain Model

What follows are the more advanced classes in the aggregate, along with the EF Core 2.0 DbContext I'm using to map to the database, which in my case happens to be SQLite. The diagram in **Figure 1** shows the aggregate with its class details. The code listings will start with the non-root entities and finish up with the root, Samurai, which controls the others. Note that I've removed

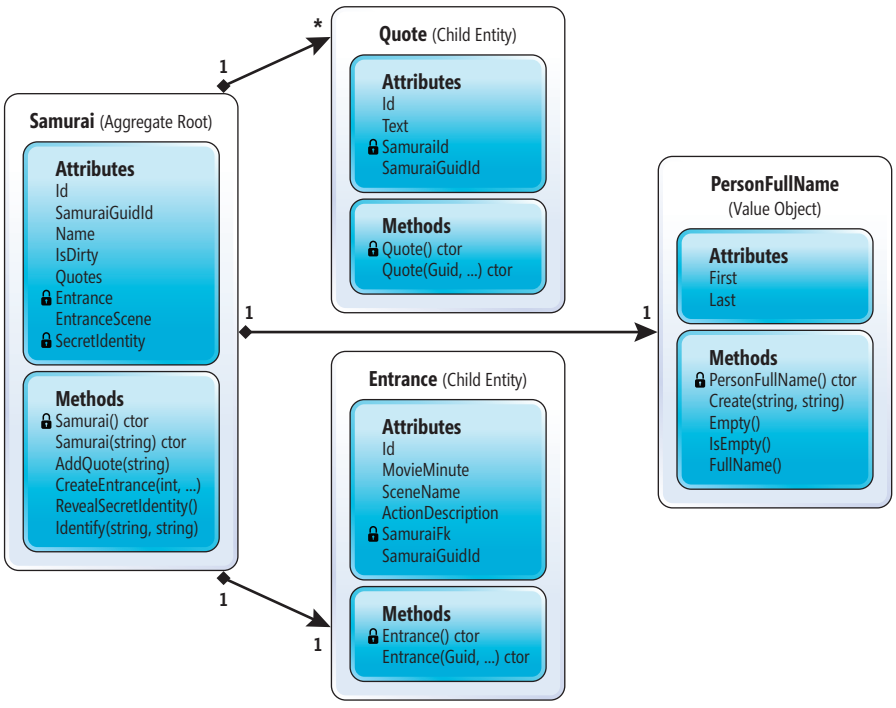


Figure 1 **Diagram of the Advanced Aggregate**

Code download available at msdn.com/magazine/1017magcode.

namespace references, but you can see them in the download that accompanies this article.

Figure 2 shows the evolved Entrance class.

So much of DDD code is about protecting your domain from being unintentionally misused or abused. You constrain access to the logic within the classes to ensure they can be used only in the way you intend. My intention for the Entrance class (**Figure 1**) is that it be immutable. You can define its property values using the overloaded constructor, passing in the values for all of its properties except for SamuraiFk. You're allowed to read any of the properties—but notice they all have private setters. The constructor is the only way to affect those values. Therefore, if you need to modify it, you'll need to replace it with a whole new Entrance instance. This class looks like a candidate for a value object, especially because it's immutable, but I want to use it to demonstrate one-to-one behavior in EF Core.

With EF Core (and earlier iterations of EF), when you query for data, EF is able to materialize results even when properties have private setters because it uses reflection. So EF Core can work with all these properties of Entrance that have private setters.

There's a public constructor with four parameters to populate properties of Entrance. (In the previous sample, I used a factory method that added no value to this class, so I've removed it in this iteration.) In this domain, an Entrance with any of those properties missing makes no sense, so I'm constraining its design to avoid that. Following that constructor is a private parameterless constructor. Because EF Core and EF use reflection to materialize results, like other APIs that instantiate objects for you—such as JSON.NET—it requires that a parameterless constructor be available. The first constructor overrides the parameterless constructor that's provided by the base class (object) that all classes derive from. Therefore, you must explicitly add that back in. This is not new behavior to EF Core; it's something you've had to do with EF for a long time. In the context of this article, however, it bears repeating. If you're new to EF with this version, it's also notable that when an Entrance is created as a result of a query, EF Core will only use that parameterless constructor to create the object. The public constructor is available for creating new Entrance objects.

Figure 2 **The Entrance Class Designed Following DDD Patterns**

```
public class Entrance {
  public Entrance (Guid samuraiGuidId,int movieMinute, string sceneName,
string description) {
    MovieMinute = movieMinute;
    SceneName = sceneName;
    ActionDescription = description;
    SamuraiGuidId=samuraiGuidId;
  }
  private Entrance () { } // Needed by ORM
  public int Id { get; private set; }
  public int MovieMinute { get; private set; }
  public string SceneName { get; private set; }
  public string ActionDescription { get; private set; }

  private int SamuraiFk { get;  set; }
  public Guid SamuraiGuidId{get;private set;}
}
```

Figure 3 **The Quote Type Designed Following DDD Patterns**

```
public class Quote {
  public Quote (Guid samuraiGuidId,string text) {
    Text = text;
    SamuraiGuidId=samuraiGuidId;
  }
  private Quote () { } //ORM requires parameterless ctor
  public int Id { get; private set; }
  public string Text { get; private set; }
  private int SamuraiId { get; set; }
  public Guid SamuraiGuidId{get;private set;}
}
```

The first constructor overrides the parameterless constructor that's provided by the base class (object) from which all classes derive.

What about that Guid and int pointing back to Samurai? The Guid is used by the domain to connect the samurai and entrance so that the domain logic has no reliance on the data store for its Ids. The SamuraiFk will only be used for persistence. SamuraiFk is private, but EF Core is able to infer a backing field for it. If it were named SamuraiId, EF Core would recognize it as the foreign key, but because it doesn't follow convention, there's a special mapping in the context to let EF Core know that it is, indeed, the foreign key. The reason it's private is that it's not relevant to the domain but needed for EF Core to comprehend the relationship in order to store and retrieve the data correctly. This is a concession to avoiding persistence logic in my domain class but, in my opinion, a minor one that doesn't justify the extra effort of introducing and maintaining a completely separate data model.

There's a new entity in my aggregate: Quote, shown in **Figure 3**. In the movie this sample domain honors, the various characters have some notable quotes that I want to keep track of in this domain. It also gives me a chance to demonstrate a one-to-many relationship.

Note that the patterns are the same as those I've explained for the Entrance entity: the overloaded public constructor and the private parameterless constructor, the private setters, the private foreign key property for persistence, and the Guid. The only difference is that the SamuraiId, used as the persistence FK, follows EF Core convention. When it's time to look at the DbContext class, there won't be a special mapping for this property. The reason I've named these two properties inconsistently is so you can see the difference in the mappings for the conventional vs. unconventional naming.

Next is the PersonFullName type (renamed from PersonName), shown in **Figure 4**, which is a value object. I explained in the previous article that EF Core 2.0 now allows you to persist a value object by mapping it as an Owned Entity of any entity that owns it, such as the Samurai class. As a value object, PersonFullName is used as a property in other types and entities. A value object has no identity of its own, is immutable and isn't an entity. In addition to the previous article, I have also explained value objects in more depth in other articles, as well as in the Pluralsight course,

Domain-Driven Design Fundamentals, which I created with Steve Smith (bit.ly/PS-DDD). There are other important facets to a value object and I use a ValueObject base class created by Jimmy Bogard (bit.ly/13SWd9h) to implement them.

PersonFullName is used to encapsulate common rules in my domain for using a person's name in any other entity or type. There are a number of notable features of this class. Although it hasn't changed from the earlier version, I didn't provide the full listing in the previous article. Therefore, there are a few things to explain here, in particular the Empty factory method and the IsEmpty method. Because of the way Owned Entity is implemented in EF Core, it can't be null in the owning class. In my domain, PersonFullName is used to store a samurai's secret identity, but there's no rule that it must be populated. This creates a conflict between my business rules and the EF Core rules. Again, I have a simple enough solution that I don't feel the need to create and maintain a separate data model, and it doesn't impact how Samurai is used. I don't want anyone using my domain API to have to remember the EF Core rule, so I built two factory methods: You use Create if you have the values and Empty if you don't. And the IsEmpty method can quickly determine the state of a PersonFullName. The entities that use PersonFullName as a property will need to leverage this logic and then anyone using those entities won't have to know anything about the EF Core rule.

Tying It All Together with the Aggregate Root

Finally, the Samurai class is listed in **Figure 5**. Samurai is the root of the aggregate. An aggregate root is a guardian for the entire aggregate, ensuring the validity of its internal objects and keeping them consistent. As the root of this aggregate, the Samurai type is responsible for how its Entrance, Quotes and SecretIdentity properties are created and managed.

Like the other classes, Samurai has an overloaded constructor, which is the only way to instantiate a new Samurai. The only data expected when creating a new samurai is the samurai's known name. The constructor sets the Name property and also generates a value for the GuidId property. The SamuraiId property will get populated by the database. The GuidId property ensures that my domain isn't dependent on the data layer to have a unique identity and that's what's used to connect the non-root entities (Entrance and Quote) to the Samurai, even if the Samurai hasn't yet been persisted and honored with a value in the SamuraiId field. The constructor appends ": this()" to call the parameterless constructor in the constructor chain. The parameterless constructor (reminder: it's also used by EF Core when creating objects from query results) will ensure that the Quotes collection is instantiated and that SecretIdentity is created. This is where I use that Empty factory method. Even if someone writing code with the Samurai never provides values for the SecretIdentity property, EF Core will be satisfied because the property isn't null.

The full encapsulation of Quotes in Samurai isn't new. I'm taking advantage of the support for IEnumerable that I discussed in an earlier column on EF Core 1.1 (msdn.com/magazine/mt745093).

The fully encapsulated Entrance property has changed from the previous sample in only two minor ways. First, because I removed the factory method from Entrance, I'm now instantiating it directly. Second, the Entrance constructor now takes additional values so I'm passing those in even though at this time the Samurai class isn't currently doing anything with these extra values.

Figure 4 **The PersonFullName Value Object**

```
public class PersonFullName : ValueObject<PersonFullName> {
  public static PersonFullName Create (string first, string last) {
    return new PersonFullName (first, last);
  }
  public static PersonFullName Empty () {
    return new PersonFullName (null, null);
  }
  private PersonFullName () { }

  public bool IsEmpty () {
    if (string.IsNullOrEmpty (First) && string.IsNullOrEmpty (Last)) {
      return true;
    } else {
      return false;
    }
  }
  private PersonFullName (string first, string last) {
    First = first;
    Last = last;
  }
  public string First { get; private set; }
  public string Last { get; private set; }
  public string FullName () => First + " " + Last;
}
}
```

Figure 5 **The Samurai Entity, Which Is the Root of the Aggregate**

```
public class Samurai {
  public Samurai (string name): this() {
    Name = name;
    GuidId=Guid.NewGuid();
    IsDirty=true;
  }
  private Samurai () {
    _quotes = new List<Quote> ();
    SecretIdentity = PersonFullName.Empty ();
  }
  public int Id { get; private set; }
  public Guid GuidId{get;private set;}
  public string Name { get; private set; }
  public bool IsDirty { get; private set; }

  private readonly List<Quote> _quotes = new List<Quote> ();
  public IEnumerable<Quote> Quotes => _quotes.ToList ();
  public void AddQuote (string quoteText) {
     // TODO: Ensure this isn't a duplicate of an item already in Quotes collection
    _quotes.Add (Quote.Create(GuidId,quoteText));
    IsDirty=true;
  }

  private Entrance _entrance;
  private Entrance Entrance { get { return _entrance; } }
  public void CreateEntrance (int minute, string sceneName, string description) {
    _entrance = Entrance.Create (GuidId, minute, sceneName, description);
    IsDirty=true;
  }
  public string EntranceScene => _entrance?.SceneName;

  private PersonFullName SecretIdentity { get; set; }
  public string RevealSecretIdentity () {
    if (SecretIdentity.IsEmpty ()) {
      return "It's a secret";
    } else {
      return SecretIdentity.FullName ();
    }
  }
  public void Identify (string first, string last) {
    SecretIdentity = PersonFullName.Create (first, last);
    IsDirty=true;
  }
}
```

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

All trademarks or registered trademarks are propertyof their respective owners.

There are some enhancements to the SecretIdentity property since the earlier sample. First, the property originally was public, with a public getter and a private setter. This allowed EF Core to persist it in the same way as in earlier versions of EF. Now, however, SecretIdentity is declared as a private property yet I've defined no backing property. When it comes time to persist, EF Core is able to infer a backing property so it can store and retrieve this data without any additional mapping on my part. The Identify method, where you can specify a first and last name for the secret identity, was in the earlier sample. But in that case, if you wanted to read that value, you could access it through the public property. Now that it's hidden, I've added a new method, RevealSecretIdentity, which will use the PersonFullName.IsEmpty method to determine if the property is populated or not. If so, then it returns the FullName of the SecretIdentity. But if the person's true identity wasn't identified, the method returns the string: "It's a secret."

There's a new property in Samurai, a bool called IsDirty. Any time I modify the Samurai properties, I set IsDirty to true. I'll use that value elsewhere to determine if I need to call SaveChanges on the Samurai.

So throughout this aggregate, there's no way to get around the rules I built into the entities and the root, Samurai. The only way to create, modify or read Entrance, Quotes and SecretIdentity is through the constrained logic built into Samurai, which, as the aggregate root, is guarding the entire aggregate.

Mapping to the Data Store with EF Core 2.0

The focus of the previous article was on how EF Core 2.0 is able to persist and retrieve data mapped to these constrained classes. With this enhanced domain model, EF Core is still able to work out most of the mappings even with things so tightly encapsulated in the Samurai class. In a few cases I do have to provide a little help to the DbContext to make sure it comprehends how these classes map to the database, as shown in **Figure 6**.

Figure 6 **The SamuraiContext DbContext Class**

```
public class SamuraiContext : DbContext {
  public DbSet<Samurai> Samurais { get; set; }
  protected override void OnConfiguring (DbContextOptionsBuilder optionsBuilder) {
    optionsBuilder.UseSqlite ("Filename=DPO917Samurai.db");
  }
  protected override void OnModelCreating (ModelBuilder modelBuilder) {
    modelBuilder.Entity<Samurai> ()
      .HasOne (typeof (Entrance), "Entrance")
      .WithOne ().HasForeignKey(typeof (Entrance), "SamuraiFk");

    foreach (var entityType in modelBuilder.Model.GetEntityTypes ()) {
      modelBuilder.Entity (entityType.Name).Property<DateTime>
        ("LastModified");
      modelBuilder.Entity (entityType.Name).Ignore ("IsDirty");
    }
    modelBuilder.Entity<Samurai> ().OwnsOne (typeof (PersonFullName),
      "SecretIdentity");
  }

  public override int SaveChanges () {
    foreach (var entry in ChangeTracker.Entries ()
      .Where (e => e.State == EntityState.Added ||
        e.State == EntityState.Modified)) {
      if (!(entry.Entity is PersonFullName))
        entry.Property ("LastModified").CurrentValue = DateTime.Now;
    }
    return base.SaveChanges ();
  }
}
```

Not a lot has changed in the SamuraiContext since the first sample from my first article, but there are a few things to point out as reminders. For example, the OwnsOne mapping lets EF Core know that SecretIdentity is an Owned Entity and that its properties should be persisted *as though they were individual properties of Samurai. For the sake of this sample, I'm hardcoding the provider in the OnConfiguring method as opposed to leveraging dependency injection and inversion of control (IoC) services. As mentioned in the first article, EF Core can figure out the one-to-one relationship between Samurai and Entrance, but I have to express the relationship in order to access the HasForeignKey method to inform the context about the non-conventional foreign key property, SamuraiFk. In doing so, because Entrance is private in Samurai, I can't use a lambda expression and am using an alternate syntax for the HasForeignKey parameters.

LastModifed is a shadow property—new to EF Core—and will get persisted into the database even though it's not a property in the entities. The Ignore mapping is to ensure that the IsDirty property in Samurai isn't persisted as it's only for domain-relevant logic.

And that's it. Given how much of the DDD patterns I've applied in my domain classes, there's very little in the way of special mappings that I have to add to the SamuraiContext class to inform EF Core 2.0 what the database looks like or how to store and retrieve data from that database. And I'm pretty impressed by that.

There's No Such Thing as a Perfect DDD Sample

This is still a simple example because other than outputting "It's a secret" when a SecretIdentity hasn't been given a value, I'm not solving any complex problems in the logic. The subtitle of Eric Evan's DDD book is "Tackling Complexity in the Heart of Software." So much of the guidance regarding DDD is about breaking down overwhelming complex problems in to smaller solvable problems. The code design patterns are only a piece of that. Everyone has different problems to solve in their domains and, often, readers ask for a sample that can be used as a template for their own software. But all that those of us who share our code and ideas can do is provide examples as learning tools. You can then extrapolate those lessons and apply some of the thinking and decision making to your own problems. I could spend even more time on this tiny bit of code and apply additional logic and patterns from the DDD arsenal, but this sample does go pretty far in leveraging DDD ideas to create a deeper focus on behavior rather than on properties, and further encapsulate and protect the aggregate.

My goal in these two columns was to show how EF Core 2.0 is so much friendlier for mapping your DDD-focused domain model to your database. While I demonstrated that, I hope you were also inspired by the DDD patterns I've included in these classes. ■

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

Thanks to the following Microsoft technical expert for reviewing this article: *Cesar de La Torre*

File Format APIs

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

Try risk free - 30 day trial

Aspose.Total

Enable your applications to manipulate Word, Excel, PDF, PowerPoint, Outlook and more than 100 other file formats for all major platforms.

Aspose.Words

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

Aspose.Cells

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

Aspose.Pdf

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

Aspose.Slides

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

Aspose.Email

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

Aspose.BarCode

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

Aspose.Imaging | Aspose.Tasks | Aspose.OCR | Aspose.Diagram | Aspose.Note | Aspose.HTML

ASPOSE File Format APIs

Download a Free Trial at **https://downloads.aspose.com**

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

How To Be MEAN: Angular Plays Fetch

Welcome back again, MEANers.

In my last column, I wrote, "One of the common needs of an Angular application is to obtain or update data from locations not inside the browser—which is usually everywhere." I then proceeded to go on for another couple thousand words without ever actually doing anything across HTTP.

To be fair, the intention of that column was to prepare the code for being able to fetch data from an HTTP API by placing the HTTP request code inside of an Angular service. Given that I hadn't looked at an Angular service prior to that point, I first needed to work through the basics of what an Angular service looks like and how to use it from the various components that might need it. Alas, all that work left me with no room to get to the actual goal: the data residing in the HTTP API endpoints.

It's time to fix that. Let's go fetch some data.

Speaker API

First things first, I need an HTTP API that I can hit with requests. Somewhere back in that pile of old *MSDN Magazine* issues, there's the last version of the server-side API that I built a year ago. If you actually dig it up, you'll find it's an API that provides "Persons," not speakers. Amazing how a project's goals can "drift" over time, isn't it?

> In Node.js, however, the preferred approach is to hand back not an actual object, but the promise that one will show up—eventually. These are called, not surprisingly, Promises.

Given how simple that MEAN server was, and the fact that the only differences between then and now would be the objects returned, I could easily change the code to push Speaker objects (defined in the last column) back and forth. But doing that means missing out on a golden opportunity to point out something else: the Node.js community has a plethora of "MEAN stack frameworks," including Sails.js (sailsjs.com) and one I've been working with called Loopback (strongloop.com), among others. Many of these frameworks are "built up" off the same tools I used a year ago (Express, MongoDB and the like) to construct a server-side HTTP API, while gluing things together a bit more tightly and painting a lovely veneer over the top. Or you could put together an ASP.NET MVC Web API endpoint. The server-side implementation doesn't matter, as long as it hides behind HTTP.

The first order of business is to see what the interface for the API looks like—what URLs to use, what verbs to use with those URLs, the expected parameters and the format of the returned JSON response. Given that this is a pretty simple service, it's easy to imagine what they will look like, so there's no need to list them out formally here. (Just enter GET on /speakers for the full list of speakers, GET on /speakers/:id to retrieve a particular speaker and so on.)

SpeakerService

If you recall from my last column, the module responsible for fetching data was called SpeakerService. Given that the entire point of the service is to encapsulate away the details of fetching and storing Speakers, you can assume that most (if not all) of the work I'm about to show here will be contained there.

When I left things last time, SpeakerService drew results from an in-memory constant:

```
@Injectable()
export class SpeakerService {
  constructor() { }
  getSpeakers() : Array<Speaker> {
    return SPEAKERS;
  }
  getSpeaker(id: number) : Speaker {
    return SPEAKERS.find(speaker => speaker.id === id);
  }
}
```

The service will eventually need to provide update and deletion facilities, but for now let's just work with the read operations. Notice how in each case, the "get" method returns an immediate object—either the Speaker itself, or an array of Speakers. For an in-memory operation, this is pretty reasonable, but for an operation that will take place through the TCP/IP stack, it's not. Network operations take time, and forcing the caller to wait until the network operation finishes—assuming it ever does—puts an undue burden on the developer building the front-end, because now the front-end has to somehow keep the UI responsive while blocked waiting for the response. Historically, in other UI platforms, the solution has been to have the UI incorporate multiple threads, forcing the UI

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 1 **Modeling the Service with the Promise Placeholder**

```
@Injectable()
export class SpeakerService {
  constructor() { }
  getSpeakers(): Promise<Array<Speaker>> {
    return new Promise((resolve, reject) => {
      setTimeout( () => {
        resolve(SPEAKERS);
      }, 2000);
    });
  }
  getSpeaker(id: number): Promise<Speaker> {
    return new Promise((resolve, reject) => {
      setTimeout( () => {
      resolve(SPEAKERS.find(speaker => speaker.id === id));
      }, 2000);
    });
  }
}
```

developer to account for the discrepancy. In Node.js, however, the preferred approach is to hand back not an actual object, but the promise that one will show up—eventually. These are called, not surprisingly, Promises.

The short concept of a Promise is that it's a placeholder, an object that will, at some point in the future, hold the desired result. But until that happens, execution can continue forward without the actual result. Fundamentally, a Promise is an object wrapped around a callback that will yield the value desired, called a "resolver." (Actually, a Promise is wrapped around two such callbacks, the other being called the "rejecter," in case the Promise can't fulfill its obligations.)

This is the beauty of a Promise—Angular understands that when the value changes, it needs to re-render that part of the DOM that has changed, and does so automatically. Sweet.

Because HTTP traffic is by definition slow, it behooves me to model the service as such, at least until I put in the network traffic. Combining the switch to Promises with a short two-second delay in returning the data suffices nicely. **Figure 1** shows the modified code.

This code simulates a network delay of two seconds before serving up data, but developers who want to more closely simulate network conditions could throw a random failure into the mix, choosing to call reject once every 20 calls or so.

Upvotes and Such

On the component side, the templates and logic must be adjusted slightly to accept Promises instead of the real data. That's actually easier done than said, as shown in **Figure 2**.

Two major changes happened here: First, the speaker field was initialized to an empty Speaker value, so that when Angular first evaluates the component's template, there's always a Speaker object with which to work. Second, the constructor is modified to make use of the returned Promise by providing a then method call, which will take the data handed by the Promise and make use of it. In this case that means saving the constructor into the component's speaker field. The template remains unchanged, and if "ng serve" is still running in the background, reloading the page yields something curious. The page comes up, but the "votes" part of the Upvote component remains empty … for two seconds. Then the fetched value appears, entirely as if by magic.

This is the beauty of a Promise—Angular understands that when the value changes, it needs to re-render that part of the DOM that has changed, and does so automatically. Sweet.

Speaker, O Speaker, Where Art Thou, Speaker?

But I'm still fetching data locally. How boring. Time to go back to the SpeakerService and start doing calls over HTTP. This means making a few changes: importing the HttpModule (the Angular team chose to break it out of "core"), referencing the HttpService, and making use of it in the SpeakerService. Note that (not surprisingly) the HttpService makes use of Promises, so the subtle shift in the SpeakerService API makes using the HttpService pretty straightforward, as **Figure 3** illustrates.

The reason for the call to toPromise will become more clear in a future piece, when I discuss Observables. For now, this call is

Figure 2 **Working with Speaker Values**

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'SpeakerApp';
  speaker: Speaker = new Speaker();

  constructor(private speakerSvc: SpeakerService) {
    this.speakerSvc.getSpeaker(1).then( (data) => {
      this.speaker = data;
    });
  }
}
```

Figure 3 **Using the HttpService**

```
@Injectable()
export class SpeakerService {
  private speakersUrl = "http://localhost:3000/api/Speakers";

  constructor(private http : Http) { }
  getSpeaker(id: number): Promise<Speaker> {
    return this.http.get(`${this.speakersUrl}/${id}`)
                .toPromise()
                .then(response => {
                   let raw: any = response.json();
                   let speaker = new Speaker();
                   speaker.firstName = raw.FirstName;
                   speaker.lastName = raw.LastName;
                   speaker.votes = new Upvote(raw.Votes);
                   speaker.id = raw.id;
                   return speaker;
                });
  }
```

necessary to convert the returned object for use as a Promise. Note that to make the code in **Figure 3** work, one more import is necessary. To bring in the toPromise method, make sure to add the following code to the top of the file:

```
import 'rxjs/add/operator/toPromise';
```

It may seem odd that the SpeakerService is "transforming" the returned JSON from the API into a local Speaker object, but this is actually a fairly common task. The JSON published by an API is often not quite the right "shape" for use by the UI, and so it needs to be massaged into the local concept of what that type should look like. In this particular case, I'm assuming that the service is returning the speakers with uppercase field names (less unusual than you might think), and taking the additional step to transform the "Votes" field into an Upvote object. While some "full stack" systems will try to insist there's one schema definition that can be shared across both client and server, it's generally safer to assume that there isn't, and perform "adjustments" to the data received to transform it into something the UI finds more palatable.

In some cases, that "adjustment" consists of leaving some amount of data on the floor, or "flattening" it for easier use, or some other transformation. For example, if the server sends back distinct "Speaker" and "Talk" objects, the SpeakerService may wish to "flatten" that into the single Speaker object that I'm using, collecting all of the votes for each talk in the speaker's repertoire into a single scalar value.

Go ahead and save. Assuming you have a server running on localhost at port 3000, you should see everything work as it did before the flip to using HTTP. (If you don't have a server running on localhost, feel free to change the URL in the SpeakerService to http://msdn-mean.azurewebsites.net/, which is where I've parked the newly refurbished Speaker API. Feel free to browse the complete set of API endpoints using the "/explorer" URL there, if you wish—that is a Swagger-generated UI, which comes out of the box when working with Loopback.)

Wrapping Up

The HttpModule (and its corresponding service object, HTTP) certainly has support for PUTs and POSTs and all the other goodness that HTTP offers, but those will be addressed once I've added the ability to edit and add new Speakers in the Angular app. For that matter, I still need to flesh out this front end a bit more, by showing a list of all the speakers and then allowing the user to drill down into a specific one—the classic "master-detail" approach that so many applications use (and abuse). To do that, I'll need to talk about how Angular manages "routing" (to move between components, as if they were "pages" in the application) and "forms" (for doing data entry). But all that has to wait for next time. Until then … 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 a ton of articles, authored and co-authored 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 who reviewed this article: *James Bender*

dtSearch®

Instantly Search Terabytes of Data

across a desktop, network, Internet or Intranet site with dtSearch enterprise and developer products

Over 25 search features, with easy multicolor hit-highlighting options

dtSearch's document filters support popular file types, emails with multilevel attachments, databases, web data

Developers:

- APIs for .NET, Java and C++
- SDKs for Windows, UWP, Linux, Mac and Android
- See dtSearch.com for articles on faceted search, advanced data classification, working with SQL, NoSQL & other DBs, MS Azure, etc.

Visit dtSearch.com for

- hundreds of reviews and case studies
- fully-functional evaluations

The Smart Choice for Text Retrieval® since 1991

dtSearch.com 1-800-IT-FINDS

FRANK LA VIGNE

Exploring Azure Machine Learning Studio

Longtime readers of my blog (franksworld.com) have noticed over the past 18 months a marked shift in content toward data science, artificial intelligence (AI) and machine learning (ML). So, it's only fitting that this column also shifts gears and focuses on the revolution happening all around us: The AI Revolution. Not too long ago, AI was the stuff of science fiction. Now we can add intelligence to virtually any app or Web site. In fact, many of your favorite apps and Web sites already employ some form of AI. Cortana and many other voice assistants are obvious examples of AI in the UI layer. Less obvious, but no less important, are intelligent algorithms optimizing resources, recommendation systems telling you what movies you might like and determining what you see in your social media feeds.

Over the last 10 years, the focus of many developer and IT organizations was the capture and storage of Big Data. During that time, the notion of what a "large" database size was grew in orders of magnitude from terabytes to petabytes. Now, in 2017, the rush is on to find insights, trends and predictions of the future based on the information buried in these large data stores. Combined with recent advancements in AI research, cloud-based analytics tools and ML algorithms, these large data stores can not only be mined, but monetized.

With the cloud providing affordable computing power and storage, even small businesses can predict the future by anticipating customer behavior and identifying trends at the individual level and at scale. Organizations that can discover and deploy actionable predictive models before the competition does will dominate their market segment. Properly leveraged, AI can add serious value to any business. As Peter Drucker put it, "The best way to predict the future is to create it." In that spirit, here's a deeper look at AI and ML.

Figure 1 **Generally Accepted Artificial Intelligence Phrases**

Term	Description
Data Science	Interdisciplinary field that applies scientific methods and processes in order to extract insights from data.
Artificial Intelligence	Computer systems able to perform tasks that have traditionally required human intelligence, such as computer vision, speech recognition and more.
Machine Learning	A type of artificial intelligence that allows software to predict outcomes or classify data without explicitly being programmed.
Binary Classification	The process of classifying data into one of two groups. For example, determining if a flight will be delayed or on time.
Multi-Class Classification	The process of classifying data into one of three or more groups. For example, determining if a flight will be delayed, canceled or on schedule.
Regression	The process of determining the output of one value based on a number of other values. For example, determining how much a flight will be delayed based on weather, day of week, carrier and so on.
R	An open source programming language used in statistical processing.
Python	An interpreted multi-paradigm language often used in the field of data science.

Getting the Terms Right

Before getting into an AI project, it's important to define the scope of what exactly is "artificial intelligence." This will be important as future columns will rely on a common set of meaning for terms associated with this field. A quick Internet search of the term "artificial intelligence" yields a lot of various results, from chatbots and computer vision systems to debates on the nature of consciousness itself. While there's no firm consensus as to what the term means, most experts agree generally on basic phrases, which are listed in **Figure 1**.

The Power of the Cloud in the Palm of Your Hand

Around the middle of the last decade, I was an early adopter and proponent of the Tablet PC platform. As I would deliver presentations to various user groups and speak at conferences, one criticism would inevitably come up: lack of high-performance hardware. The reason for the lack of serious computing power on these devices had more to do with the constraints of making a tablet device viable: namely weight, battery life and cost. Many would object that while they admired the tablet PC form factor, they needed a device with a more powerful CPU. Fast forward 10 years and the limitations of cost, battery life and network connectivity have largely gone away. Any device within range of Wi-Fi and 4G networks can now connect to limitless computing services and storage resources in the cloud.

AI in the Cloud

As a developer, you have choices in terms of what types of intelligent services to consume. If an app or Web site requires image recognition or natural language processing, then Microsoft has made several services available as part of the Microsoft Cognitive Services, a set of APIs, SDKs and services that expands on Microsoft's evolving portfolio of ML APIs. They enable you to easily add intelligent features—such as

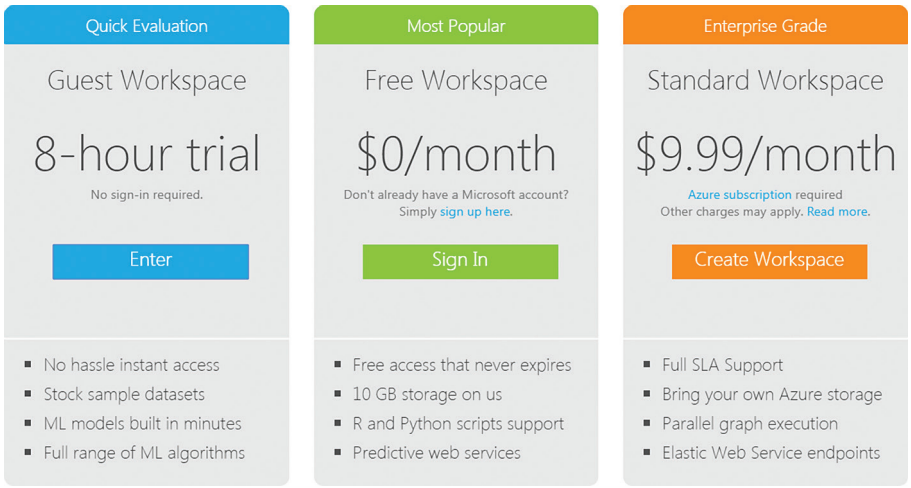


Figure 2 **Pricing Tiers of Azure Machine Learning Studio**

emotion and video detection; facial, speech and vision recognition; and speech and language understanding—to your applications. Microsoft's vision is for more personal computing experiences and enhanced productivity aided by systems that increasingly can see, hear, speak, understand and even begin to reason.

These services expose models that have been trained on millions of sample images. In a video on Channel9, Microsoft's Anna Roth briefly explains the process of training the algorithms with a wide variety of sample data (bit.ly/2x7u1D4). The models exposed by the many Cognitive Services APIs were trained over years and millions of data points by dozens of engineers and researchers. That's why they're so good at what they do. For when your app or Web site requires a solution that one of the Cognitive Services APIs can resolve, use them. The list of Microsoft Cognitive Services offerings continues to grow. For an updated list, go to bit.ly/2vGWcuN.

However, when your data has a more limited scope inside a specific domain, then you'll have to create your own models from your own data. While that might sound intimidating or impractical, the process is quite straightforward with another cloud service provided by Microsoft: Azure ML Studio.

Azure ML Studio

Azure ML Studio is an online service that makes ML and building predictive models approachable and straightforward. For the most part, there's no code involved. Users drag around various modules representing actions and algorithms in an interface resembling Visio. For maximum flexibility and extensibility, there are modules for inserting R and Python code for cases where the built-in models don't suffice or for using existing code.

Getting Started Open a browser and head over to studio.azureml.net. If you've never used Azure ML Studio before, click on the Sign Up button. You may choose either the Guest Workspace or Free Workspace option in the dialog that follows (see **Figure 2**). For the purposes of this article, I recommend using the free workspace, as you'll have the chance to save your projects and expose your models via Web services.

If you already have a Microsoft account, click on Sign In under the Free Workspace option. If this is the first time you've logged into Azure ML Studio, you'll see an empty list of experiments. As ML is considered a subset of data science, the term "experiment" is used here.

Creating an Experiment The best way to examine the power of Azure ML Studio is to start with a sample experiment. Fortunately, there are a number of pre-built samples provided by Microsoft. First, click on the New button on the lower-left corner of the browser window. In the resulting dialog, type flight into the textbox. The screen should look similar to **Figure 3**. Clicking on the View in Gallery link will bring up a page detailing information about the experiment (bit.ly/2i9Q61i). Move the mouse over the tile and click the Open in Studio button to open the experiment to start working on it.

This experiment runs what's known as a Binary Classification, which means the ML algorithm will place each record in the dataset into one of two categories. In this case, whether or not the flight will be delayed.

Once the experiment loads your screen will look something like **Figure 4**.

While this might look intimidating at first, what's going on is actually quite simple. Zoom in using the scroll wheel on the mouse or using the zoom controller on the lower left of the workspace canvas.

Navigating the Workspace Canvas Azure ML Studio has built-in navigation controls to explore and manipulate the view of the workspace canvas. The navigation controls from left to right are: a Mini Map of the canvas, zoom slider control, zoom to actual

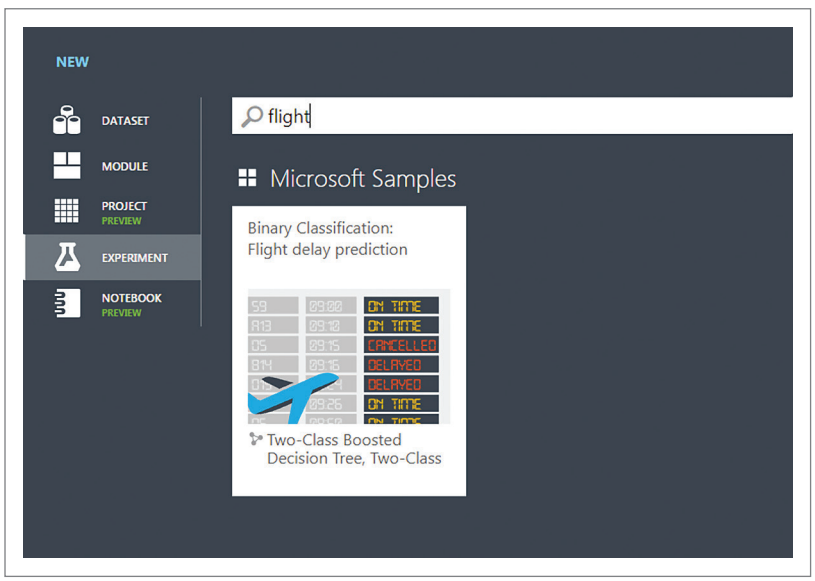


Figure 3 **The Flight Delay Prediction Sample Experiment**

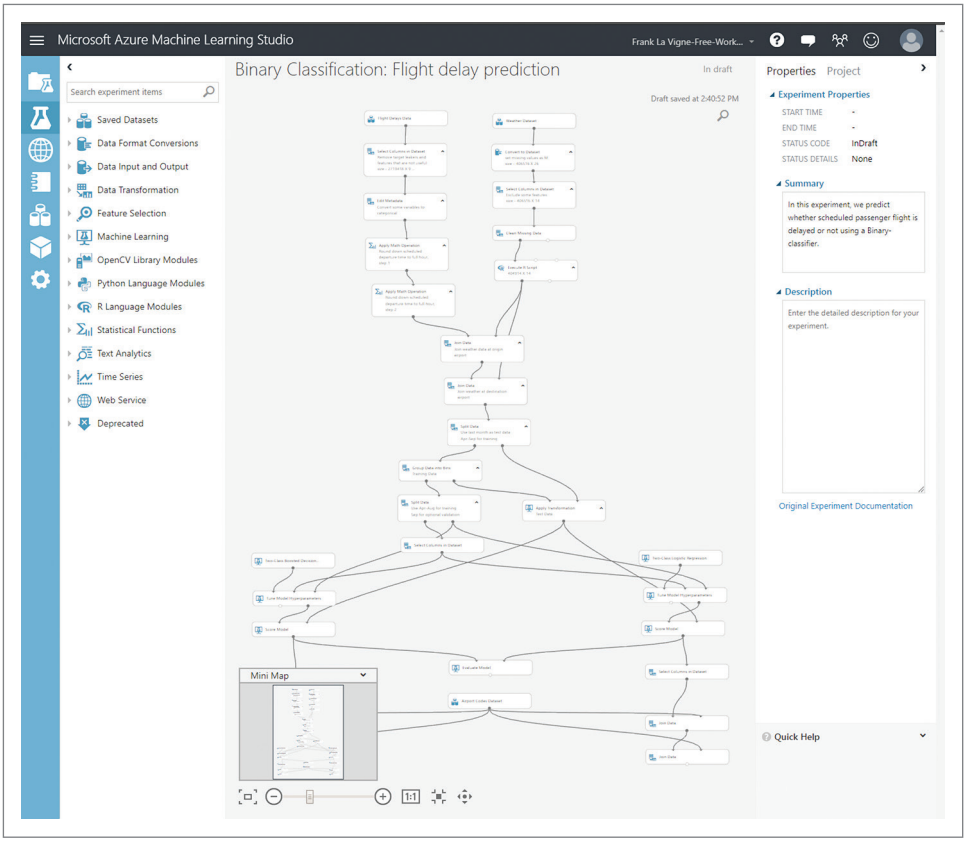


Figure 4 **The Flight Delay Experiment Opened in Azure Machine Learning Studio**

size button, a zoom to fit button and a pan toggle button. You may have already noticed that clicking and dragging around the canvas selects modules and does not move the canvas around. Clicking the pan toggle button will toggle the mode from selecting to panning. When pan mode is activated, the button appears blue.

Modules The workspace canvas contains modules linked together. Each module represents either a data set, manipulation of data or an algorithm. To get an idea of the contents of the source data set, select the Flight Delays Data module, right-click on the 1, and click Visualize on the context menu (see **Figure 5**).

In the resulting dialog, the contents of the data set appear in a grid. Click on one of the fields and expand the Statistics and Visualization panels. In **Figure 6**, the Carrier field is chosen and between the Statistics and Visualization panel, the basic shape of the data can be discovered. Click the X in the upper-right corner of the dialog to close out this view.

Repeat the previous steps to visualize the structure and content of the Weather Dataset.

Manipulating the Raw Data Sets Note that there are a number of modules making modifications to the data in the data set and there are two branches: one for the Flight Delays data set and the other for the Weather data set. The data in each of the data sets needs to be cleaned in order to be merged and analyzed by the ML algorithm. Notice that in the steps attached to the Weather data set, there's even a module that executes R code. Select the Execute R Script module and, as before, right mouse click on the 1. The context menu has a Visualize option, yet it's grayed out, as is every other option. This means that the experiment hasn't been run. On the lower portion of the screen, click the Run button and choose Run to run the experiment. In a few moments, the experiment will finish. Depending on server load, this experiment may take longer when using the free service. Now click on the Execute R Script and right-click the 1. The visualization dialog appears displaying the output of the module. In fact, now that the entire experiment has run, every module's data can be visualized. Exploring the modules and visualizing the data at each step, you can track the data transformations throughout the process. However, some modules' visualizations appear different than others.

Machine Learning As mentioned previously, this experiment classifies flights into one of two categories: delayed or not delayed. The experiment first cleans the data and shapes the data into a format and structure with which an ML algorithm can work. Data scientists often refer to this process as "data wrangling" and it can represent the majority of effort in any kind of data science project.

Generally speaking, the process of an ML experiment once the data has been shaped and cleaned follows the following steps: split the data into a test set and training set, pick an algorithm to examine the data, and score the results. This experiment runs the data through two algorithms: Two-Class Boosted Decision Tree and Two-Class Logistic Regression. Each algorithm processes the data in different ways. Certain algorithms are better at certain data sets and problems than others. This is where the experimentation comes into play.

When there's more than one algorithm in an experiment, then the models can be evaluated against one another with an Evaluate Model module. Select the Evaluate Model module, right-click on 1 and select Visualize in the context menu. The dialog will look something like **Figure 7**.

The Evaluate Model visualization dialog contains vital information to understanding the performance of the

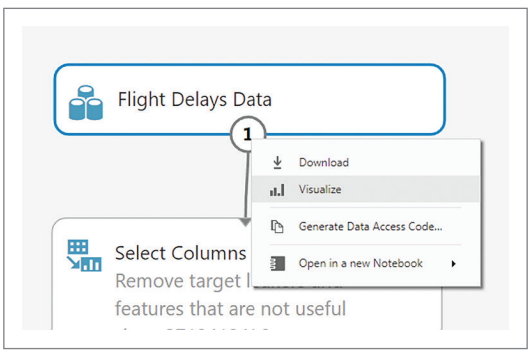


Figure 5 **Flight Delays Data Module Context Menu**

BEST SELLER

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

LEADTOOLS
THE WORLD LEADER IN IMAGING SDKS

Powerful DICOM, PACS, and HL7 functionality.

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

BEST SELLER

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

BEST SELLER

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

ASPOSE
File Format APIs

Create, Edit & Manipulate Word, Excel, PDF and PowerPoint Files in your .NET apps.

- Aspose.Total for .NET contains every Aspose .NET API in one complete package
- Also works with MS Visio, MS OneNote, MS Project, Email, Images, Barcodes, OCR & OMR
- Mail Merge, Document Assembly, Text Extraction, PST & OST Creation and File Conversion
- Create and Recognize Barcodes, Control Text Formatting, Paragraphs, Images and Tables
- Now includes the new Aspose.CAD and Aspose.3D APIs

BEST SELLER

DevExpress DXperience 17.1 | 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

© 1996-2017 ComponentSource. All Rights Reserved. All prices correct at the time of press. Online prices may vary from those shown due to daily fluctuations & online discounts.

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

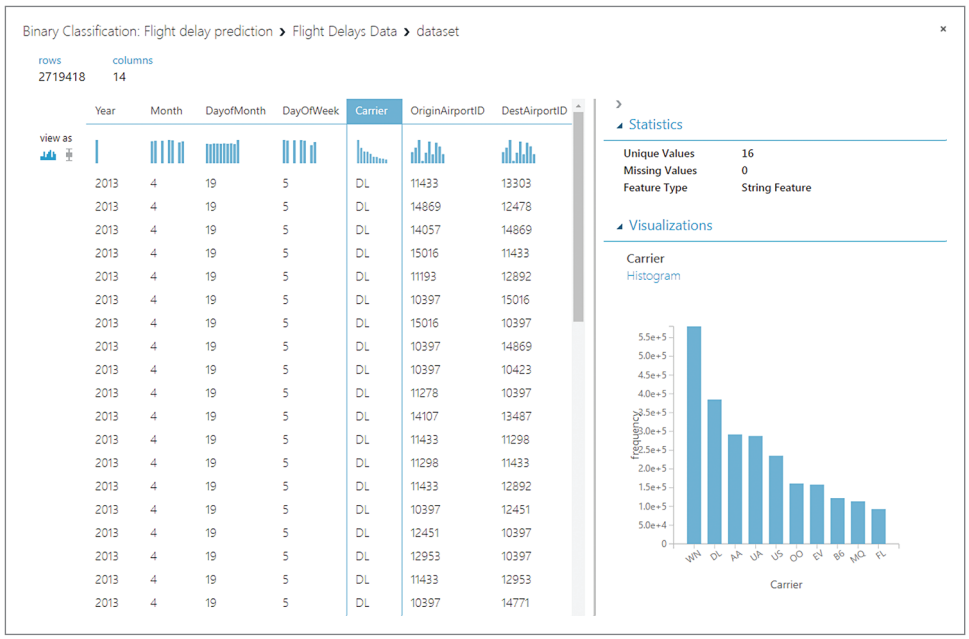


Figure 6 **Visualizing the Raw Data**

ML models just created. The blue line represents the model created via the Two-Class Boosted Decision Tree algorithm and the red line represents the model created by the Two-Class Logistic Regression algorithm. The blue model, selected by default, has an accuracy rating of 0.806, meaning it was correct 80.6 percent of the time. Click on the red square in the chart legend to see the results from the Two-Class Logistic Regression model. Its accuracy was slightly better at 81.7 percent. Also note, the Matrix of Confusion numbers on both models. A matrix of confusion is a measure of the quality of a classification model. It measures the number of times a record was correctly flagged positively or negatively, as well as how often the model was wrong with "false positives" and "false negatives."

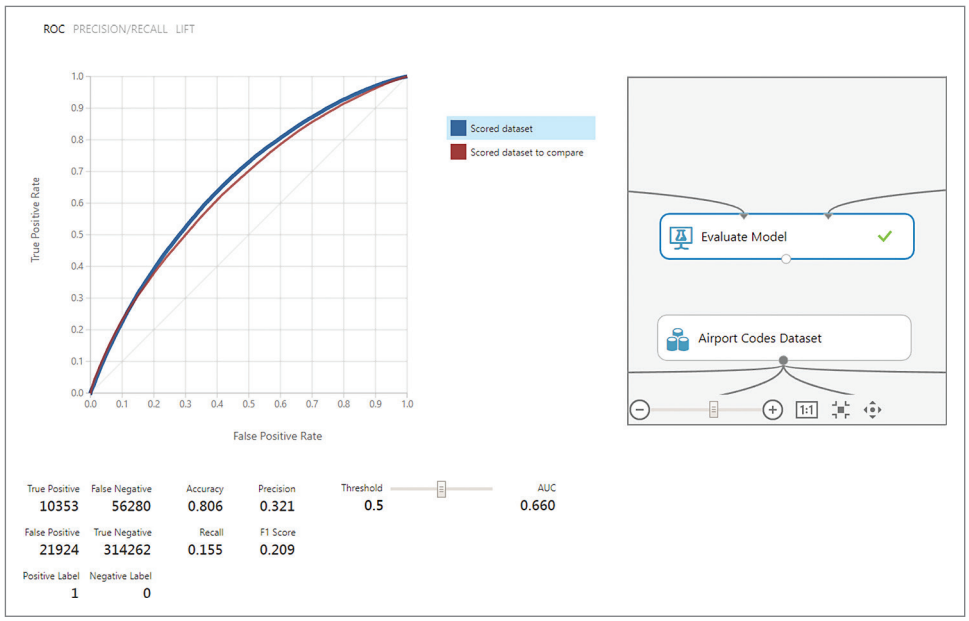


Figure 7 **The Evaluate Model Visualization Dialog**

The main graphical feature on this screen is the Receiver Operating Characteristic, or ROC, curve. A full explanation of this metric warrants an article of its own. More information about this metric can be found on Wikipedia at bit.ly/2fPKJnf. Assuming that random guessing would be correct about half of the time, the ROC curve would display a model with 50 percent accuracy as a straight line at a 45-degree angle. Given that both models are greater than 80 percent, the algorithm performs significantly better than random guessing. In other words, the computer has learned to predict outcomes with a fair bit of accuracy.

More Data Manipulation The remainder of the workflow involves trimming down the number of columns in the dataset from 31 to six. In order to make the data more readable, the fields OriginAirportID and DestAirportID are joined to a table with city, state, and airport names. That way 12264 becomes more readable as Washington Dulles International.

Wrapping Up

Some of the terms that Azure ML Studio uses are related to statistics and are generally outside the usual vocabulary of most developers. In truth, that's where the bulk of the learning curve of Azure ML Studio lies, in learning the jargon of data science.

I've barely scratched the surface of what can be built with Azure ML Studio. The next step will be making this predictive model accessible to Web sites and apps using the built-in support for Web services. In future columns, I'll explore other aspects of AI both inside and outside Azure ML Studio.

While machine learning, artificial intelligence, and data science in general might seem intimidating to the average developer or data engineer, the overall goal of this column is to help you discover that quite the opposite is true. ■

Frank La Vigne *is a data scientist at Wintellect and co-host of the DataDriven podcast. He blogs regularly at FranksWorld.com and you can watch him on his YouTube channel, "Frank's World TV" (FranksWorld.TV).*

Thanks to the following Microsoft technical experts for reviewing this article: *Rachel Appel and Andy Leonard*

RD

Rider

New .NET IDE

Cross-platform.
Killer code analysis.
Great for refactoring.

From the makers of ReSharper,
IntelliJ IDEA, and WebStorm.

Learn more
and download
jetbrains.com/rider

JET
BRAINS

From Algorithms to Coroutines in C++

Kenny Kerr

There's a C++ Standard Library algorithm called iota that has always intrigued me. It has a curious name and an interesting function. The word iota is the name of a letter in the Greek alphabet. It's commonly used in English to mean a very small amount and often the negative, not the least amount, derived from a quote in the New Testament Book of Matthew. This idea of a very small amount speaks to the function of the iota algorithm. It's meant to fill a range with values that increase by a small amount, as the initial value is stored and then incremented until the range has been filled. Something like this:

```
#include <numeric>

int main()
{
  int range[10];
  // Range: Random missile launch codes

  std::iota(std::begin(range), std::end(range), 0);
  // Range: { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }
}
```

It's often said that C++ developers should expunge all naked for loops and replace them with algorithms. Certainly, the iota algorithm qualifies as it takes the place of the for loop that any C or C++ developer has undoubtedly written thousands of times. You can imagine what your C++ Standard Library implementation might look like:

```
namespace std
{
  template <typename Iterator, typename Type>
  void iota(Iterator first, Iterator last, Type value)
  {
    for (; first != last; ++first, ++value)
    {
      *first = value;
    }
  }
}
```

So, yeah, you don't want to be caught in a code review with code like that. Unless you're a library developer, of course. It's great that the iota algorithm saves me from having to write that for loop, but you know what? I've never actually used it in production. The story usually goes something like this: I need a range of values. This is such a fundamental thing in computer science that there must be a standard algorithm for it. I again scour the list over at bit.ly/2i5WZRc and I find iota. Hmm, it needs a range to fill with values. OK, what's the cheapest range I can find … I then print the values out to make sure I got it right using … a for loop:

```
#include <numeric>
#include <stdio.h>

int main()
{
  int range[10];
  std::iota(std::begin(range), std::end(range), 0);

  for (int i : range)
  {
    printf("%d\n", i);
  }
}
```

To be honest, the only thing I like about this code is the range-based for loop. The problem is that I simply don't need nor want that range. I don't want to have to create some container just to hold

This article discusses:

- Introduction to the iota algorithm
- The built-in range function in Python
- Attempts to avoid raw loops
- Using coroutines to write algorithms

Technologies discussed:

C++, Algorithms, Python, Iterators, Generators, Coroutines

Code download available at:

godbolt.org/g/NXHBZR

the values so that I can iterate over them. What if I need a lot more values? I'd much rather just write the for loop myself:

```
#include <stdio.h>

int main()
{
  for (int i = 0; i != 10; ++i)
  {
    printf("%d\n", i);
  }
}
```

To add insult to injury, this involves a lot less typing. It sure would be nice, however, if there were an iota-like function that could somehow generate a range of values for a range-based for loop to consume without having to use a container. I was recently browsing a book about the Python language and noticed that it has a built-in function called range. I can write the same program in Python like this:

```
for i in range(0, 10):
  print(i)
```

Be careful with that indentation. It's how the Python language represents compound statements. I read that Python was named after a certain British comedy rather than the nonvenomous snake. I don't think the author was kidding. Still, I love the succinct nature of this code. Surely, I can achieve something along these lines in C++. Indeed, this is what I wish the iota algorithm would provide but, alas. Essentially, what I'm looking for is a range algorithm that looks something like this:

```
template <typename T>
generator<T> range(T first, T last)
{
  return{ ... };
}

int main()
{
  for (int i : range(0, 10))
  {
    printf("%d\n", i);
  }
}
```

To my knowledge, no such function exists, so let's go and build it. The first step is to approximate the algorithm with something reliable that can act as a baseline for testing. The C++ standard vector container comes in handy in such cases:

```
#include <vector>

template <typename T>
std::vector<T> range(T first, T last)
{
  std::vector<T> values;

  while (first != last)
  {
    values.push_back(first++);
  }

  return values;
}
```

It also does a good job of illustrating why you don't want to build a container in the first place, or even figure out how large it should be, for that matter. Why should there even be a cap? Still, this is useful because you can easily compare the output of this range generator to a more efficient alternative. Well, it turns out that writing a more efficient generator isn't that difficult. Have a look at **Figure 1**.

The range function simply creates a generator initialized with the same pair of bounding values. The generator can then use those values to produce lightweight iterators via the conventional begin and end member functions. The most tedious part is spitting out the largely boilerplate iterator implementation. The iterator can simply hold a given value and increment it as needed. It must also provide a set of type aliases to describe itself to standard algorithms. This isn't strictly necessary for the simple range-based for loop, but it pays to include this as a bit of future-proofing:

```
template <typename T>
struct generator
{
  struct iterator
  {
    T value;

    using iterator_category = std::input_iterator_tag;
    using value_type = T;
    using difference_type = ptrdiff_t;
    using pointer = T const*;
    using reference = T const&;
```

Incrementing the iterator can simply increment the underlying value. The post-increment form can safely be deleted:

```
iterator& operator++()
{
  ++value;
  return *this;
}

iterator operator++(int) = delete;
```

The other equally important function provided by an iterator is that of comparison. A range-based for loop will use this to determine whether it has reached the end of the range:

```
bool operator==(iterator const& other) const
{
  return value == other.value;
}

bool operator!=(iterator const& other) const
{
  return !(*this == other);
}
```

Finally, a range-based for loop will want to dereference the iterator to return the current value in the range. I could delete the member call operator, because it isn't needed for the range-based for loop, but that would needlessly limit the utility of generators to be used by other algorithms:

Figure 1 **A Classical Generator**

```
template <typename T>
struct generator
{
  T first;
  T last;

  struct iterator{ ... };

  iterator begin()
  {
    return{ first };
  }

  iterator end()
  {
    return{ last };
  }
};

template <typename T>
generator<T> range(T first, T last)
{
  return{ first, last };
}
```

```
T const& operator*() const
{
  return value;
}

T const* operator->() const
{
  return std::addressof(value);
}
```

It might be that the generator and associated range function are used with number-like objects rather than simple primitives. In that case, you might also want to use the addressof helper, should the number-like object be playing tricks with operator& overloading. And that's all it takes. My range function now works as expected:

```
template <typename T>
generator<T> range(T first, T last)
{
  return{ first, last };
}

int main()
{
  for (int i : range(0, 10))
  {
    printf("%d\n", i);
  }
}
```

Of course, this isn't particularly flexible. I've produced the iota of my dreams, but it's still just an iota of what would be possible if I switched gears and embraced coroutines. You see, with coroutines you can write all kinds of generators far more succinctly and without having to write a new generator class template for each kind of range you'd like to produce. Imagine if you only had to write one more generator and then have an assortment of range-like functions to produce different sequences on demand. That's what coroutines enable. Instead of embedding the knowledge of the original iota generation into the generator, you can embed that knowledge directly inside the range function and have a single generator class template that provides the glue between producer and consumer. Let's do it.

I begin by including the coroutine header, which provides the definition of the coroutine_handle class template:

```
#include <experimental/coroutine>
```

I'll use the coroutine_handle to allow the generator to interact with the state machine represented by a coroutine. This will query and resume as needed to allow a range-based for loop—or any other loop, for that matter—to direct the progress of the coroutine producing a pull- rather than push-model of data consumption. The generator is in some ways similar to that of the classical generator in **Figure 1**. The big difference is that rather than updating values directly, it merely nudges the coroutine forward. **Figure 2** provides the outline.

So, there's a little more going on here. Not only is there an iterator that allows the range-based for loop to interact with the generator from the outside, but there's also a promise_type that allows the coroutine to interact with the generator from the inside. First, some housekeeping: Recall that the function generating values won't be returning a generator directly, but rather allow a developer to use co_yield statements to forward values from the coroutine, through the generator, and to the call site. Consider the simplest of generators:

```
generator<int> one_two_three()
{
  co_yield 1;
  co_yield 2;
  co_yield 3;
}
```

Notice how the developer never explicitly creates the coroutine return type. That's the role of the C++ compiler as it stitches together the state machine represented by this code. Essentially, the C++ compiler looks for the promise_type and uses that to construct a logical coroutine frame. Don't worry, the coroutine frame will likely disappear after the C++ compiler is done optimizing the code in some cases. Anyway, the promise_type is then used to initialize the generator that gets returned to the caller. Given the promise_type, I can get the handle representing the coroutine so that the generator can drive it from the outside in:

```
generator(promise_type& promise) :
  handle(handle_type::from_promise(promise))
{
}
```

Of course, the coroutine_handle is a pretty low-level construct and I don't want a developer holding onto a generator to accidentally corrupt the state machine inside of an active coroutine. The solution is simply to implement move semantics and prohibit copies. Something like this (first, I'll give it a default constructor and expressly delete the special copy members):

```
generator() = default;
generator(generator const&) = delete;
generator &operator=(generator const&) = delete;
```

And then I'll implement move semantics simply by transferring the coroutine's handle value so that two generators never point to the same running coroutine, as shown in **Figure 3**.

Figure 2 **A Coroutine Generator**

```
template <typename T>
struct generator
{
  struct promise_type{ ... };

  using handle_type = std::experimental::coroutine_handle<promise_type>;

  handle_type handle{ nullptr };

  struct iterator{ ... };

  iterator begin()
  {
    ...
    handle.resume();
    ...
  }

  iterator end()
  {
    return nullptr;
  }
};
```

Figure 3 **Implementing Move Semantics**

```
generator(generator&& other) : handle(other.handle)
{
  other.handle = nullptr;
}

generator &operator=(generator&& other)
{
  if (this != &other)
  {
    handle = other.handle;
    other.handle = nullptr;
  }

  return *this;
}
```

Now, given the fact that the coroutine is being driven from the outside, it's important to remember that the generator also has the responsibility of destroying the coroutine:

```
~generator()
{
  if (handle)
  {
    handle.destroy();
  }
}
```

This actually has more to do with the result of final_suspend on the promise_type, but I'll save that for another day. That's enough bookkeeping for now. Let's now look at the generator's promise_type. The promise_type is a convenient place to park any state such that it will be included in any allocation made for the coroutine frame by the C++ compiler. The generator is then just a lightweight object that can easily move around and refer back to that state as needed. There are only two pieces of information that I really need to convey from within the coroutine back out to the caller. The first is the value to yield and the second is any exception that might have been thrown:

```
#include <variant>

template <typename T>
struct generator
{
  struct promise_type
  {
    std::variant<T const*, std::exception_ptr> value;
```

Although optional, I tend to wrap exception_ptr objects inside std::optional because the implementation of exception_ptr in Visual C++ is a little expensive. Even an empty exception_ptr calls into the CRT during both construction and destruction. Wrapping it inside optional neatly avoids that overhead. A more precise state model is to use a variant, as I just illustrated, to hold either the current value or the exception_ptr because they're mutually exclusive. The current value is merely a pointer to the value being yielded inside the coroutine. This is safe to do because the coroutine will be suspended while the value is read and whatever temporary object may be yielded up will be safely preserved while the value is being observed outside of the coroutine.

When a coroutine initially returns to its caller, it asks the promise_type to produce the return value. Because the generator can be constructed by giving a reference to the promise_type, I can simply return that reference here:

```
promise_type& get_return_object()
{
  return *this;
}
```

A coroutine producing a generator isn't your typical concurrency-enabling coroutine and it's often just the generator that dictates the lifetime and execution of the coroutine. As such, I indicate to the C++ compiler that the coroutine must be initially suspended so that the generator can control stepping through the coroutine, so to speak:

```
std::experimental::suspend_always initial_suspend()
{
  return {};
}
```

Likewise, I indicate that the coroutine will be suspended upon return, rather than having the coroutine destroy itself automatically:

```
std::experimental::suspend_always final_suspend()
{
  return {};
}
```

Figure 4 **The Inspirational Range Generator**

```
template <typename T>
generator<int> range(T first, T last)
{
  while (first != last)
  {
    co_yield first++;
  }
}

int main()
{
  for (int i : range(0, 10))
  {
    printf("%d\n", i);
  }
}
```

This ensures that I can still query the state of the coroutine, via the promise_type allocated within the coroutine frame, after the coroutine completes. This is essential to allow me to read the exception_ptr upon failure, or even just to know that the coroutine is done. If the coroutine automatically destroys itself when it completes, I wouldn't even be able to query the coroutine_handle, let alone the promise_type, following a call to resume the coroutine at its final suspension point. Capturing the value to yield is now quite straightforward:

```
std::experimental::suspend_always yield_value(T const& other)
{
  value = std::addressof(other);
  return {};
}
```

I simply use the handy addressof helper again. A promise_type must also provide a return_void or return_value function. Even though it isn't used in this example, it hints at the fact that co_yield is really just an abstraction over co_await:

```
void return_void()
{
}
```

More on that later. Next, I'll add a little defense against misuse just to make it easier for the developer to figure out what went wrong. You see, a generator yielding values implies that unless the coroutine completes, a value is available to be read. If a coroutine were to include a co_await expression, then it could conceivably suspend without a value being present and there would be no way to convey this fact to the caller. For that reason, I prevent a developer from writing a co_await statement, as follows:

```
template <typename Expression>
Expression&& await_transform(Expression&& expression)
{
  static_assert(sizeof(expression) == 0,
    "co_await is not supported in coroutines of type generator");
  return std::forward<Expression>(expression);
}
```

Wrapping up the promise_type, I just need to take care of catching, so to speak, any exception that might have been thrown. The C++ compiler will ensure that the promise_type's unhandled_exception member is called:

```
void unhandled_exception()
{
  value = std::current_exception();
}
```

I can then, just as a convenience to the implementation, provide a handy function for optionally rethrowing the exception in the appropriate context:

```
void rethrow_if_failed()
{
  if (value.index() == 1)
  {
    std::rethrow_exception(std::get<1>(value));
  }
}
```

Enough about the promise_type. I now have a functioning generator—but I'll just add a simple iterator so that I can easily drive it from a range-based for loop. As before, the iterator will have the boilerplate type aliases to describe itself to standard algorithms. However, the iterator simply holds on to the coroutine_handle:

```
struct iterator
{
  using iterator_category = std::input_iterator_tag;
  using value_type = T;
  using difference_type = ptrdiff_t;
  using pointer = T const*;
  using reference = T const&;

  handle_type handle;
```

Incrementing the iterator is a little more involved than the simpler iota iterator as this is the primary point at which the generator interacts with the coroutine. Incrementing the iterator implies that the iterator is valid and may in fact be incremented. Because the "end" iterator holds a nullptr handle, I can simply provide an iterator comparison, as follows:

```
bool operator==(iterator const& other) const
{
  return handle == other.handle;
}

bool operator!=(iterator const& other) const
{
  return !(*this == other);
}
```

Assuming it's a valid iterator, I first resume the coroutine, allowing it to execute and yield up its next value. I then need to check whether this execution brought the coroutine to an end, and if so, propagate any exception that might have been raised inside the coroutine:

```
iterator &operator++()
{
  handle.resume();

  if (handle.done())
  {
    promise_type& promise = handle.promise();
    handle = nullptr;
    promise.rethrow_if_failed();
  }

  return *this;
}

iterator operator++(int) = delete;
```

Otherwise, the iterator is considered to have reached its end and its handle is simply cleared such that it will compare successfully against the end iterator. Care needs to be taken to clear the coroutine handle prior to throwing any uncaught exception to prevent anyone from accidentally resuming the coroutine at the final suspension point, as this would lead to undefined behavior. The generator's begin member function performs much the same logic, to ensure that I can consistently propagate any exception that's thrown prior to reaching the first suspension point:

```
iterator begin()
{
  if (!handle)
  {
    return nullptr;
  }

  handle.resume();

  if (handle.done())
  {
    handle.promise().rethrow_if_failed();
    return nullptr;
  }

  return handle;
}
```

The main difference is that begin is a member of the generator, which owns the coroutine handle, and therefore must not clear the coroutine handle. Finally, and quite simply, I can implement iterator dereferencing simply by returning a reference to the current value stored within the promise_type:

```
T const& operator*() const
{
  return *std::get<0>(handle.promise().value);
}

T const* operator->() const
{
  return std::addressof(operator*());
}
```

And I'm done. I can now write all manner of algorithms, producing a variety of generated sequences using this generalized generator. **Figure 4** shows what the inspirational range generator looks like.

Who needs a limited range, anyway? As I now have a pull model, I can simply have the caller decide when they've had enough, as you can see in **Figure 5**.

The possibilities are endless! There is, of course, more to generators and coroutines and I've only just scratched the surface here. Join me next time for more on coroutines in C++. You can find the complete example from this article over on Compiler Explorer: godbolt.org/g/NXHBZR. ■

Figure 5 **A Limitless Generator**

```
template <typename T>
generator<int> range(T first)
{
  while (true)
  {
    co_yield first++;
  }
}

int main()
{
  for (int i : range(0))
  {
    printf("%d\n", i);

    if (...)
    {
      break;
    }
  }
}
```

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

Thanks to the following technical expert for reviewing this article: *Gor Nishanov*

DocuVieware 3

HTML5 Viewer & Document Management Kit

NEW RELEASE

Easy integration

Full support for custom snap-in

Zero-footprint solution

Fully customizable UI

Mobile devices optimization

Fast & crystal-clear rendering

Check the **New Features** and the **Online Demos**

DOWNLOAD YOUR FREE TRIAL www.docuvieware.com

Powered by GdPicture.NET 14

DocuVieware is an ORPALIS imaging technologies product

Write a Windows Device Portal Packaged Plug-in

Scott Jones

Amid the fanfare of the Windows 10 release, one feature made a quiet debut: Windows Device Portal (WDP). WDP is a Web-based diagnostic system built into Windows 10. After a progressive roll-out, WDP is now available on HoloLens, IoT, Mobile, Desktop and Xbox. It's expected to be included in new editions of Windows as they're released. With the exception of Windows Desktop, WDP is immediately available out of the box. On Desktop, WDP is available with the download and installation of the optional Windows Update Developer Mode package. In this article, I'll show you how to use the WDP API to implement a WDP packaged plug-in to extend your Windows Store app with custom REST APIs. While the focus of this article is on the Windows Desktop, the concepts and techniques also apply to other Windows editions.

To create and execute a WDP packaged plug-in, you'll need to update your system to at least the Windows 10 Creators Update (10.0.15063.0), and install the corresponding Windows 10 SDK (bit.ly/2tx5Bnk). For detailed instructions, see the article, "Updating Your Tooling for Windows 10 Creators Update" (bit.ly/2tx3FeD). It might be necessary to restart the machine for Visual Studio to detect and support the new SDK; otherwise projects may fail to load.

If you're not familiar with WDP, I encourage you to read the article, "Windows Device Portal Overview" (bit.ly/2uG9rco). This will provide an introduction to the features of WDP, and will also ensure that you have it installed before proceeding with writing a packaged plug-in.

Briefly, you'll need to download and install the optional Developer Mode package. From the Settings app (press Windows+I), navigate to the Update & security | For developers page, and select the Developer mode radio button. When the Developer Mode package installation is complete, check the Enable Device Portal checkbox, set credentials as desired, and browse to the supplied URL to verify functionality. After accepting the WDP self-signed certificate and entering credentials, the browser should display the WDP Web UI, as shown in **Figure 1**.

This article discusses:

- WDP architecture
- Writing a packaged plug-in
- Testing and deploying a packaged plug-in

Technologies discussed:

Windows Device Portal, C#, Universal Windows Platform, WinRT, HTTP

Code download available at:

bit.ly/2tTPFMV

WDP Architecture

The tools presented in the WDP UI in **Figure 1** are implemented with JavaScript controls that communicate with REST APIs hosted in the WDP Service. As REST APIs, these generalized, stateless Web requests are applicable in more contexts than just the WDP UI. For example, a WDP REST API could be called from a Windows PowerShell script or a custom user agent. The

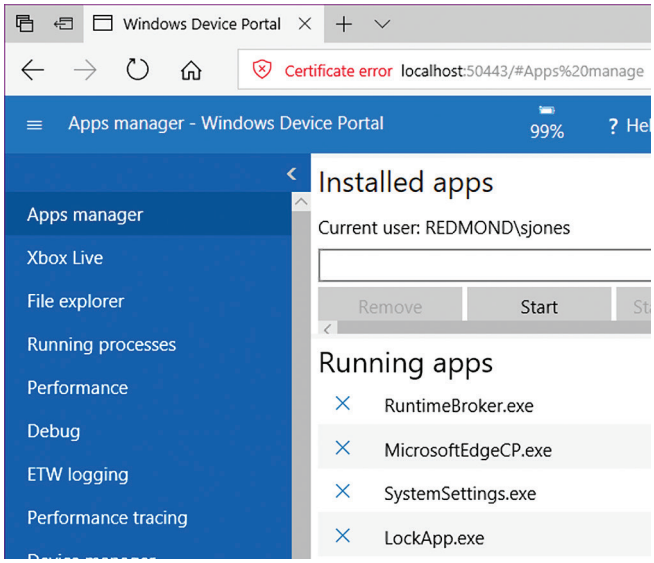


Figure 1 **Windows Device Portal Web UI**

WindowsDevicePortalWrapper (bit.ly/2tx2NXF) open source project offers a library for developing custom WDP user agents in C#. In this article, I'll use the browser and the free command-line utility curl (curl.haxx.se) to exercise my custom REST APIs.

WDP was designed with extensibility in mind. For example, WDP is customized for each edition of Windows via built-in plug-ins. With the Windows 10 Creators Update, it's now possible for third parties to extend WDP by creating packaged plug-ins. A packaged plug-in provides custom REST endpoints, with an optional Web-based UI, implemented and deployed within a Windows Store app. **Figure 2** illustrates how the system works.

Readers familiar with the internals of Microsoft's IIS will recognize the design of WDP. As with IIS, WDP is built upon the HTTP Server API (also referred to as HTTP.SYS), dividing responsibilities between the HTTP Controller and HTTP Worker roles. The WDP service implements the controller, running under the local SYSTEM account. Each WDP packaged plug-in implements a worker, running within the AppContainer sandbox in the user's security context.

It's possible to host a Web server from scratch within a store app, using the HTTP Server API. However, implementing a WDP packaged plug-in offers several advantages. WDP provides encryption, authentication, and security services for all content and REST APIs, including those in packaged plug-ins. This includes protection from cross-site request forgery and cross-site WebSocket hijacking attacks. Additionally, WDP manages the lifespan of each packaged plug-in, which may execute from either a foreground UI-visible app or as a background task. In short, it's simpler and safer to implement REST APIs with a packaged plug-in. The sequence diagram in **Figure 3** illustrates the flow of execution for a packaged plug-in.

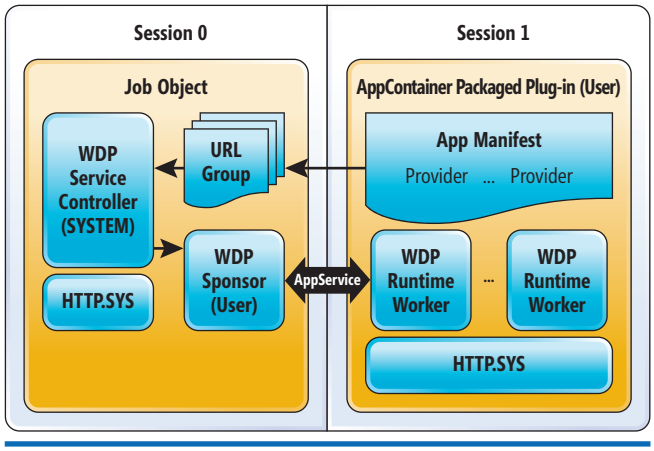


Figure 2 **Windows Device Portal Architecture**

In its package app manifest, each plug-in identifies itself with a windows.devicePortalProvider app extension and associated app service, and declares its routes (URLs) of interest. A plug-in can register either a content route, a REST API route or both. At package install time, the manifest data is registered with the system.

At startup, the WDP service scans the system for registered WDP packaged plug-ins, as identified by the windows.devicePortalProvider app extension. For each plug-in discovered, the WDP service reads the package manifest for route information. The set of routes requested by the packaged plug-in, referred to as the URLGroup, is registered with HTTP.SYS to create an on-demand HTTP request queue. The WDP service then monitors each packaged plug-in request queue for incoming requests.

At the first route request for a plug-in, the WDP service launches a WDP sponsor in the user's security context. The WDP sponsor in turn activates the packaged plug-in, transferring the HTTP request queue to the plug-in. The WDP service activates and communicates with the packaged plug-in via the app service described in the manifest. The app service connection functions like a named pipe, with the WDP sponsor acting as the client of this connection, and the packaged plug-in acting as the server. This sponsorship design ensures that long-running requests aren't interrupted by the system's resource management policies. The WDP runtime then begins servicing requests on behalf of the plug-in. Content requests are serviced automatically, while REST API requests are dispatched to the packaged plug-in's registered RequestReceived event handler. Plug-in lifetime is managed by both the WDP service and the Process Lifetime Manager (PLM). For details on managing plug-in state when a background task is suspended, see the article, "Launching, Resuming and Background Tasks" (bit.ly/2u0N7f0). A job object further ensures that WDP service rundown is complete, terminating any running WDP sponsors and their associated packaged plug-ins.

Writing a Packaged Plug-in

Creating the Project Before creating your packaged plug-in, you'll need to decide whether your Web request handler can run as a background task or whether it must run in the process of a foreground app. Foreground execution is necessary if, for example, the handler requires access to the app's internal data structures. This flexibility in execution is enabled by the underlying AppService, which can be configured for either background or foreground operation. For a more thorough discussion of AppServices, please consult the articles, "Create and Consume an App Service" (bit.ly/2uZsSfz), and, "Convert an App Service to Run in the Same Process as Its Host App" (bit.ly/2u0G8n1).

In this example, I'll implement both a foreground handler and a background handler, and demonstrate integrating your static content and REST APIs. Begin by creating a new solution in Visual Studio with the Blank App (Universal Windows) C# project template. Name the solution MyPackagedPlugin and the project MyApp. The app will host the foreground handler. When prompted, target at least the Creator's Update SDK (15063), to ensure availability of the WDP API. Next, add a runtime component library to the solution, using the Windows Runtime Component Visual C# template. Name this project MyComponent. This library will host the background handler.

To ensure that the component is included in the app package, add a reference to it in the app project. In Solution Explorer, expand the app project node, right-click the References node, and pick Add Reference. In the Reference Manager dialog, expand the Projects node, select Solution and check the MyComponent project.

Before moving on, set the solution platform to match your machine's architecture. This is a requirement for a packaged plug-in, as WoW64 is not supported. Note that in this case, I'm deploying to the local machine, but the advice also applies when deploying to a secondary target device.

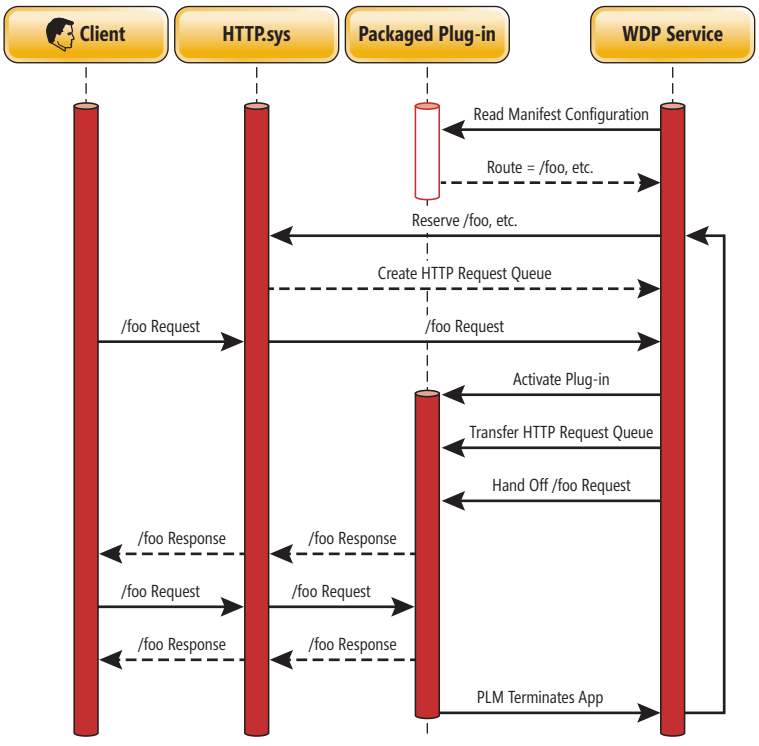


Figure 3 **Activation and Execution of a Packaged Plug-in**

Editing the Manifest Because there are a number of manifest changes necessary, I'll edit the Package.appxmanifest file directly, rather than using the designer. In Solution Explorer, under the app node, right-click on the Package.appxmanifest file node, and pick View Code to edit the XML.

Begin by adding the uap4 and rescap namespace declarations and aliases that will be necessary for subsequent elements:

```
<Package ...
  xmlns:uap4="http://schemas.microsoft.com/appx/manifest/uap/windows10/4"
  xmlns:rescap="http://schemas.microsoft.com/appx/manifest/foundation
    /windows10/restrictedcapabilities"
  IgnorableNamespaces="... uap4 rescap">
```

To make the package more discoverable during debugging, I'll change the package name from a generated GUID to something meaningful:

```
<Identity Name="MyPackagedPlugin" Publisher="CN=msdn" Version="1.0.0.0" />
```

Next, I'll add the app extensions required for each packaged plug-in handler to the package\applications\application\extensions element, as shown in **Figure 4**.

Each handler requires two extensions. The AppService extension provides the activation mechanism and communication channel between the WDP service and the WDP runtime hosting the handler. By convention, AppServices uses a reverse domain name scheme to ensure uniqueness. If implementing a background AppService, the EntryPoint attribute is required and specifies where execution begins. If implementing a foreground AppService, execution begins with the app's OnBackgroundActivated method and the EntryPoint attribute is omitted.

The DevicePortalProvider extension provides configuration data for the DevicePortalConnection that hosts the handler. A DevicePortalProvider represents the client side of the AppService connection, providing URL handlers to the DevicePortalConnection. The AppServiceName attribute must correspond with the Name attribute of the AppService element (for example, com.contoso.www.myapp). The DevicePortalProvider element may specify a ContentRoute for serving static Web content; a HandlerRoute for dispatching requests to a REST API handler; or both. Both ContentRoute and HandlerRoute must be unique. If either route conflicts with a built-in WDP route, or with a previously registered packaged plug-in route, the plug-in will fail to load, presenting an appropriate diagnostic message. Additionally, the ContentRoute relative URL must map to a relative path under the package install folder (such as \myapp\www). For more details, see the DevicePortalProvider Extension Spec (bit.ly/2u1aqG8).

Finally, I'll add the capabilities required for my packaged plug-in:

```
<Capabilities>
  <Capability Name="privateNetworkClientServer" />
  <rescap:Capability Name="devicePortalProvider" />
</Capabilities>
```

The privateNetworkClientServer capability enables HTTP.SYS functionality within an AppContainer. In this demo, I'll be deploying the package directly from Visual Studio onto the local machine for execution. However, to onboard your app to the store, you'll also need to obtain the devicePortalProvider capability, which is restricted to Microsoft partners. For more information, please refer to the article, "App Capability Declarations" (bit.ly/2u7gHkt). These capabilities are the minimum required by the WDP runtime to host a packaged plug-in. Your plug-in's handler code may require additional capabilities, depending on the Universal Windows Platform APIs it calls.

Adding Static Content Next, let's create a status page for the plug-in. The page, and any other static content it references, should be placed in an app-relative path corresponding to the content route reserved for the plug-in, in this case, /myapp/www.

In Solution Explorer, right-click on your app project node and select Add | New Folder. Name the folder myapp. Right-click on the newly added folder node and again select Add | New Folder to create a subfolder named www. Press Ctrl+N, and within the File New dialog, select the General | HTML Page template. Save this file as index.html, under the solution's MyPackagedPlugin\MyApp\myapp\www path. Then add this file to the project folder path, so that it's included as package content. Right-click on the newly added index.html file node and select Properties. Confirm the default property values of Build Action: Content and Copy to Output Directory: Do not copy.

Figure 4 **Adding App Extensions**

```
<Package ...><Applications><Application ...>

  <Extensions>

    <!--Foreground (in app) packaged plug-in handler app service and WDP provider-->
    <uap:Extension
      Category="windows.appService">
      <uap:AppService Name="com.contoso.www.myapp" />
    </uap:Extension>
    <uap4:Extension
      Category="windows.devicePortalProvider">
      <uap4:DevicePortalProvider
        DisplayName="MyPluginApp"
        AppServiceName="com.contoso.www.myapp"
        ContentRoute="/myapp/www/"
        HandlerRoute="/myapp/API/" />
    </uap4:Extension>

    <!--Background packaged plug-in handler app service and WDP provider-->
    <uap:Extension
      Category="windows.appService"
      EntryPoint="MyComponent.MyBackgroundHandler">
      <uap:AppService Name="com.contoso.www.mycomponent" />
    </uap:Extension>
    <uap4:Extension
      Category="windows.devicePortalProvider">
      <uap4:DevicePortalProvider
        DisplayName="MyPluginComponent"
        AppServiceName="com.contoso.www.mycomponent"
        HandlerRoute="/mycomponent/API/" />
    </uap4:Extension>

  </Extensions>

</Application></Applications></Package>
```

Figure 5 **Status Page Markup**

```
<html>
<head>
  <title>My Packaged Plug-in Page</title>
  <script src="/js/rest.js"></script>
  <script src="/js/jquery.js"></script>
  <script type="text/javascript">
    function InitPage() {
      var webb = new WebbRest();
      webb.httpGetExpect200("/myapp/API/status")
        .done(function (response) {
          $('#app_status')[0].innerText = response.status;
        });
      webb.httpGetExpect200("/mycomponent/API/status")
        .done(function (response) {
          $('#comp_status')[0].innerText = response.status;
        });
    }
  </script>
</head>
<body onload="InitPage();">
  <div style="font-size: x-large;">
    My Packaged Plug-in Page
    <br/><br/>
    App Status: <span id="app_status" style="color:green"></span>
    <br/>
    Component Status: <span id="comp_status" style="color:green"></span>
  </div>
</body>
</html>
```

Now add the markup shown in **Figure 5** to the newly created index.html. This page demonstrates several things. First, note that built-in WDP resources, such as the jquery.js and rest.js libraries, are available to packaged plug-ins, as well. This enables a packaged plug-in to combine domain-specific functionality with built-in WDP functionality. For more information, please consult the article, "Device Portal API Reference" (bit.ly/2uFTTFD). Second, note the reference to the plug-in's REST API calls, both the app's /myapp/API/status and the component's /mycomponent/API/status. This shows how a plug-in's static and dynamic content can be easily combined.

Adding a REST API Now I'll implement a REST API handler. As mentioned earlier, choice of entry point depends on whether a background or foreground AppService is being implemented. Beyond a minor difference in how the incoming AppServiceConnection is obtained, the DevicePortalConnection implementation is identical for both scenarios. I'll start with the app's foreground handler.

Open the source file App.xaml.cs and add the following namespaces, required for every handler:

```
using Windows.ApplicationModel.AppService;
using Windows.ApplicationModel.Background;
using Windows.System.Diagnostics.DevicePortal;
using Windows.Web.Http;
using Windows.Web.Http.Headers;
```

Inside the MyApp.App class definition, I'll add a few members to implement the handler's state. BackgroundTaskDeferral defers completion of the background task that hosts execution of the handler, and DevicePortalConnection implements the connection to the WDP service itself.

```
sealed partial class App : Application
{
  private BackgroundTaskDeferral taskDeferral;
  private DevicePortalConnection devicePortalConnection;
  private static Uri statusUri = new Uri("/myapp/API/status", UriKind.Relative);
```

Next, override the background activation handler for the app, to instantiate a DevicePortalConnection and subscribe to its events, as shown in **Figure 6**.

For this handler, I'll simply look for requests to the URI /myapp/API/status, and respond with a JSON structure, as shown in **Figure 7**. A more complex implementation could support several routes, distinguish between GETs and POSTs, inspect URI parameters, and so on.

Finally, I handle closing of the DevicePortalConnection, either directly with its Closed event, or indirectly by cancellation of the background task, as shown in **Figure 8**.

Implementation of the component project's background handler is nearly identical to the foreground example. Open the source file Class1.cs and add the required namespaces, as earlier. Inside the component namespace, replace the generated class definition with a class implementing IBackgroundTask and its required Run method:

```
public sealed class MyBackgroundHandler : IBackgroundTask
{
  public void Run(IBackgroundTaskInstance taskInstance)
  {
    // Implement as for foreground handler's OnBackgroundActivated...
  }
}
```

The Run method takes an IBackgroundTaskInstance parameter value directly. Recall that the foreground handler obtained this value from the TaskInstance member of the BackgroundActivatedEventArgs parameter of OnBackgroundActivated. Aside from this difference in obtaining the taskInstance, the implementation is identical for both handlers. Copy the remaining implementation from your foreground handler's App.xaml.cs file.

Testing a Packaged Plug-in

Deploying the Plug-in To test your plug-in, you must first deploy it, either via Visual Studio F5 deployment or by using loose .appx files (generated via the Visual Studio Project | Store | Create App Packages menu). Let's use the former. Right-click on the MyApp project node and select Properties. In the project property sheet, select the Debug tab and then enable the Do not launch checkbox.

I suggest setting a breakpoint in your RequestReceived event handler, and possibly in your plug-in's entry point—either your BackgroundActivated event handler or Run method override. This will confirm that your plug-in is properly configured and successfully activating when you test it in the next section. Now, press F5 to deploy and debug your app.

Running WDP in Debug Mode After deploying your package, WDP must be restarted to detect it. (In case you skipped the WDP introduction mentioned earlier, you may also need to install the Developer Mode package now to ensure WDP is available). This can be done by toggling the Enable Device Portal checkbox from the Settings app's Update & security | For developers page. However, I'm going to instead run the WDP Service in debug mode as a console app. This will allow me to see WDP trace output, which will help when troubleshooting packaged plug-in configuration and execution issues. The WDP service is configured to execute under the SYSTEM account, and this is also required when running WDP in debug mode. The PsExec utility, available as part of the PsTools suite (bit.ly/2tXiDf4), will help do that.

First, create a SYSTEM console, running PsExec from within an Administrator console:

```
> psexec -d -i -s -accepteula cmd /k title SYSTEM Console - Careful!
```

This command will launch a second console window running in the SYSTEM account context. Normally, this window uses the legacy console, and new console features, such as text wrap on resize, text selection enhancements and clipboard shortcuts, aren't available. If you want to enable these features when running as SYSTEM, save the following to a registry script file (for example, EnableNewConsoleForSystem.reg) and execute it:

```
Windows Registry Editor Version 5.00

[HKEY_USERS\.DEFAULT\Console]
"ForceV2"=dword:00000001
```

In the SYSTEM console, run WDP in debug mode, enabling both clear and encrypted requests:

```
> webmanagement -debug -clearandssl
```

You should see output similar to that in **Figure 9**. To safely terminate WDP in debug mode, simply press Ctrl+C. This will maintain your SYSTEM console so you can iterate on plug-in development.

Note that the trace output uses various terms: webmanagement, WebB and WDP. These exist for historical reasons and refer to different subsystems, but the distinctions can be ignored. Also note that executing in debug mode uses a different configuration from executing as a service. For example, in debug mode, default authorization is disabled by default. This avoids the need to enter credentials and prevents automatic HTTPS redirection, simplifying testing from a browser or command-line utility. Also, the ports have been assigned values 54321 and 54684. Normally, the desktop WDP service uses ports 50080 and 50443 (although if these are taken, random ports will be dynamically assigned). This permits WDP to execute in debug mode without interfering with the production mode of WDP running as a service. However,

Figure 6 **Implementing the DevicePortalConnection**

```
// Implement background task handler with a DevicePortalConnection
protected override void OnBackgroundActivated(BackgroundActivatedEventArgs args)
{
  // Take a deferral to allow the background task to continue executing
  var taskInstance = args.TaskInstance;
  this.taskDeferral = taskInstance.GetDeferral();
  taskInstance.Canceled += TaskInstance_Canceled;

  // Create a DevicePortal client from an AppServiceConnection
  var details = taskInstance.TriggerDetails as AppServiceTriggerDetails;
  var appServiceConnection = details.AppServiceConnection;
  this.devicePortalConnection =
    DevicePortalConnection.GetForAppServiceConnection(
      appServiceConnection);

  // Add handlers for RequestReceived and Closed events
  devicePortalConnection.RequestReceived += DevicePortalConnection_RequestReceived;
  devicePortalConnection.Closed += DevicePortalConnection_Closed;
}
```

Figure 7 **Handling the Request**

```
// RequestReceived handler demonstrating response construction, based on request
private void DevicePortalConnection_RequestReceived(
  DevicePortalConnection sender,
  DevicePortalConnectionRequestReceivedEventArgs args)
{
  if (args.RequestMessage.RequestUri.AbsolutePath.ToString() ==
    statusUri.ToString())
  {
    args.ResponseMessage.StatusCode = HttpStatusCode.Ok;
    args.ResponseMessage.Content =
      new HttpStringContent("{ \"status\": \"good\" }");
    args.ResponseMessage.Content.Headers.ContentType =
      new HttpMediaTypeHeaderValue("application/json");
  }
  else
  {
    args.ResponseMessage.StatusCode = HttpStatusCode.NotFound;
  }
}
```

Figure 8 **Closing the DevicePortalConnection**

```
// Complete the deferral if task is canceled or DevicePortal connection closed
private void Close()
{
  this.devicePortalConnection = null;
  this.taskDeferral.Complete();
}

private void TaskInstance_Canceled(IBackgroundTaskInstance sender,
  BackgroundTaskCancellationReason reason)
{
  Close();
}

private void DevicePortalConnection_Closed(DevicePortalConnection sender,
  DevicePortalConnectionClosedEventArgs args)
{
  Close();
}
```

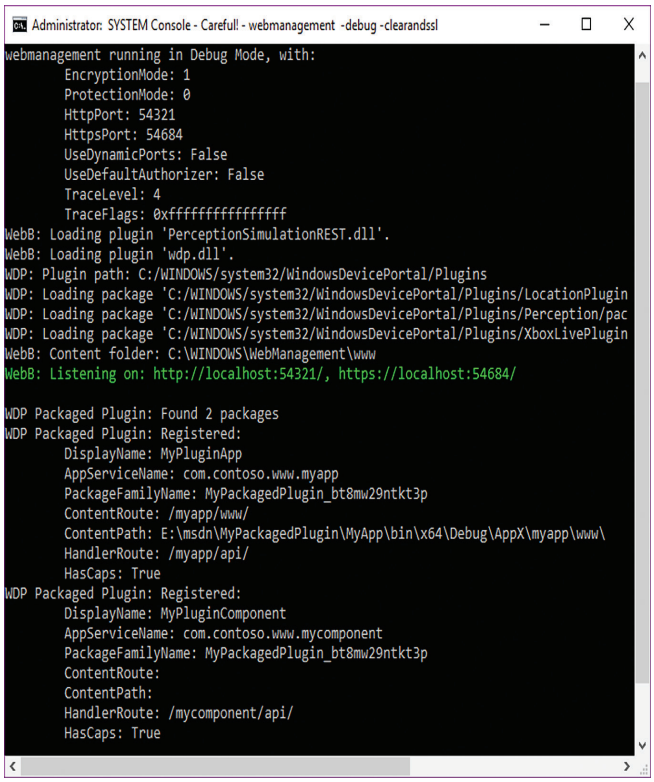


Figure 9 **Windows Device Portal in Debug Mode**

it may become irksome to switch port numbers on URLs when testing, based on the WDP mode of execution. If so, you can use the -httpPort and -httpsPort command-line options to explicitly set the port numbers to match production mode. In this case, you'll need to ensure that the WDP service is turned off to prevent conflicts. To see all WDP command-line options, type:

```
> WebManagement.exe -?
```

As indicated in the trace output, several built-in plug-ins are automatically loaded as part of WDP startup. WDP then reports the discovery of the deployed packaged plug-in with "Found 2 packages" (more precisely, one package with two providers). The first provider describes the foreground handler, and confirms reservation of the static ContentRoute URL and its corresponding mapped package file path and REST HandlerRoute. At this point, WDP has created an on-demand HTTP request queue to service these routes and is awaiting requests. The second provider describes the background handler, which specifies only a REST HandlerRoute. It, too, is ready for business.

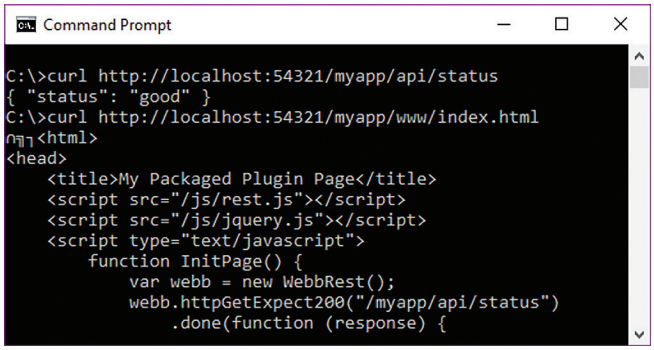


Figure 10 **Testing with Curl**

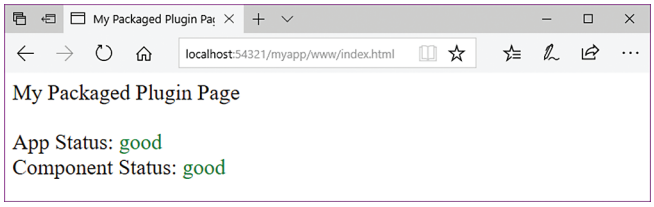


Figure 11 **Testing with the Browser**

Executing the Plug-in I recommend developing and testing your plug-in with the excellent command-line utility I mentioned earlier, curl. For simpler scenarios, a browser is adequate, but curl affords fine-grained control over headers, authentication and so on. This makes it easier to use when WDP encryption, authentication and CSRF protection options are enabled. In these situations, the curl "—verbose" option is also useful for troubleshooting.

Figure 10 demonstrates requesting the foreground handler's resources: its status REST API and Web page.

Figure 11 shows using the browser to request the app's status page, in turn demonstrating the embedded REST API calls.

When any requests are made to your plug-in's routes, the breakpoint you set in your entry point should hit in the debugger. And for REST API requests specifically, the breakpoint in your RequestReceived event handler should hit. You should also see confirmation in the WDP diagnostic output that the packaged plug-in was activated.

Troubleshooting It's worth noting a few of the common errors that prevent correct execution of a handler:

- Manifest mismatches, such as AppService name or entry point
- Neglecting to add a background handler component project reference to the app
- Deploying with an architecture mismatch between the package and the service
- Implementing IBackgroundTask in your app project, rather than as a Windows Runtime Component

When running in debug mode, WDP will provide specific diagnostic messages for these errors, where possible. After three failed attempts to activate a plug-in, WDP will disable the plug-in for that session as indicated with the diagnostic:

```
WDP Packaged Plugin: Disabling request queue after 3 failed attempts
```

After making corrections to your package manifest or code, it's usually necessary to restart WDP, both to reset the disabled counter and to rescan for package configuration changes.

Wrapping Up

The Windows Device Portal API offers a safe, simple mechanism for extending your Windows Store app's diagnostics capabilities with packaged plug-ins. Future enhancements to this API will enable integration of packaged plug-ins within the WDP UI, introduce REST APIs for installing and controlling packaged plug-ins, and expand the applications possible for packaged plug-ins. ■

Scott Jones *works for Microsoft as a software engineer on the Application Platform and Tools team.*

Thanks to the following Microsoft technical expert for reviewing this article: *Hirsch Singhal*

WE ARE CHANG
THE WAY YOU LO
REPORTING

Microsoft .NET
WINDOWS FORMS

WPF

ASP.NET MVC

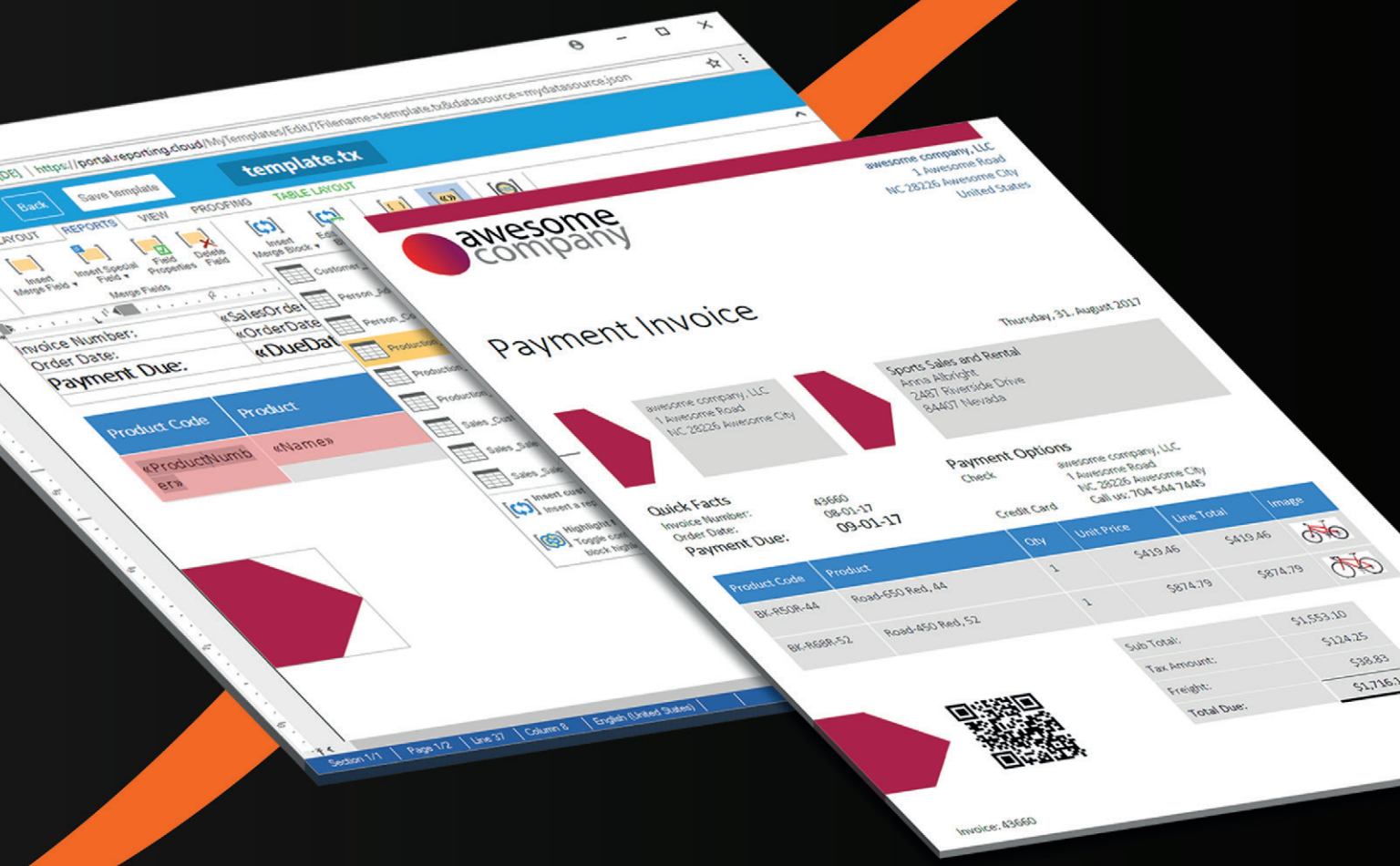
ASP.NET AJAX

CLOUD WEB API

www.textcontrol.com

TEXT CONTROL

GING
OOK AT



© 2017 Text Control GmbH. All rights reserved. All other product and brand names are trademarks and/or registered trademarks of their respective owners.

Visual Studio
Partner

Building Better Cloud Deployments: 5 Steps to Immutability

Martin Albisetti

Cloud delivery has emerged as the accepted standard for Web service deployment, whether those services reside on public or private cloud infrastructures. While cloud promises on-demand availability, automation and agility, not all Web services are designed to take full advantage of these promises. Unpredictable cloud infrastructure and agile DevOps cycles have created a need for isolated software components, which enable rapid movement through testing, development and production of Web services in the cloud.

At Bitnami, we've been living and breathing the evolution of software deployment for a very long time, and we've invested heavily in making it easy for developers to embrace the best deployment techniques available. One technique is the concept of immutable infrastructure, which is an approach to managing services and software deployments in a way that components are replaced rather than changed. In effect, an application or service is redeployed each time any change occurs.

This article discusses:

- Isolating layers of code infrastructure, including OS, runtime framework and application code
- Adapting workflows to automate and improve immutable image deployment
- Optimizing for stateless scenarios such as worker farms, caching systems or static Web sites

Technologies discussed:

Cloud Services, Configuration Management Software

Pets Versus Cattle

Randy Bias, founder of the OpenStack Foundation, coined the analogy of pets versus cattle to represent this. Traditional servers are treated like pets. They have their own personalities, and if they get sick you nurse them back to health. Cattle, on the other hand, all look the same. If they get sick you just get rid of them. It costs you more to treat cattle than to replace them. Also, cattle scales. A farmer will add to the herd as the operation grows, based on demand and budget.

Clouds provide just enough functionality to let you continue treating your servers as pets. The problem is, they're pet goldfish. It's boring. If the pet gets sick, you might be able to treat it, but vets will usually laugh at you if you try. The fish will often be unresponsive and will die at any time without warning.

This is why immutable (or image-based) deployments and using public or private clouds usually go together. If you want to confidently run production-grade Web services in the cloud, it's time to set aside pet ownership and deploy infrastructure and software that can be easily reproduced or replaced. Like cattle.

To understand the subtleties in this approach, let's consider the pros and cons of using clouds. Clouds offer benefits that used to be prohibitively expensive to achieve with private infrastructure. In addition to worldwide redundancy and fast delivery by serving content geographically close to users, cloud deployments enable elasticity. IT organizations can call upon resources as they're needed, paying only for what they use at any time. This on-demand model also makes it easy to experiment with new technologies and topologies without steep, up-front investments.

There's also the benefit that comes with the "Everything"-as-a-Service model. Clouds offer most of the common services you'd want in modern Web software, including databases (SQL and NoSQL), HTTP front ends, load balancers, DNS, block storage, object storage, log analysis, monitoring, queues, AI and more. The list grows with each month. And clouds let you focus on your core business, by offloading common services to a provider.

Of course, everything comes at a cost. Adopting on-demand infrastructure means you forego reliability guarantees. An instance in the cloud could disappear or reboot at any time without warning. Likewise, failure modes can be impossible to reproduce. Erratic behavior can also occur. For example, a new instance might fail to boot or servers might experience slow disk IO or spiking CPU loads, with no clear cause.

Above the Clouds

Given the advantages and challenges of cloud deployments, immutable services are a perfect fit. Among the key benefits:

- Predictably deploy exactly the same bytes everywhere in the world.
- Launch dozens, hundreds or even thousands of new instances in response to demand, with little or no preparation.
- Build automatic error recovery into deployments. Misbehaving instances can be automatically shut down or removed from serving requests and left for later inspection and new ones brought up, that way debugging can happen whenever convenient instead of in the middle of an outage.
- Automate deployments by employing images that don't have to be transformed after boot.

Beyond working well with clouds, there are many benefits to deploying your services immutably, from reduced complexity and speedier deployments to improved security and compliance.

Reduced Complexity If you have a basic set of three services, each with an average of four instances, you quickly end up with 12 pets. That's a lot of pets to feed! You might argue that the four instances are all the same, and that's probably how it started off, but you can see how over the period of six months each instance will have been upgraded, debugged and revived in different ways, no longer guaranteeing that they're actually the same anymore.

By contrast, an immutable approach means that you have exactly one image. Each of the four instances will contain the exact same code at deployment time, configured in the exact same way. Unless you change something afterward, you only have three different variations of the images to understand and worry about. Additionally, any drift that occurs while in production will be reset on each subsequent deployment.

Speedier Deployment Immutable deployments shift a lot of the heavy lifting (including downloading, compiling and validating) to the build stage, adding time to this phase of deployment but protecting against failures in production.

Once images are built, deployments are as fast as you can bring up a server on a particular cloud, including rolling back a problematic change. Also, deployments become effectively atomic, letting you perform rolling or cluster-scale deployments while ensuring no half-state exists on any particular server.

Simpler Experimentation and Rollbacks A single image per deployment means you can create a short-lived experiment and destroy it once it's complete. Experiments are also cheaper because you know you're starting from the same base you have running in production, down to the last byte. This makes it easier to track the changes you're trying to make.

Better Reproducibility Immutability ensures that the same code returns the same output in every laptop, every continuous integration (CI) and every server. There is no uncertainty as to what combination of code and configuration is being used. If you have to make changes to the environment or running images, you can easily redeploy from the same image to go back to a known-good state. This provides the closest thing to a "production-like environment," given that the images themselves are produced every time you make a change.

Tighter Security and Compliance By baking an image each time, you know *exactly* what's running on any server at any point in time, allowing you to audit a single artifact and ensure that's what's running everywhere. In extreme cases, you can use resources like the Docker read-only mode to ensure no subsequent changes were made to instances when they're run.

Improved Disaster Recovery To create immutable infrastructure, you need to have a minimum level of automation in your deployments. When (not if) your infrastructure collapses, this improvement can allow you to recover in minutes instead of hours, days or weeks.

Climb the Immutable Ladder

Creating immutable services isn't always easy, especially if your project was built on assumptions common to legacy datacenters—like being able to write anywhere on the file system or overwrite existing files for fast in-place upgrades. However, you can work toward an immutable service in just five steps, as depicted in **Figure 1**, with each step providing tangible benefits.

A climb all the way up the immutable ladder requires investment and patience, but you'll see early benefits even with some small changes. Let's explore the steps in an immutable services effort.

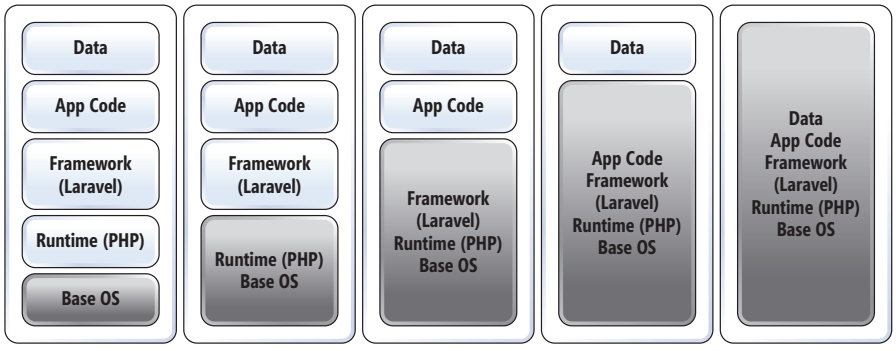


Figure 1 **5 Steps to Immutability**

Step 0: Initial Disentanglement

This step is where a lot of disentangling happens. By taking time to see how current processes, scripts and templates are deeply coupled to each other, you can reap many of the benefits of immutability by having a more reproducible deployment strategy:

- Immutable deployments forbid in-place upgrades of your code.
- Choose the fastest path that allows for image-based deployments instead of in-place upgrades. There are several ways to tackle this: You can snapshot your current VM and use that as the base image and replace the code on each deployment, for instance, or you can tweak your config management software to fully configure a vanilla image.
- With the new model you'll deploy each instance as a new image. That means you must be able to switch traffic to new instances either at the DNS level or in the HTTP front end, be it load balancers, HTTP servers, caching layers or the like.
- Critical data must be stored outside of the image file system, either externally or in a mounted volume. This will continue to be a requirement in all subsequent steps.

At this stage, many things can fail—for example, you might not be able to download a system package dependency or the system could fail to configure properly due to a race condition or unexpected version skew. Rebuilding the whole image each time in one way or another will usually take a significant amount of time.

Step 1: Isolate from the OS

Once you've completed step 0, the next target is to untangle yourself from the OS layer. Doing this lets you start many different services and deployments from the same, well-known barebones image. It can also reduce the number of images being maintained and speed up the build process. Among things to keep in mind:

- Start by moving all your dependencies out of system directories.
- Programming languages and frameworks are generally well-suited to the separation required in this step. For example, Python's Virtualenv, Ruby's Gemfiles and PHP's Composer all have easy ways to isolate dependencies from the rest of the system by using relative paths to store them by default.
- Compiled dependencies can be more difficult to handle, because they often expect to find dependencies on specific system paths.
- At this stage you'll be replacing the whole server on each deployment, so you'll need to make sure you don't throw away valuable data. It's important that system-level logs be stored externally to the instance.

Step 2: Isolate from the Runtime

So you've untangled your service from the underlying OS. Now you need to make sure your language runtime is predictable and reliable. Roll your language runtime into your base image—including any language configuration options you may need to set by default—and you'll know you can rely on the same behavior from it in every instance:

- Runtimes are trivial to update. They don't change that often and are easy to keep up-to-date even when rolled into base images.
- Rolling runtimes into base images lets you control when updates happen, which is important because updates can break compatibility and force you to port applications.
- An isolated runtime can eliminate baffling production challenges produced by small configuration differences, like between an 8MB and 64MB memory_limit.
- Immutability at this step is still easy to achieve because both the OS and the language runtime are amenable to isolation from everything else on the system.

Step 3: Isolate from the Framework

If you got to this step, you've probably got the hang of this immutability thing. You can start to see why the immutable ladder is worth climbing. With step 3, your commitment will get tested, as some unique challenges start to show themselves when working with frameworks. Among them:

- Frameworks change a bit more often than runtimes, so you'll have to update base images more often. That said, being able to manage framework updates on your own schedule is important, as even minor updates can break compatibility.
- Frameworks often present more security issues, because their surface of attack is wider and usually exposed to the Internet. For instance, Python produces 26 common vulnerabilities and exposures (CVE) versus 48 CVEs for Django.
- Disentangling your deployments gets tougher at this stage, because frameworks tend to assume they're part of your code base. You must learn more about the inner workings of your framework to understand the best way to manage the separation between code and framework.

Step 4: Fully Baked Image per Deploy

You've made the climb! This is your destination. When your images boot, they're ready for action. There's no downloading, initializing or major configuring of any software in the image. The only things that need tweaking at this stage are minor per-instance values, such as IP addresses or instance names:

- Your app code and data are completely separate from the underlying OS, runtime and framework. Bake those three underlying elements into your base image, and you can update code and data independently.
- You don't need to rebuild your service from scratch to get here, but some extra effort may be needed to ensure you're only writing in specific places.
- Once you've managed to set everything up to match this step, you can use auto-scale solutions to spin images up or down quickly and reliably.

Bonus Side Step: Immutable + Stateless

Not all services will fit into this model. However, if you happen to have a stateless service such as a worker farm, a caching system or a static Web site, this approach allows you to keep all the data in the image itself. This arrangement reduces the complexity of your deployment by serving it directly from the instance, and is ideal for:

- Read-only data like static Web sites (no runtime in production).
- Workers that process data and farm out to a queue (same runtime, no local state).

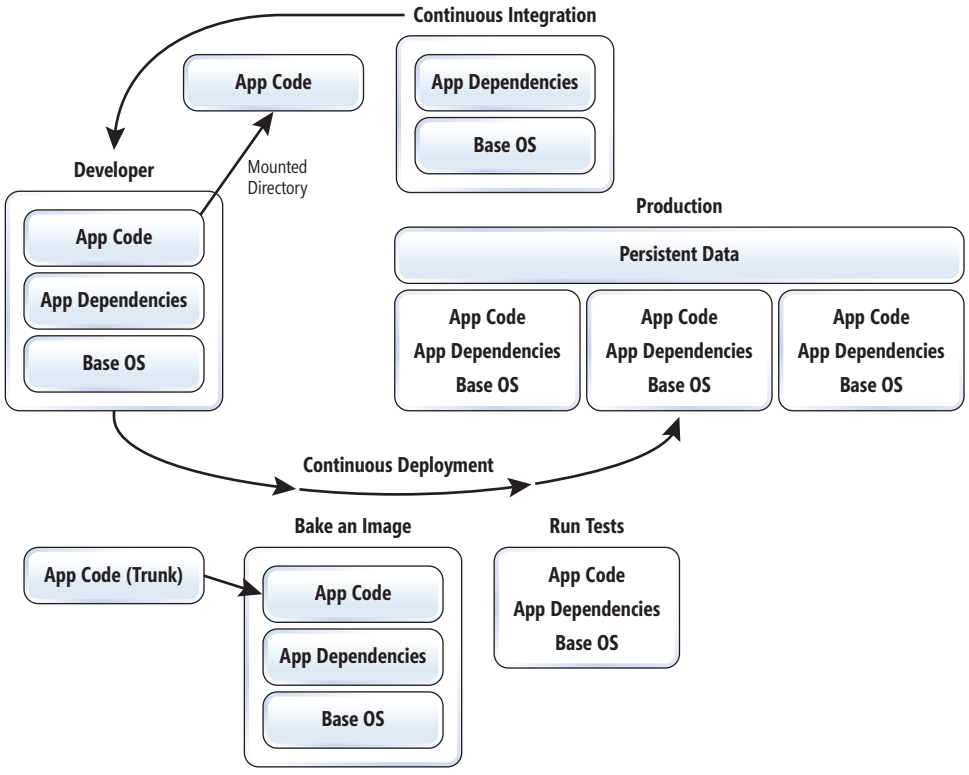


Figure 2 **The Optimized Production Cycle**

The Immutable Ladder

Now that your infrastructure is immutable, your development-to-production cycle will be optimized, seamless and much more fluid, as shown in **Figure 2**.

Public clouds are generally built to be less reliable than datacenters, because they share underlying resources with potential noisy neighbors. In exchange, you can automatically scale and shut down resources in response to changes in demand. This arrangement means you should optimize for instances to be shut down and replaced. The more your instance scales, the more the investment in automation pays off, because you'll need to spin up instances more frequently. You'll find that optimizing your instances becomes a much more seamless process, enabling you to focus on development rather than handholding deployments.

To quote Sylvester Stallone in the movie "Rocky Balboa": "It hurts now … but, one day, it'll be your warm up." As you start at step 0 of the immutable ladder, you may have to remind yourself that the pain is worth the gain. But the gain is considerable. An immutable approach allows you to make sure that things work well *before* they go to production, rather than discovering issues after OSes, frameworks, runtimes and application code have a chance to work against each other in a live fire deployment. Now more failures can be discovered and dealt with "on the ground" before deployment, rather than in the air when your environment is up and running.

The 3 Layers

Once you're producing an image per deployment, you'll want to think about your system architecture as having three layers: a volatile layer, a persistent data layer and a routing layer. At Bitnami, we've spent a lot of time thinking about these layers and how they affect each open source application, especially those that are usually used as open source building blocks for others, such as databases and language frameworks.

Layer 1: Volatile Deployments are now disposable images, which get thrown away at each deployment and replaced with a freshly built one. That makes the code and all the data it writes locally disposable. You should be prepared to delete and replace any of these instances at any point in time. These effectively become stateless servers.

Layer 2: Persistent Data This is where you'll store your important databases, user files, session state and anything else that's important to keep around. You'll want to store service and system logs at this layer for compliance and debugging purposes, as well. It can be hard to ensure that no single point of failure exists in this layer, so cloud Software-as-a-Service solutions can come in handy to address these concerns. This layer should be robust, highly available and fully backed up.

Layer 3: Routing Once you have a volatile layer, you need a predictable and addressable layer for external users and other services to access your services. You'll need to have predictable public IP addresses and domain names, decoupled from the volatile servers running your code and able to forward requests to a flexible pool of servers.

Wrapping Up

The take-away for developers is clear: Building and running immutable services provides enormous benefits across the pipeline from development and build to production. With each step you take on the immutable ladder, your existing services gain extra benefits and advantages.

Immutable infrastructure at its heart is about full automation of deployments and enforcing a process that guarantees—to the maximum extent possible—what's running on your servers. Step 0 is the place to start. Even if you only walk up that first step, your cloud workloads will be in a much better situation than they were before. ■

Martin Albisetti *is a senior architect for Bitnami and was previously director of engineering at Canonical (Ubuntu). Albisetti spends his free time working with his hands in rural Uruguay, where he lives with his family. You can find him on Twitter: @beuno.*

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

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

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

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 by October 16 and Save $200!

Must use discount code ANEB01

REGISTER NOW

EVENT PARTNER Microsoft

GOLD SPONSOR TOOLS ContaNet Tools

SUPPORTED BY Visual Studio msdn magazine Visual Studio MAGAZINE

PRODUCED BY 1105 MEDIA YOUR GROWTH. OUR BUSINESS.

ANAHEIM 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, October 16, 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: Developer Dive into SQL Server 2016 - *Leonard Lobel*	
6:45 PM	9:00 PM	Dine-A-Round			
START TIME	**END TIME**	**Visual Studio Live! Day 1: Tuesday, October 17, 2017**			
7:00 AM	8:00 AM	Registration - Coffee and Morning Pastries			
8:00 AM	9:00 AM	KEYNOTE: Self Help for Developers - **Dustin Campbell,** *Principal Software Engineer, .NET Developer Experience Team Microsoft*			
9:15 AM	10:30 AM	**T01** What's New in TypeScript? - *Doris Chen*	**T02** Go Mobile with C#, Visual Studio, and Xamarin - *James Montemagno*	**T03** What's New in Visual Studio 2017 - *Robert Green*	**T04** Accelerating Continuous App Innovation with Shift Left and DevOps - *Abel Wang*
10:45 AM	12:00 PM	**T05** Build Object-Oriented Enterprise Apps in JavaScript with TypeScript - *Rachel Appel*	**T06** Optimizing and Extending Xamarin.Forms Mobile Apps - *James Montemagno*	**T07** What's New for Developers in SQL Server 2016 - *Leonard Lobel*	**T08** To Be Announced
12:00 PM	1:30 PM	Lunch			
1:30 PM	2:45 PM	**T09** Angular 101: Part 1 - *Deborah Kurata*	**T10** Get Started with Git and GitHub - *Robert Green*	**T11** Exploring T-SQL Enhancements: Windowing and More - *Leonard Lobel*	**T12** Microsoft Teams - More Than Just Chat! - *Nedra Allmond*
3:00 PM	4:15 PM	**T13** Angular 101: Part 2 - *Deborah Kurata*	**T14** Do It Again, Faster! Automate Your Windows Deployment Pipeline - *Abel Wang*	**T15** What's New in Azure IaaS v2 - *Eric D. Boyd*	**T16** Open Source for the Microsoft Developer - *Rockford Lhotka*
4:15 PM	5:30 PM	Welcome Reception			
START TIME	**END TIME**	**Visual Studio Live! Day 2: Wednesday, October 18, 2017**			
7:00 AM	8:00 AM	Registration - Coffee and Morning Pastries			
8:00 AM	9:15 AM	**W01** I Just Met You, and "This" is Crazy, But Here's My NaN, So Call(Me), Maybe? - *Rachel Appel*	**W02** Tactical DevOps with VSTS - *Brian Randell*	**W03** Go Serverless with Azure Functions - *Eric D. Boyd*	**W04** What's New in C#7 - *Jason Bock*
9:30 AM	10:45 AM	**W05** Practical Performance Tips and Tricks to Make Your HTML/ JavaScript Faster - *Doris Chen*	**W06** Real World VSTS Usage for the Enterprise - *Jim Szubryt*	**W07** Cloud Oriented Programming - *Vishwas Lele*	**W08** I'll Get Back to You: Understanding Task, Await, and Asynchronous Methods - *Jeremy Clark*
11:00 AM	12:00 PM	General Session: The Future of Technology Seen through the Eyes of Computer Vision - **Tim Huckaby,** *Founder / Exec Chairman, InterKnowlogy, Actus, VSBLTY*			
12:00 PM	1:30 PM	Birds-of-a-Feather Lunch			
1:30 PM	2:45 PM	**W09** Assembling the Web - A Tour of WebAssembly - *Jason Bock*	**W10** Database Lifecycle Management and the SQL Server Database - *Brian Randell*	**W11** Microservices with Azure Container Service & Service Fabric - *Vishwas Lele*	**W12** Getting Started with Entity Framework Core - *Jim Wooley*
3:00 PM	4:15 PM	**W13** Building Single Page Web Applications Using Aurelia.js and the MVVM Pattern - *Ben Hoelting*	**W14** Getting to SAFe in the Enterprise - *Jim Szubryt*	**W15** Busy Developer's Guide to the Clouds - *Ted Neward*	**W16** Improving Code Quality with Roslyn Code Analyzers - *Jim Wooley*
4:30 PM	05:45 PM	**W17** Securing Angular Apps	**W17** Building Applications with DocumentDb - New Features and Best Practices - *Raj Krishnan*	**W18** Busy Developer's Guide to the Google Cloud Platform - *Ted Neward*	**W19** Learn to Love Lambdas (and LINQ, Too) - *Jeremy Clark*
7:00 PM	9:00 PM	VSLive! Grand Prix Event			
START TIME	**END TIME**	**Visual Studio Live! Day 3: Thursday, October 19, 2017**			
7:30 AM	8:00 AM	Registration - Coffee and Morning Pastries			
8:00 AM	9:15 AM	**TH01** ASP.NET Core MVC - What You Need to Know - *Philip Japikse*	**TH02** Tools for Modern Web Development Dev Ops - *Ben Hoelting*	**TH03** The Rise of the Machines - Machine Learning for developers - *Adam Tuliper*	**TH04** Storyboarding 101 - *Billy Hollis*
9:30 AM	10:45 AM	**TH05** Role-Based Security Stinks: How to Implement Better Authorization in ASP.NET and ASP.NET Core - *Benjamin Day*	**TH06** Building Cross-Platform Apps in C# using CSLA .NET - *Rockford Lhotka*	**TH07** Introduction to Machine Learning with R - *Raj Krishnan*	**TH08** Agile: You Keep Using That Word... - *Philip Japikse*
11:00 AM	12:15 PM	**TH09** From Zero to the Web API - *Paul Sheriff*	**TH10** Programming with the Model-View-ViewModel Pattern - *Rockford Lhotka*	**TH11** I'm Emotional - Using Microsoft Cognitive Services to Understand the World Around You - *Adam Tuliper*	**TH12** Top 10 Ways to Go from Good to Great Scrum Master - *Benjamin Day*
12:15 PM	1:15 PM	Lunch			
1:15 PM	2:30 PM	**TH13** Cortana Everywhere: Speech, Conversation & Skills Development - *Nick Landry*	**TH14** Roll Your Own Dashboard in XAML - *Billy Hollis*	**TH15** Unit Testing T-SQL Code - *Steve Jones*	**TH16** C# and Visual Basic code-focused development with Visual Studio 2017 - *Dustin Campbell*
2:45 PM	4:00 PM	**TH17** Securing Web Apps and APIs with IdentityServer - *Brian Randell*	**TH18** Windows 10 for Developers: Building Universal Apps for 1B Devices - *Nick Landry*	**TH19** A Tour of SQL Server Security Features - *Steve Jones*	**TH20** Real World Applications for Dependency Injection - *Paul Sheriff*

Speakers and sessions subject to change

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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

vslive.com/anaheimmsdn

Speed Thrills: Could Managed AJAX Put Your Web Apps in the Fast Lane?

Thomas Hansen

According to several studies on the subject, two of the most important concerns when creating an AJAX Web app are speed and responsiveness. These are probably some of the reasons why some developers choose to create native apps instead of Web apps. But what if I told you there exists a way to create AJAX Web apps that are 100 times faster and more responsive than the apps you might know?

I've invented a way to create 100 percent pure JavaScript-based AJAX Web apps that reduce the bandwidth usage for your apps by at least 10 times, sometimes by as much as 300 times, depending on what types of tools you're using and what you want to create. I refer to this technique as "Managed AJAX."

Managed AJAX is to some extent inspired by the way Microsoft built the common language runtime (CLR). For instance, when you create a C# application, the compiler creates a CLR assembly. This implies that your runtime environment for your end result is a "managed environment." When you create a managed AJAX app, your result doesn't compile down to anything different than a normal plain ASP.NET Web site; it becomes a managed environment, where the JavaScript parts of your end results are completely abstracted away, the same way the CPU instructions are abstracted away when you have a CLR assembly.

This article discusses:

- The managed AJAX concept and how it can sharply reduce bandwidth requirements and yield improved performance
- How to adapt an ASP.NET Web Forms application to reference the p5.ajax.dll assembly
- Leveraging managed AJAX widgets to control HTML markup and update pages via AJAX requests and responses

Technologies discussed:

ASP.NET, Managed AJAX

How Does This Work?

Managed AJAX requires almost no new knowledge. If you've done any ASP.NET development, you can drop a new Control library into your .aspx pages, and continue your work, almost exactly as you've done before. You create a couple of ASP.NET controls, either in your .aspx markup, or in your C#/F#/Visual Basic .NET codebehind. Then you decorate your controls' properties, add a couple of AJAX event handlers and you're done!

The initial rendering creates plain-old HTML for you. But every change you make to any of your controls on the server side during an AJAX request is passed to the client as JSON. Therefore, the client can get away with a tiny JavaScript library, less than 5KB in total size, and you can create rich AJAX controls, such as TreeViews, DataGrids and TabViews, without ever having to use more than 5KB of JavaScript.

Realize that at this point, you've already outperformed jQuery as a standalone JavaScript file by almost one order of magnitude (jQuery version 2.1.3 after minification and zopflinication is 30KB).

Hence, by simply including jQuery on your page, and no other JavaScript, you've already consumed 6 times as much bandwidth as you would using a managed AJAX approach. As you start consuming jQuery in your own JavaScript, this number skyrockets.

Pulling in jQuery UI in its minified and zipped version makes your JavaScript portions increase by 73.5KB. Simply including jQuery and jQuery UI on your page increases its size by an additional 103.4KB (103.4KB divided by 4.8KB becomes 21.5 times the bandwidth consumption). At this point, you still haven't created as much as a single UI element on your page, yet jQuery+jQuery UI consumes almost 22 times the space of your managed AJAX approach. And you can create almost every possible UI widget you can dream of with this 5KB of JavaScript, including most UI controls that jQuery+jQuery UI can create. Basically, regardless of what you do, you'll rarely if ever exceed this 5KB limit of JavaScript. And the other parts of your app, such as your HTML and your CSS, might also become much smaller in size.

Using this approach, you create rich AJAX UI controls, such as AJAX TreeView controls, DataGrid controls and TabView controls. And you never need additional JavaScript as you create these widgets. There's also no new knowledge required to use these controls. You simply use them (almost) the same way you'd consume a traditional WebControl from ASP.NET.

The managed AJAX approach has two distinct ways of handling the HTTP requests to your page. One of the handlers is the normal plain HTTP request, which will simply serve HTML to the client. The other handler lets you check for the existence of an HTTP post parameter. If this parameter exists, the handler will render only the changes done to each control back to the client as JSON. During an AJAX request, each control created by modifying the page's control hierarchy will be automatically recreated for you, with the properties it had in your previous request. On the client side, you have a general handler that handles your JSON properties to your controls and generically updates the attributes and DOM event handlers of the DOM elements on the client.

This approach has a lot of positive side effects, such as not fighting the way the Web was originally created by rendering the HTML elements as just that—HTML elements. That implies that semantically, your Web apps become more easily understood (by search engine spiders, as an example). In addition, it creates a superior environment for actually modifying things on the client, if you wish to use the best of both worlds.

You can still combine this approach—with as much custom JavaScript as you wish—at which point you can simply inspect the HTML rendered in your plain HTML requests. Compare this to the "magic div" approach, used by many other AJAX UI libraries, often containing megabytes of JavaScript to create your DataGrids and TreeViews.

Thus, you can understand that the 100 times faster and more responsive figure isn't an exaggeration. In fact, compared to all the most commonly used component UI toolkits, used in combination with C# and ASP.NET, it's usually between 100 and 250 times faster and more responsive with regard to bandwidth consumption. I recently did a performance measure between the managed AJAX TreeView widget in System42 and three of the largest component toolkits on the ASP.NET stack. I found the difference in bandwidth consumption to be somewhere between 150 and 220 times faster and more responsive.

To illustrate what this implies, imagine you're on an extremely slow Internet connection, where it takes one second to download 5KB of JavaScript. This implies one second to download the managed AJAX JavaScript and possibly as much as three minutes 40 seconds to download the JavaScript parts of some of the other toolkits. Needless to say, imagine what the difference would be with regard to conversion if you built two e-commerce solutions with these two approaches.

Show Me the Code

OK, enough talk, let's get down and dirty. First, download Phosphorus Five at bit.ly/2u5W0EO. Next, open the p5.sln file and build Phosphorus Five, such that you can get to your p5.ajax.dll assembly. You're going to consume p5.ajax.dll in your Web app as a reference, so you need to build it before you proceed. Notice that Phosphorus Five consists of more than 30 projects, but in this article I'm focusing on the p5.ajax project.

Next, create a new ASP.NET Web site in Visual Studio. Make sure you create a Web Forms application. In Visual Studio for Mac, you can find this under File | New Solution | Other | .NET | ASP.NET Web Forms Project.

Create a reference inside your newly created Web site to the already built p5.ajax.dll assembly and modify the web.config to resemble the following code:

```
<?xml version="1.0"?>
<configuration>
  <system.web>
    <pages clientIDMode="Static">
      <controls>
        <add assembly="p5.ajax" namespace="p5.ajax.widgets" tagPrefix="p5" />
      </controls>
    </pages>
    <compilation debug="true" targetFramework="4.5" />
    <httpRuntime targetFramework="4.5" />
  </system.web>
</configuration>
```

The important parts in this code are the "clientIDMode" and the "add assembly." At this point you can use any of the p5.ajax controls from inside your .aspx markup by prefixing them with p5. Make sure you modify the Default.aspx page's content to resemble the code in **Figure 1**.

Figure 1 **Creating a Page with a Single Button that Changes Its Text When Clicked**

```
<%@ Page Language="C#" Inherits="foobar.Default" %>
<!DOCTYPE html>
<html>
<head runat="server">
  <title>Default</title>
</head>
<body>
  <form id="form1" runat="server">
    <p5:Literal
      runat="server"
      id="foo"
      onclick="foo_onclick"
      Element="button">Click me!</p5:Literal>
  </form>
</body>
</html>
```

Then change its codebehind to the following:

```
using System;
namespace foobar
{
  public partial class Default : p5.ajax.core.AjaxPage
  {
    [p5.ajax.core.WebMethod]
    public void foo_onclick(p5.ajax.widgets.Literal sender, EventArgs args)
    {
      sender.innerValue = "Hello World from Managed Ajax!";
    }
  }
}
```

Notice that you first need to inherit your page from AjaxPage, add a WebMethod attribute to the event handler and specifically strongly type the first parameter to your event handler as a "Literal" widget. At this point you can start your Web site, click the button and enjoy your result. If you get weird bugs when debugging your Web site, make sure you turn off the "browser link" settings for Visual Studio, which is normally a toolbar button, at the top of Visual Studio. If you're curious about what goes on here, try to inspect your HTTP requests. Also make sure you take a look at its initial HTML.

Whoa, What Was That?

That was managed AJAX in practice. There are several important points to this idea that should be considered. First, you can modify any property in any control on your page from any AJAX event handler in your page. If you created another Literal widget, "Element" type "p" for instance, you could update its content from your button's "foo_onclick" event handler.

Second, you can dynamically add, remove, update or retrieve any property from your widget any way you see fit. To reference any attribute or event handler, simply use the subscript operator from C#. In **Figure 2**, instead of setting the widget's innerValue, its style property is checked and toggles a yellow background, using the CSS style property. Notice how it's able to persist and "remember" the style property of your widget. Notice also that this is done without passing huge amounts of ViewState back and forth between the client and the server.

In a real-world application, you'd probably want to use CSS classes, which could be done by exchanging the reference in **Figure 2** from "style" to "class." But I wanted to keep this example simple, so I didn't mix CSS files in here, instead using the style attribute for convenience. Using the approach shown in **Figure 2**, you can add, remove and change any attribute you wish, on any widget on your page.

Figure 2 **Toggling the Background Color**

```
using System;
namespace foobar
{
  public partial class Default : p5.ajax.core.AjaxPage
  {
    [p5.ajax.core.WebMethod]
    public void foo_onclick(p5.ajax.widgets.Literal sender, EventArgs args)
    {
      if (sender.HasAttribute ("style"))
        sender.DeleteAttribute ("style");
      else
        sender ["style"] = "background-color:Yellow;";
    }
  }
}
```

And the third—probably most important—point is that you can dynamically add, remove, update and insert any new AJAX control into any other widget, as you see fit. Before you have a look at this final point, though, you'll need to examine the "trinity of widgets."

There are three different widgets in p5.ajax, but they're also very similar in their API. By combining these three widgets together using composition, you can create any HTML markup you wish. In the example in **Figure 2**, you used the Literal widget. The Literal widget has an "innerValue" property, which on the client side maps to "innerHTML," and simply lets you change its content as a piece of string or HTML.

The Container widget can contain widgets. And it will remember its Controls collection and let you dynamically add, remove or change its collection of controls dynamically during AJAX requests.

The third widget is the Void widget, which is exclusively used for controls that have no content at all, such as HTML input elements, br elements, hr elements and so on. The most important one for the example here is probably the Container widget. So go ahead and change the code in the .aspx page to what you see in **Figure 3**.

The widget hierarchy in **Figure 3** will create one "button" and a "ul" element with one "li" child element. Next, change the C# codebehind to the code in **Figure 4**.

Realize that the last piece of code dynamically injected new widgets into the Container widget. Basically, new "li" elements were appended into the "ul" element, dynamically during an AJAX request, and it simply worked! These widgets are also persistently remembered across AJAX requests, such that you can change their properties and invoke event handlers for them. In addition, through the "Element" property any HTML elements can be rendered and any attribute added through the subscript operator.

You now have 100 percent perfect control over your HTML markup, and you can create tiny AJAX requests and responses that update anything you want to update on your page in any way you see fit. And you did it with 4.8KB of JavaScript. You've turned Web

Figure 3 **Creating a Page with a Button and a Bulleted List Containing One List Item**

```
<%@ Page Language="C#" Inherits="foobar.Default" %>
<!DOCTYPE html>
<html>
<head runat="server">
  <title>Default</title>
</head>
<body>
  <form id="form1" runat="server">
    <p5:Literal
      runat="server"
      id="foo"
      onclick="foo_onclick"
      Element="button">Click me!</p5:Literal>
  <p5:Container
      runat="server"
      id="bar"
      Element="ul">
      <p5:Literal
        runat="server"
        id="initial"
        onclick="initial_onclick"
        Element="li">Initial list element, try clicking me!</p5:Literal>
  </p5:Container>
  </form>
</body>
</html>
```

msdn
magazine

MOBILE.
TABLET.
DESKTOP.
PRINT.

WHERE YOU
NEED US
MOST.

MSDN.MICROSOFT.COM

Figure 4 **Mapping Up AJAX Event Handlers to Create a New List Item**

```
using System;

namespace foobar
{
  public partial class Default : p5.ajax.core.AjaxPage
  {
    protected p5.ajax.widgets.Container bar;

    [p5.ajax.core.WebMethod]
    public void foo_onclick(p5.ajax.widgets.Literal sender, EventArgs args)
    {
      // Using the factory method to create a new child widget for our "ul" widget.
      var widget = bar.CreatePersistentControl<p5.ajax.widgets.Literal>();
      widget.Element = "li";
      widget.innerValue = "Try clicking me too!";

      // Notice we supply the name of the method below here.
      widget ["onclick"] = "secondary_onclick";
    }

    [p5.ajax.core.WebMethod]
    public void initial_onclick(p5.ajax.widgets.Literal sender, EventArgs args)
    {
      sender.innerValue = "I was clicked!";
    }

    [p5.ajax.core.WebMethod]
    public void secondary_onclick(p5.ajax.widgets.Literal sender, EventArgs args)
    {
      sender.innerValue = "I was ALSO clicked!";
    }
  }
}
```

app AJAX development into a thing that can be done just as easily as plain-old Windows Forms development. And in the process, you ended up with 100 times faster and more responsive Web apps.

An Exercise in Hyperlambda

A few months back I wrote an article in the June 2017 issue of *MSDN Magazine titled* "Make C# More Dynamic with Hyperlambda" (msdn.com/magazine/mt809119), which explored the non-traditional Hyperlambda programming language with its roots in execution trees. I bring this up because Hyperlambda's tree-based approach makes it extremely easy to declare an AJAX widget hierarchy. Combine p5.ajax with Hyperlambda to consume an AJAX TreeView widget, and some impressive efficiencies show up.

Figure 5 **Creating an AJAX TreeView, Which Will Allow for Traversing Folders on Disk**

```
create-widget
  parent:content
  widgets
    sys42.widgets.tree
      crawl:true
      items
        root:/
      .on-get-items
        list-folders:x:/../*/_item-id?value
        for-each:x:/@list-folders/*?name
          list-folders:x:/@_dp?value
          split:x:/@_dp?value
            =:/
          add:x:/../*/return/*
            src:@"{0}:{1}"
              :x:/@split/0/-?name
              :x:/@_dp?value
          if:x:/@list-folders/*
            not
            add:x:/../*/return/*/items/0/-
              src
                class:tree-leaf
        return
          items
```

Let's explore this. First, in addition to Phosphorus Five, you need to download System42 and put it into the main p5.webapp folder according to the instructions at bit.ly/2vbkNpg. Then start up System42, which contains an ultra-fast AJAX TreeView widget, open up "CMS," create a new lambda page by clicking the +, and paste the code shown in **Figure 5**.

Click Settings, choose empty as your Template, click OK, save your page and click View page.

Try expanding the AJAX TreeView while inspecting what goes over the wire in your HTTP requests, and realize that you just built a folder browsing AJAX TreeView with 24 lines of Hyperlambda that will display your folders from your p5.webapp folder, and that its initial total bandwidth consumption was only 10.9KB!

If you compare these results with any other AJAX toolkit, you'll often find that other toolkits require downloading several megabytes of JavaScript—in addition to all the other stuff that goes over the wire—while Hyperlambda TreeView has no more than 4.8KB of JavaScript.

This AJAX TreeView was built with a total of 717 lines of code, in pure Hyperlambda, using nothing but the Literal, Container and Void widgets. Most of its code is made up of comments, so roughly 300 lines of code were required to create the AJAX TreeView control. The widget was consumed with 24 lines of Hyperlambda, which let you browse your folders on disk. But it would require thousands of lines of code to create the control with anything else, and hundreds of lines to consume it, as was done in the 24 lines of code in **Figure 5**.

If you wanted, you could now exchange three lines of code in the Hyperlambda example and end up with your own specialized Active Event custom widget, which would let you consume your specialized widget with a single line of code. Read how to do that at bit.ly/2t96gsQ.

So, you're now able to create an AJAX TreeView that will browse your server's folders with one line of code. To create something equivalent in other toolkits would often require hundreds of lines of code in four different languages. You did it with one line of code, in one programming language and it performs up to 300 times better than its competition.

Wrapping Up

Imagine being able to produce 100 times better results, 100 times faster and more optimized results, 100 times better quality, with 100 times fewer less bugs, and being 100 times more productive than you were before. To make sure you're using the latest goods, download Phosphorus Five at bit.ly/2uwNv65 and System42 at bit.ly/2vbkNpg. ■

Thomas Hansen *has been creating software since he was 8 years old, when he started writing code using the Oric-1 computer in 1982. Occasionally, he creates code that does more good than harm. His passions include the Web, AJAX, Agile methodologies and software architecture. Contact him at thomas@gaiasoul.com.*

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

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

Multiplayer Networked Physics for Web Game Development

Gary Weiss

Web games have a shoddy reputation. Well-written games are available on high-end game consoles, and for computers equipped with modern graphics processors. Even mobile gaming has come a long way since the canonical "Snake." But Web games, like Rodney Dangerfield, get no respect. That said, recent advances in modern browsers have the potential to bring quality gaming experiences to the browser. The most important technology in this respect is WebGL, which brings direct access to hardware-accelerated graphics.

In this article, I'll investigate Web gaming from a specific point of view: implementation of a simple multiplayer networked-physics game. This exercise investigates which technologies and game development tools are available for the task, and stress tests modern browsers for networking performance, computation efficiency, and rendering. This article can be viewed as a small investigation that is part of a much larger theme: Can a professional game studio really write a high-end game in a browser?

The source code for the game is freely available at bit.ly/2toc4h4, and the game is playable online at sprocketleague.lance.gg.

This article discusses:

- Game architecture
- Client-server communication
- Game logic
- Client-side prediction
- Web game challenges

Technologies discussed:

WebGL, A-Frame, Cannon.js, Lance.gg

Code download available at:

bit.ly/2toc4h4

The Game

The game I'll model involves cars driving around in a stadium. There's a large ball in the stadium that can be bounced into the goals. This concept is inspired by "Rocket League," a popular e-sports title by Psyonix. Therefore, the game is assembled from the following objects: an arena (including the goals), cars and a ball, as you can see in **Figure 1**. The player can move the car forward or in reverse and turn the wheels.

The Architecture of the Game

The game implementation uses the authoritative server model, whereby each client forwards user inputs to a central server. Clients won't wait for the network before they apply the inputs locally; rather, the clients will perform what's commonly referred to as client-side prediction. This technique allows the client to continue running the game logic and game physics. The catch, of course, is that when updates to positions and velocities are broadcast from the authoritative server, each client must make its own incremental corrections so that the game preserves a smooth visual experience.

Figure 1 **Cars Playing Soccer in Your Browser**

Rendering The main goal in rendering is accessing the GPU. There are several tools that can fill this slot by simplifying the interface to WebGL. Babylon.js and Three.js have been used for many advanced Web games. For this game I used A-Frame, a relatively new library that allows you to build a 3D scene using HTML elements. A-Frame defines new HTML elements that you can use as building blocks. Also, A-Frame does a good job of modeling the objects using an Entity-Component-System design pattern. Another interesting advantage of A-Frame is that it was designed for virtual reality (VR), so the game can be easily made into a VR experience.

Physics Cannon.js is a relatively easy-to-use physics engine. It provides rigid-body mechanics with collision detection. It was written in JavaScript from scratch, which means the library is much smaller than transpiled engines such as ammo.js.

Object Serialization and Networking For the server to broadcast object positions and velocities, game objects need to be serialized. For this purpose I use the Lance game server (Lance.gg), an open source multiplayer engine. Game objects that derive the Lance GameObject base class will be serialized and forwarded to clients. Advanced game engines use UDP sockets for better performance, while Lance still uses TCP sockets. This is because of the complexity of WebRTC for client-server communication (as opposed to peer-to-peer). This is one area where Web gaming significantly lags behind other platforms. However, in my experience, the resulting game provides a smooth experience even at network lags of 150ms, which is typical within the continental United States or Europe.

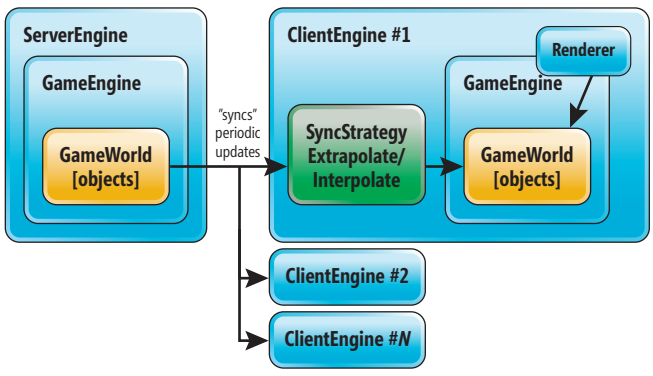


Figure 2 **The Architecture of a Multiplayer Game**

Client-Side Prediction Here, too, I relied on Lance to perform the extrapolation of positions and velocities. Lance implements the client-side collection of server broadcasts, calculates the necessary corrections for each position and orientation, and applies the correction incrementally over the progress of the rendering loop. Position corrections are applied by calculating the vector difference, and dividing this difference into increments. Orientation corrections, in contrast, are applied by calculating the quaternion difference. The correction quaternion q_Δ for a server quaternion q_s and a client quaternion q_c is given by $q_\Delta = q_s \circ q_c^{-1}$. This quaternion can be represented as a rotation about a single axis, and then divided into increments about that axis.

The game architecture, then, consists of server logic, client logic and shared logic, as shown in **Figure 2**. The client logic must handle all aspects that involve the user, including the collection of inputs and the rendering work. The server logic is simpler by comparison, but it does need to take some authoritative steps, like accepting connections, creating game objects and deciding who scored.

The shared logic is the most interesting part of the implementation, and is discussed in a separate section. It includes the main game logic, which must run on the authoritative server, but must also run on each client as the game progresses in between server broadcasts. Let's start with the nuts and bolts of multiplayer games: the communication layer.

Client-Server Communication

A multiplayer game with an authoritative server requires communication code. This code has many responsibilities. Initially, it must manage the bring-up of the servers, which listen on TCP/IP ports for incoming data. Next, there's a handshake process when a client connects to the server. This handshake registers the new client at the server, and establishes a dedicated socket for the client. Some games may require an authentication or a matchmaking process, as well. The connection of a new client, and disconnection of an existing client, have implications for the game. The game needs to be notified so it can create a new car for the new client and assign the player to a team.

While the game executes, the server sends periodic broadcasts to all clients. The broadcasts describe the game's current state. A state includes the position, velocity, orientation (quaternion) and angular velocity of each game object. Additionally, game objects may well have other non-physical attributes, such as strength, shield power, spell timeout and so forth. Here, the server needs to be very efficient

and send as little data as is absolutely required. For example, an object that hasn't moved or changed shouldn't be broadcast. On the other hand, if a new player has just connected, that player does need a full description of all the game objects to bootstrap the game state.

Last, the client-server communication will need to emit and capture out-of-band events, which describe game events that don't fit as part of the normal game progress. For example, the server may want to communicate to a specific player that he's reached a special achievement. This mechanism can also be used to report a message to all players.

Here's the implementation. On the server side, I bring up an express node.js server and a socket.io service. The code shown in **Figure 3** configures the HTTP server on port 3000, initializes the server engine and the game engine classes, and routes all HTTP requests to the dist sub-directory where all the HTML and assets are placed. This means that the node.js server has two roles. In the first role, it serves as an HTML server that serves HTML content, including CSS files, images, audio and the 3D models in GLTF format. In the second role, the HTTP server acts as a game server that accepts incoming connections on socket.io. Note that best practice is to cache all static assets behind a content-delivery network (CDN) for good performance. This configuration isn't shown in the example in **Figure 3**, but the use of a CDN dramatically increased the game bring-up performance, while reducing the unnecessary load from the game server. Finally, at the last line of the code, the game is started.

The server has started, but has yet to handle new connections. The base ServerEngine class declares a handler method for new connections called onPlayerConnected and a corresponding onPlayerDisconnected handler. This is the place where the authoritative server subclass methods can instruct the game engine to create a new car or remove an existing car. The code snippet in **Figure 4** from the base class shows how socket.io is used to ensure those handlers are called when a new player has connected.

When a player connects, this code emits the playerJoined event twice. The first event is emitted on the game engine's event controller, where other parts of the game code may have registered listeners to this specific event. The second event is sent over the player's socket. In this case, the event can be captured on the client side of the socket, and acts as an acknowledgement that the server has allowed this player to join the game.

Note the server engine's step method—here, the game state is broadcast to all connected players. This broadcast only occurs at a fixed interval. In my game I chose to configure the schedule such that 60 steps are executed per second, and a broadcast is sent every sixth step, or 10 times per second.

Now I can implement the methods in the subclass, as they apply specifically to the game. As shown in **Figure 5**, this code handles new connections by creating a new car and joining that car to

Figure 3 **Server Entry Point**

```
const express = require('express');
const socketIO = require('socket.io');

// Constants
const PORT = process.env.PORT || 3000;
const INDEX = path.join(__dirname, './dist/index.html');


// Network servers
const server = express();
const requestHandler = server.listen(PORT, () => console.log(`Listening
on ${PORT}`));
const io = socketIO(requestHandler);


// Get game classes
const SLServerEngine = require('./src/server/SLServerEngine.js');
const SLGameEngine = require('./src/common/SLGameEngine.js');


// Create instances
const gameEngine = new SLGameEngine({ traceLevel: 0 });
const serverEngine =
  new SLServerEngine(io, gameEngine, { debug: {}, updateRate: 6,
timeoutInterval: 20 });


// HTTP routes
server.get('/', (req, res) => { res.sendFile(INDEX); });
server.use('/', express.static(path.join(__dirname, './dist/')));

// Start the game
serverEngine.start();
```

Figure 4 **Connection Handler Implementation with socket.io**

```
class ServerEngine {

  // The server engine constructor registers a listener on new connections
  constructor(io, gameEngine, options) {
      this.connectedPlayers = {};
      io.on('connection', this.onPlayerConnected.bind(this));
  }

  // Handle new player connection
  onPlayerConnected(socket) {
    let that = this;

    // Save player socket and state
    this.connectedPlayers[socket.id] = {
      socket: socket,
      state: 'new'
    };
    let playerId = socket.playerId = ++this.gameEngine.world.playerCount;

    // Save player join time, and emit a `playerJoined` event
    socket.joinTime = (new Date()).getTime();
    this.resetIdleTimeout(socket);
    let playerEvent = { id: socket.id, playerId,
      joinTime: socket.joinTime, disconnectTime: 0 };
    this.gameEngine.emit('playerJoined', playerEvent);
    socket.emit('playerJoined', playerEvent);

    // Ensure a handler is called when the player disconnects
    socket.on('disconnect', function() {
      playerEvent.disconnectTime = (new Date()).getTime();
      that.onPlayerDisconnected(socket.id, playerId);
      that.gameEngine.emit('playerDisconnected', playerEvent);
    });


  }

  // Every server step starts here
  step() {

    // Run the game engine step
    this.gameEngine.step();

    // Broadcast game state to all players
    if (this.gameEngine.world.stepCount % this.options.updateRate === 0) {
      for (let socketId of Object.keys(this.connectedPlayers)) {
        this.connectedPlayers[socketId].socket.emit(
          'worldUpdate', this.serializeUpdate());
      }
    }
  }
}
```

GROUPDOCS
Document Manipulation APIs

Manipulating Documents?

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

Try GroupDocs APIs for FREE

Microsoft .NET

GroupDocs.Viewer

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

GroupDocs.Annotation

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

GroupDocs.Conversion

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

GroupDocs.Comparison

Compare two documents and get a difference summary report.

GroupDocs.Signature

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

GroupDocs.Assembly

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

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

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

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

Figure 5 **Server Connection Handling**

```
// Game-specific logic for player connections
onPlayerConnected(socket) {
  super.onPlayerConnected(socket);

  let makePlayerCar = (team) => {
    this.gameEngine.makeCar(socket.playerId, team);
  };

  // Handle client restart requests
  socket.on('requestRestart', makePlayerCar);
  socket.on('keepAlive', () => { this.resetIdleTimeout(socket); });
}

// Game-specific logic for player dis-connections
onPlayerDisconnected(socketId, playerId) {
  super.onPlayerDisconnected(socketId, playerId);
  this.gameEngine.removeCar(playerId);
}
```

either the blue team or the red team. When a player disconnects, I remove that player's car.

Communication can also occur out-of-band, meaning that an event must sometimes be sent to one or more clients asynchronously, or data must be sent that's not part of the game object states. The following sample shows how socket.io can be used on the server side:

```
// ServerEngine: send the event monsterAttack! with data
// monsterData = {...} to all players
this.io.sockets.emit('monsterAttack!', monsterData);

// ServerEngine: send events to specific connected players
for (let socketId of Object.keys(this.connectedPlayers)) {
  let player = this.connectedPlayers[socketId];
  let playerId = player.socket.playerId;

  let message = `hello player ${playerId}`;
  player.socket.emit('secret', message);
}
```

On the client side, listening to events is simple:

```
this.socket.on('monsterAttack!', (e) => {
  console.log(`run for your lives! ${e}`);
});
```

Figure 6 **The GameEngine Step Method**

```
// A single Game Engine Step
step() {
  super.step();

  // Car physics
  this.world.forEachObject((id, o) => {
    if (o.class === Car) {
      o.adjustCarMovement();
    }
  });

  // Check if we have a goal
  if (this.ball && this.arena) {

    // Check for ball in Goal 1
    if (this.arena.isObjInGoal1(this.ball)) {
      this.ball.showExplosion();
      this.resetBall();
      this.metaData.teams.red.score++;
    }

    // Check for ball in Goal 2
    if (this.arena.isObjInGoal2(this.ball)) {
      this.ball.showExplosion();
      this.resetBall();
      this.metaData.teams.blue.score++;
    }
  }
}
```

The Game Logic

The fundamental game logic involves the application of user inputs to the game state, as shown in **Figure 6**. For example, pressing on the gas should apply a forward-directed force on a car. Pressing the brakes should apply a backward-directed force on a car, potentially going in reverse. Turning the steering wheel applies an angular velocity, and so on.

These operations are implemented as calls to the physics engine, as time progresses. An interesting twist is that applying correct physical forces to a car provides a poor gaming experience. If you apply the correct physical forces just as they work in the real world, the handling and control of the car doesn't "feel right" in the hands of the player. The resulting game action is too slow. For the game to be enjoyable, I needed to apply an artificial boost when the car accelerates from very low velocities, otherwise the game action was just too slow.

Finally, the game logic must check if the ball passed the goal posts, and update the score.

Client-Side Prediction

Each multiplayer game has different requirements for client-side prediction and needs to be configured correspondingly. Game object types can also have a specific configuration. The code that follows shows a small subset of the game's configuration. User inputs are delayed by three steps, or 50ms, to better match the time at which the inputs will be applied on the server. This is done by setting delayInputCount to 3. syncOptions.sync is set to "extrapolate" to enable client-side prediction; localObjBending is set to 0.6, indicating that locally controlled object positions should be corrected by 60 percent before the next server broadcast is estimated to arrive; remoteObjBending is set higher, to 80 percent, because those objects are more likely to diverge significantly. Last, I set bendingIncrements to 6, indicating that each correction must not be applied at once, but rather over 6 increments of the render loop:

```
const options = {
  delayInputCount: 3,
  syncOptions: {
    sync: 'extrapolate',
    localObjBending: 0.6,
    remoteObjBending: 0.8,
    bendingIncrements: 6
  }
};
```

The ball has its own velocity-bending parameter (not shown) set to zero. This is because when one team scores, the ball gets teleported back to the center of the field, and any attempt to bend the resulting velocity will result in undesired visual effects.

The client-side prediction logic itself is too large to include here, so I presented it as general pseudocode in **Figure 7**. It's the secret sauce, which makes a multiplayer game possible. The first step scans the last server broadcast and checks if any new objects have been created. Existing objects remember their current state before they're synced to the server state. The second step is the reenactment phase, which re-executes the game logic for all the steps that have occurred since the step described in the broadcast. Inputs need to be reapplied, and the game engine step is executed in reenactment mode. In the third step, each object records the required correction, and reverts to the remembered state. Finally, in the last step, objects that are marked for destruction can be removed.

LIVE! 360
TECH EVENTS WITH PERSPECTIVE

SQL Server LIVE!

TRAINING FOR DBAs AND IT PROS

ROYAL PACIFIC RESORT AT UNIVERSAL ORLANDO
November 12-17

Data. Driven.

After 5 days of workshops, deep dives and breakout sessions, SQL Server Live! will leave you with the skills needed to drive data forward.

With timely, relevant content, SQL Server Live! helps administrators, DBAs, and developers do more with their SQL Server investment. Sessions will cover performance tuning, security, reporting, data integration, adopting new techniques, improving old approaches, and modernizing the SQL Server infrastructure.

REGISTER NOW

REGISTER BY OCT 13 AND SAVE $300!*

Use promo code ORLOCT1

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

A Part of Live! 360: The Ultimate Education Destination

5 GREAT CONFERENCES, 1 GREAT PRICE

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

SQLLIVE360.COM

EVENT PARTNERS
Microsoft Magenic

PLATINUM SPONSORS
okta Google Cloud

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

PRODUCED BY
1105 MEDIA YOUR GROWTH, OUR BUSINESS.

The Devil in the Details

There's much more work in this game than appears at first glance. I haven't touched on camera movement tracking the car, differences in input between mobile devices and desktops (in mobile devices car movement is controlled by tilting the device), and topics like matchmaking and debugging. All these topics are outside the scope of this article. However, it's important to mention that some of these entail significant road bumps in Web gaming. For example, a Web game can run on a desktop or on a tablet or on a mobile device—so sometimes a GPU is available and sometimes it's not, while the likelihood of low networking bandwidth increases.

Yet another challenge is the workflow required for generating compatible 3D models from existing standards. The game development toolsets for Web platforms aren't as mature as their counterparts on other platforms. Last, Web games don't have local storage in the same sense that other platforms provide. So care must be taken to load assets frugally. An impressive achievement in this area is the online game "Urban Galaxy" (bit.ly/2uf3iWB).

These issues present new challenges to game developers on the Web platform. However, there are some distinct advantages to the Web platform that must be mentioned. The debugging environment on browsers is very advanced. The dynamic nature of Web programming leads to very fast prototyping, and continuous delivery of patches. The community of Web developers is much larger.

Probably the most significant advantage for Web games is the way they're started. A Web game can be accessed by simply clicking on a Web link and doesn't require pre-downloading the entire set of game assets on the client device before it starts. By downloading assets only as they're needed, the game play experience can start within seconds, instead of minutes, and where possible, offline support can also be leveraged to improve subsequent game play.

For example, visitors logging into the NBA Web site might find themselves playing a basketball game against players from all over the world, right in the browser.

Wrapping Up

Building a multiplayer game for the Web platform is different. The Web is naturally directed at a wide array of devices and this presents a significant challenge to game development. The absence of simple UDP networking in the browser impacts multiplayer response times. However, recent advances in Web technologies have made it possible to write a multiplayer physics demo in a relatively short time. Libraries and workflows used in this exercise are making fast progress and can already be used today. With additional resources and time, it's possible to develop a quality gaming experience in a browser.

I want to thank the co-author of the game discussed in the article, Opher Vishnia, a co-founder of Lance and a multidisciplinary creator involved with game development, programming, design and music composition. ■

Figure 7 **Simplified Client-Side Prediction Pseudocode**

```
applySync(lastSync):

  // Step 1: sync to all server objects in the broadcast
  for each obj in lastSync.objects
    localObj = getLocalObj(obj.id)
    if localObj
      localObj.rememberState()
      localObj.syncTo(obj)
    else
      addLocalObj(obj)

  // Step 2: re-enactment using latest state information
  for step = lastSync.serverStep; step < clientStep; step++
    processInputsForStep(step)
    gameEngine.step(reenactment=true)

  // Step 3: record bending and revert
  for each obj in world.obects
     obj.bendToCurrentState()
     obj.revertToRememberedState()

  // Step 4: remove destroyed objects
  for each obj in world.obects
    objSyncEvents = lastSync.getEvents(obj.id)
    for each event in objSyncEvents
      if event.name == 'objectDestroy'
        gameEngine.removeObjectFromWorld(obj.id)
```

Web Technologies for Game Development

In the past couple of years, there has been significant progress in browser ability. The primary advance is the development of WebGL, which has brought direct access to GPU functionality in the browser. WebGL is a very low-level interface, so game developers typically rely on higher-level interfaces from libraries like Babylon.js, three.js and A-Frame to build complex scenes.

Other recent Web technologies include WebAudio, which provides advanced audio capabilities; Media Source Extensions (MSE), which optimize video access and allow real-time video manipulation; and WebVR, which is relatively new and provides an interface to virtual reality (VR) hardware including VR headsets and VR controllers. Using WebVR together with WebAudio, it's possible to implement ambisonic 3D audio experiences in a Web-driven VR scene.

Even the JavaScript language has been updated in useful ways with the latest ECMA-262 6th Edition. The JavaScript ecosystem is wide, providing many frameworks for developers. In fact, there are so many frameworks and methodologies that the abundance sometimes leads to confusion.

On the network front, alongside the HTTP protocol, WebSockets and WebRTC are two standards that are available for applications. WebSockets are simple bi-directional data streams, while WebRTC provides a more complex mechanism for peer-to-peer communication. For multiplayer game development, Lance.gg is an open source library that handles the networking and synchronization of game objects.

Another interesting technology is WebAssembly, which has recently reached cross-browser consensus, and has the potential to deliver a substantial performance advantage by running compiled binary format in a browser.

You can easily find more information about all of these technologies on the Web.

Gary Weiss *is a software architect and co-founder of Lance, the group that develops the open source project Lance.gg. Reach him at garyweiss74@gmail.com*

Thanks to the following technical expert for reviewing this article: *Raanan Weber*

Spreadsheets Made Easy.

SpreadsheetGear 2017 Released

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

Scalable Reporting

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

.NET Standard

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

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

WIN Windows Forms | Silverlight | WPF WPF

Powerful Controls

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

Comprehensive Charting

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

Fastest Calculations

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

Download your free fully functional evaluation at SpreadsheetGear.com

SpreadsheetGear

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

DINO ESPOSITO

Policy-Based Authorization in ASP.NET Core

The authorization layer of a software application ensures that the current user is allowed to access a given resource, perform a given operation or perform a given operation on a given resource. In ASP.NET Core there are two ways to set up an authorization layer. You can use roles or you can use policies. The former approach—role-based authorization—has been maintained from previous versions of the ASP.NET platform, while policy-based authorization is new to ASP.NET Core.

The Authorize Attribute

Roles have been used in ASP.NET applications since the early days. Technically speaking, a role is a plain string. Its value, however, is treated as meta information by the security layer (checked for presence in the IPrincipal object) and used by applications to map a set of permissions to a given authenticated user. In ASP.NET the logged user is identified by an IPrincipal object, and in ASP.NET Core the actual class is ClaimsPrincipal. This class exposes a collection of identities and each identity is represented by IIdentity objects, specifically ClaimsIdentity objects. This means that any logged user comes with a list of claims, which are essentially statements about her status. Username and role are two common claims of users of ASP.NET Core applications. However, role presence depends on the backing identity store. For example, if you use social authentication, you're never going to see roles.

> Note that in ASP.NET 2.0, authentication middleware is replaced with a service that has multiple handlers.

Authorization goes one step further than authentication. Authentication is about discovering the identity of a user, whereas authorization is about defining requirements for users to call into application endpoints. User roles are typically stored in the database and retrieved when the user credentials are validated, at which point role information is attached in some way to the user account. The IIdentity interface features an IsInRole method that must be implemented. The ClaimsIdentity class does that by checking that the Role claim is available in the collection of claims resulting from the authentication process. In any case, when the user attempts to call into a secured controller method, her role should be available for check. If not, the user is denied the call to any secured methods.

The Authorize attribute is the declarative way to secure a controller or some of its methods:

```
[Authorize]
public class CustomerController : Controller
{
  ...
}
```

Specified without arguments, the attribute only checks that the user is authenticated. However, the attribute supports additional attributes such as Roles. The Roles property indicates that users in any of the listed roles would be granted access. To require multiple roles, you can apply the Authorize attribute multiple times, or write your own filter.

```
[Authorize(Roles="admin, system"]
public class BackofficeController : Controller
{
  ...
}
```

Optionally, the Authorize attribute can also accept one or more authentication schemes through the ActiveAuthenticationSchemes property.

```
[Authorize(Roles="admin, system", ActiveAuthenticationSchemes="Cookie"]
public class BackofficeController : Controller
{
  ...
}
```

The ActiveAuthenticationSchemes property is a comma-separated string listing the authentication middleware components that the authorization layer will trust in the current context. In other words, it states that access to the BackofficeController class is allowed only if the user is authenticated through the Cookies scheme and has any of the listed roles. As mentioned, string values passed to the ActiveAuthenticationSchemes property must match authentication middleware registered at the startup of the application.

Note that in ASP.NET 2.0, authentication middleware is replaced with a service that has multiple handlers. As a result, an authentication scheme is a label that selects a handler. For more information about authentication in ASP.NET Core, you might want to check my September 2017 column, "Cookies, Claims and Authentication in ASP.NET Core" (msdn.com/magazine/mt842501).

Authorization Filters

The information provided by the Authorize attribute is consumed by the system-provided authorization filter. Because it's responsible for checking if the user is able to perform the requested

LIVE! 360
TECH EVENTS WITH PERSPECTIVE

Office & SharePoint LIVE!

ON-PREMISE, CLOUD & CROSS-PLATFORM TRAINING

ROYAL PACIFIC RESORT AT UNIVERSAL ORLANDO
November 12-17

Training to Collaborate Anywhere

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

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

REGISTER NOW

REGISTER BY OCT 13 AND SAVE $300!*

Use promo code ORLOCT1

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

A Part of Live! 360: The Ultimate Education Destination

5 GREAT CONFERENCES, 1 GREAT PRICE

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

SPLIVE360.COM

EVENT PARTNERS
Microsoft Magenic

PLATINUM SPONSORS
okta Google Cloud

SUPPORTED BY
SharePoint msdn magazine Redmond

PRODUCED BY
1105 MEDIA
YOUR GROWTH. OUR BUSINESS.

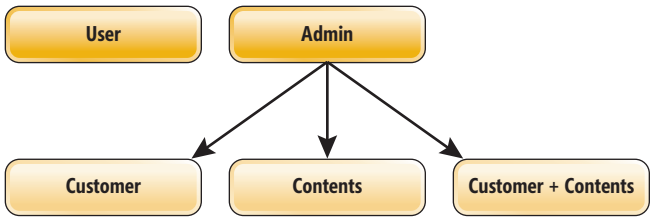


Figure 1 **Hierarchy of Roles**

operation, this filter runs before any of the other ASP.NET Core filters. If the user isn't authorized, the filter short-circuits the pipeline and cancels the requests.

Custom authorization filters can be created, but most of the time you won't need to do that. In fact, it's preferable that you configure the existing authorization layer on which the default filter relies.

Roles, Permissions and Overrules

Roles are an easy way to group the users of an application based on what they can or cannot do. But they aren't very expressive; at least, not enough to meet the needs of most modern applications. For example, consider a relatively simple authorization architecture, serving regular users of the Web site and power users authorized to access the back office software and update content. A role-based authorization layer can be built around two roles—user and admin—defining which controllers and methods each group can access.

You run into problems when it comes to overrules—subtle distinctions that describe what users can or cannot do within a given role. For example, you may have users who enjoy access to back office systems. But of those users, some are authorized only to edit customer data, some only to work on content, and some to both edit customer data *and* work on content (see **Figure 1**).

> In ASP.NET Core, the policy-based authorization framework is designed to decouple authorization and application logic.

Roles are essentially flat concepts. How would you flatten out even a simple hierarchy like the one shown in **Figure 1**? You could create four different roles—User, Admin, CustomerAdmin and ContentsAdmin—but as soon as the number of overrules grows, the number of required roles increases significantly. Even a simple exercise like this one shows that roles may not be the most effective way to handle authorizations, except for simple scenarios and instances where backward compatibility is a priority. For everything else, there's a different requirement. Enter policy-based authorization.

What's a Policy, Anyway?

In ASP.NET Core, the policy-based authorization framework is designed to decouple authorization and application logic. Simply put, a policy is an entity devised as a collection of requirements, which themselves are conditions that the current user must meet.

The simplest policy is that the user is authenticated, while a common requirement is that the user is associated with a given role. Another common requirement is for the user to have a particular claim or a particular claim with a particular value. In the most general terms, a requirement is an assertion about the user identity that attempts to access a method that holds true. You create a policy object using the following code:

```
var policy = new AuthorizationPolicyBuilder()
  .AddAuthenticationSchemes("Cookie, Bearer")
  .RequireAuthenticatedUser()
  .RequireRole("Admin")
  .RequireClaim("editor", "contents")
  .RequireClaim("level", "senior")
  .Build();
```

The builder object collects requirements using a variety of extension methods and then builds the policy instance. As you can see, requirements act on the authentication status and schemes, role, and any combination of claims read through the authentication cookie or bearer token.

If none of the predefined extension methods for defining requirements work for you, then you can always resort to defining a new requirement through your own assertion. Here's how:

```
var policy = new AuthorizationPolicyBuilder()
  .AddAuthenticationSchemes("Cookie, Bearer")
  .RequireAuthenticatedUser()
  .RequireRole("Admin")
  .RequireAssertion(ctx =>
  {
    return ctx.User.HasClaim("editor", "contents") ||
           ctx.User.HasClaim("level", "senior");
  })
  .Build();
```

The RequireAssertion method takes a lambda that receives the HttpContext object and returns a Boolean value. Therefore, the assertion is simply a conditional statement. Note that if you concatenate RequireRole multiple times, then all roles must be honored by the user. If you want to express instead an OR condition, then you may resort to an assertion. In this example, in fact, the policy allows users that are either editor of contents or senior users.

Registering Policies

It's not enough to define policies—you must also register them with the authorization middleware. You do this by adding the authorization middleware as a service in the ConfigureServices method of the startup class, like so:

```
services.AddAuthorization(options=>
{
  options.AddPolicy("ContentsEditor", policy =>
  {
    policy.AddAuthenticationSchemes("Cookie, Bearer");
    policy.RequireAuthenticatedUser();
    policy.RequireRole("Admin");
    policy.RequireClaim("editor", "contents");
  });
}
```

Each policy added to the middleware has a name, which is used to reference the policy within the Authorize attribute on the controller class:

```
[Authorize(Policy = "ContentsEditor")]
public IActionResult Save(Article article)
{
  // ...
}
```

Delivering Modern Apps

Presented in partnership with Magenic, Modern Apps Live! brings Development Managers, Software Architects and Development Leads together to break down the complex landscape of mobile, cross-platform, and cloud development and learn how to architect, design and build a complete Modern Application from start to finish.

In-depth and educational sessions taught by the industry's top thought leaders will lay out how to get an app done successfully and at a low cost!

REGISTER NOW

REGISTER BY OCT 13 AND SAVE $300!*

Use promo code ORLOCT1

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

A Part of Live! 360: The Ultimate Education Destination

5 GREAT CONFERENCES, 1 GREAT PRICE

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

MODERNAPPSLIVE.COM

EVENT PARTNERS: Microsoft, Magenic

PLATINUM SPONSORS: okta, Google Cloud

SUPPORTED BY: Visual Studio Magazine, msdn magazine, Redmond Channel Partner, Redmond Magazine, Virtualization & Cloud Review, Visual Studio Magazine

PRODUCED BY: 1105 MEDIA

Figure 2 **Checking Policies Programmatically**

```
public class AdminController : Controller
{
  private IAuthorizationService _authorization;
  public AdminController(IAuthorizationService authorizationService)
  {
    _authorization = authorizationService;
  }

  public async Task<IActionResult> Save(Article article)
  {
    var allowed = await _authorization.AuthorizeAsync(
      User, "ContentsEditor"));
    if (!allowed)
      return new ForbiddenResult();

    // Proceed with the method implementation
    ...
  }
}
```

The Authorize attribute allows you to set a policy declaratively, but policies can also be invoked programmatically right from an action method, as shown in **Figure 2**.

If the programmatic check of permissions fails, you might want to return a ForbiddenResult object. Another option is returning ChallengeResult. In ASP.NET Core 1.x, returning a challenge tells the authorization middleware to return a 401 status code, or redirect the user to a login page, depending on configuration. The redirect won't happen in ASP.NET Core 2.0, however, and even in ASP.NET Core 1.x the challenge ends up in a ForbiddenResult if the user is already logged in. In the end, the best approach is to return ForbiddenResult if the permission check fails.

Note that you can even perform a programmatic check of the policies from within a Razor view, as shown in the code here:

```
@{
  var authorized = await Authorization.AuthorizeAsync(
    User, "ContentsEditor"))
}
@if (!authorized)
{
  <div class="alert alert-error">
    You're not authorized to access this page.
  </div>
}
```

For this code to work, however, you must first inject the dependency on the authorization service, as follows:

```
@inject IAuthorizationService Authorization
```

Figure 3 **Sample Authorization Handler**

```
public class ExperienceHandler :
             AuthorizationHandler<ExperienceRequirement>
{
  protected override Task HandleRequirementAsync(
    AuthorizationHandlerContext context,
    ExperienceRequirement requirement)
  {
    // Save User object to access claims
    var user = context.User;
    if (!user.HasClaim(c => c.Type == "EditorSince"))
      return Task.CompletedTask;

    var since = user.FindFirst("EditorSince").Value.ToInt();
    if (since >= requirement.Years)
      context.Succeed(requirement);

    return Task.CompletedTask;
  }
}
```

Using the authorization service in a view can help hide elements of the UI that shouldn't be within the reach of the current user given the current context. Keep in mind, though, that simply hiding options in the view is not enough. You always have to enforce policies in the controller, as well.

Custom Requirements

The stock requirements basically cover claims, authentication and provide a general-purpose mechanism for customization based on assertions, but you can create custom requirements, too. A policy requirement is made of two elements: a requirement class that holds just data, and an authorization handler that validates the data against the user. Custom requirements extend your ability to express specific policies. For example, let's say you want to extend the ContentsEditor policy by adding the requirement that the user must have at least three years of experience. Here's how you would do that:

```
public class ExperienceRequirement : IAuthorizationRequirement
{
  public int Years { get; private set; }

  public ExperienceRequirement(int minimumYears)
  {
    Years = minimumYears;
  }
}
```

A requirement must have at least one authorization handler. A handler is of type AuthorizationHandler<T>, where T is the requirement type. **Figure 3** illustrates a sample handler for the ExperienceRequirement type.

> Custom requirements extend your ability to express specific policies.

The sample authorization handler reads the claims associated with the user and checks for a custom EditorSince claim. If that isn't found, the handler returns without success. Success is returned only if the claim exists and contains an integer value not less than the specified number of years.

The custom claim is expected to be a piece of information linked in some way to the user—for example, a column in the Users table—saved to the authentication cookie. However, once you hold a reference to the user, you can always find the username from the claims and run a query against any database or external service to get hold of the years of experience and use the information in the handler. (I'll admit that this example would be a bit more realistic had the EditorSince value held a DateTime, and calculated if a given number of years had passed since the user began as an Editor.)

An authorization handler calls the method Succeed, passing the current requirement to notify that the requirement has been successfully validated. If the requirement didn't pass, the handler doesn't need to do anything and can just return. However, if the

handler wants to determine the failure of a requirement regardless of the fact that other handlers on the same requirement may succeed, it calls the method Fail on the authorization context object.

Here's how you add a custom requirement to the policy (keep in mind, as this is a custom requirement, you have no extension method; rather, you must proceed through the Requirements collection of the policy object):

```
services.AddAuthorization(options =>
{
  options.AddPolicy("AtLeast3Years",
    policy => policy
      .Requirements
      .Add(new ExperienceRequirement(3)));
});
```

In addition, you have to register the new handler with the DI system under the scope of the IAuthorizationHandler type:

```
services.AddSingleton<IAuthorizationHandler, ExperienceHandler>();
```

As mentioned, a requirement can have multiple handlers. When multiple handlers are registered with the DI system for the same requirement for the authorization layer, it suffices that at least one succeeds.

Accessing the Current HTTP Context

In the implementation of the authorization handler, you might need to inspect request properties or route data, like this:

```
if (context.Resource is AuthorizationFilterContext mvc)
{
  var url = mvc.HttpContext.Request.GetDisplayUrl();
  ...
}
```

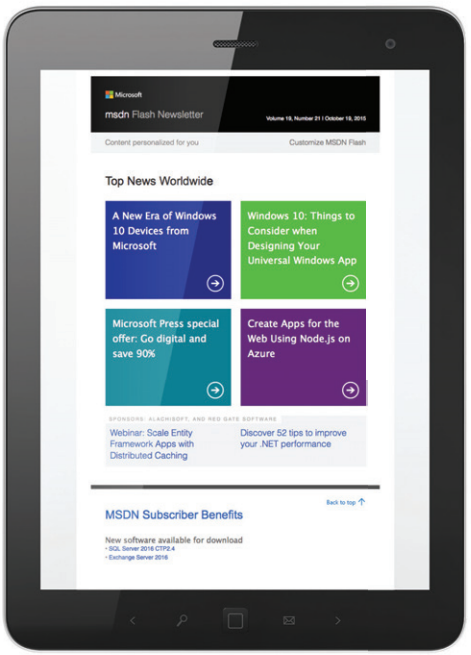
In ASP.NET Core, the AuthorizationHandlerContext object exposes a Resource property set to the filter context object. The context object is different depending on the framework involved. For example, MVC and SignalR send their own specific object. Whether you cast depends on what you need to access. For example, the User information is always there, so you don't need to cast for that, but if you want MVC-specific details such as routing information, then you have to cast.

Wrapping Up

In ASP.NET Core authorization comes in two flavors. One is traditional role-based authorization, which works the same way it does in classic ASP.NET MVC, and still has the structural limitation of being rather flat and not ideal for expressing sophisticated authorization logic. Policy-based authentication is a new approach that provides a richer and more expressive model. This is because a policy is a collection of requirements based on claims and custom logic based on any other information that can be injected from the HTTP context or external sources. These requirements are each associated with one or more handlers, which are responsible for the actual evaluation of the requirement. ■

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 technical experts for reviewing this article: *Barry Dorrans (Microsoft) and Steve Smith*



Get news from MSDN in your inbox!

Sign up to receive **MSDN FLASH**, which delivers the latest resources, SDKs, downloads, partner offers, security news, and updates on national and local developer events.

msdn magazine

msdn.microsoft.com/flashnewsletter

JAMES MCCAFFREY

Time-Series Regression Using a C# Neural Network

The goal of a time-series regression problem is to make predictions based on historical time data. For example, if you have monthly sales data (over the course of a year or two), you might want to predict sales for the upcoming month. Time-series regression is usually very difficult, and there are many different techniques you can use.

In this article, I'll demonstrate how to perform a time-series regression analysis using rolling-window data combined with a neural network. The idea is best explained by an example. Take a look at the demo program in **Figure 1**. The demo program analyzes the number of airline passengers who traveled each month between January 1949 and December 1960.

The demo data comes from a well-known benchmark dataset that you can find in many places on the Internet and is included with the download that accompanies this article. The raw data looks like:

```
"1949-01";112
"1949-02";118
"1949-03";132
"1949-04";129
"1949-05";121
"1949-06";135
"1949-07";148
"1949-08";148
...
"1960-11";390
"1960-12";432
```

There are 144 raw data items. The first field is the year and month. The second field is the total number of international airline passengers for the month, in thousands. The demo creates training data using a rolling window of size 4 to yield 140 training items. The training data is normalized by dividing each passenger count by 100:

```
[  0]   1.12   1.18   1.32   1.29   1.21
[  1]   1.18   1.32   1.29   1.21   1.35
[  2]   1.32   1.29   1.21   1.35   1.48
[  3]   1.29   1.21   1.35   1.48   1.48
...
[139]   6.06   5.08   4.61   3.90   4.32
```

Notice that the explicit time values in the data are removed. The first window consists of the first four passenger counts (1.12, 1.18, 1.32, 1.29), which are used as predictor values, followed by the fifth count (1.21), which is a value to predict. The next window consists of the second through fifth counts (1.18, 1.32, 1.29, 1.21), which are the next set of predictor values, followed by the sixth count (1.35), the value to predict. In short, each set of four consecutive passenger counts is used to predict the next count.

The demo creates a neural network with four input nodes, 12 hidden processing nodes and a single output node. The number of input nodes corresponds to the number of predictors in the rolling window. The window size must be determined by trial and error, which is the biggest drawback to this technique. The number of neural network hidden nodes must also be determined by trial and error, which is always true for neural networks. There's just one output node because time-series regression predicts one time unit ahead.

The neural network has $(4 * 12) + (12 * 1) = 60$ node-to-node weights and $(12 + 1) = 13$ biases, which essentially define the neural network model. Using the rolling-window data, the demo program trains the network using the basic stochastic back-propagation algorithm with a learning rate set to 0.01 and a fixed number of iterations set to 10,000.

```
file:///C:/NeuralTimeSeries/bin/Debug/NeuralTimeSeries.EXE

Begin neural network times series demo
Goal is to predict airline passengers over time
Data from January 1949 to December 1960

Normalized training data:
[  0]   1.12   1.18   1.32   1.29   1.21
[  1]   1.18   1.32   1.29   1.21   1.35
[  2]   1.32   1.29   1.21   1.35   1.48
[  3]   1.29   1.21   1.35   1.48   1.48
[  4]   1.21   1.35   1.48   1.48   1.36
. . .
[139]   6.06   5.08   4.61   3.90   4.32

Creating a 4-12-1 neural network

Setting maxEpochs = 10000
Setting learnRate = 0.01

Starting training
epoch = 2000  error = 0.0439
epoch = 4000  error = 0.0333
epoch = 6000  error = 0.0353
epoch = 8000  error = 0.0304
Done

Final neural network model weights and biases:

 1.41  0.83 -5.11  0.96  0.48  0.91  3.41  1.18 -3.00  1.95
 0.85  1.09 -0.09 -0.13 -1.68 -0.54 -1.61 -1.25  3.12 -0.04
 1.92 -0.23 -0.41 -0.13 -1.23 -0.39  3.93 -0.11  0.49 -0.38
-3.12 -0.78  0.51 -1.84 -0.27 -0.67  0.49  0.39  1.95  0.60
 1.28  1.24 -2.02  0.14  1.16  0.74  1.04  0.20 -1.50 -1.30
 1.66 -3.48 -1.23 -3.21 -2.17 -1.71 -2.44 -1.05 -5.08 -1.55
 0.17  0.20  0.70  0.19  0.36  0.39  0.55  0.21  0.51  0.30
 0.42  0.18  3.38

Model accuracy (+/- 30) on training data = 0.9143

Predicted passengers for January 1961 (t=145):
433

End time series demo
```

Figure 1 **Rolling-Window Time-Series Regression Demo**

Code download available at msdn.com/magazine/1017magcode.

During training, the demo displays the mean squared error between predicted output values and correct output values, every 2,000 iterations. Training error is difficult to interpret and is monitored mostly to see if something really strange happens (which is fairly common). In this case, the error seems to stabilize after about 4,000 iterations.

After training, the demo code displays the 73 weights and biases values, again mostly as a sanity check. For time-series regression problems, you must typically use a custom accuracy metric. Here, a correct prediction is one where the unnormalized predicted passenger count is plus or minus 30 from the actual count. With that definition, the demo program achieved 91.43 percent accuracy, which is 128 correct and 12 wrong for the 140 predicted passenger counts.

The demo concludes by using the trained neural network to predict the passenger count for January 1961, the first time period past the range of the training data. This is called extrapolation. The prediction is 433 passengers. That value could be used as a predictor variable to forecast February 1961, and so on.

This article assumes you have intermediate or higher programming skills and have a basic knowledge of neural networks, but doesn't assume you know anything about time-series regression. The demo program is coded using C#, but you shouldn't have too much trouble refactoring the code to another language, such as Java or Python. The demo program is too long to present in its entirety, but the complete source code is available in the file download that accompanies this article.

Time-Series Regression

Time-series regression problems are often displayed using a line chart such as the one in **Figure 2**. The blue line indicates the 144 actual, unnormalized, passenger counts in thousands, from January 1949 through December 1960. The light red line indicates the predicted passenger counts generated by the neural network time-series model. Notice that because the model uses a rolling window with four predictor values, the first predicted passenger count doesn't occur until month = 5. Additionally, I made forecasts for nine months beyond the range of the training data. These are indicated by the dashed red line.

In addition to making predictions for times beyond the training data range, time-series regression analyses can be used to identify anomalous data points. This doesn't occur with the demo passenger count data—you can see the predicted counts match the actual counts quite closely. For example, the actual passenger count for month t = 67 is 302 (the blue dot near the center in **Figure 2**) and the predicted count is 272. But suppose the actual count for month t = 67 was 400. There'd be an obvious visual indication that the actual count for month 67 was an outlier value.

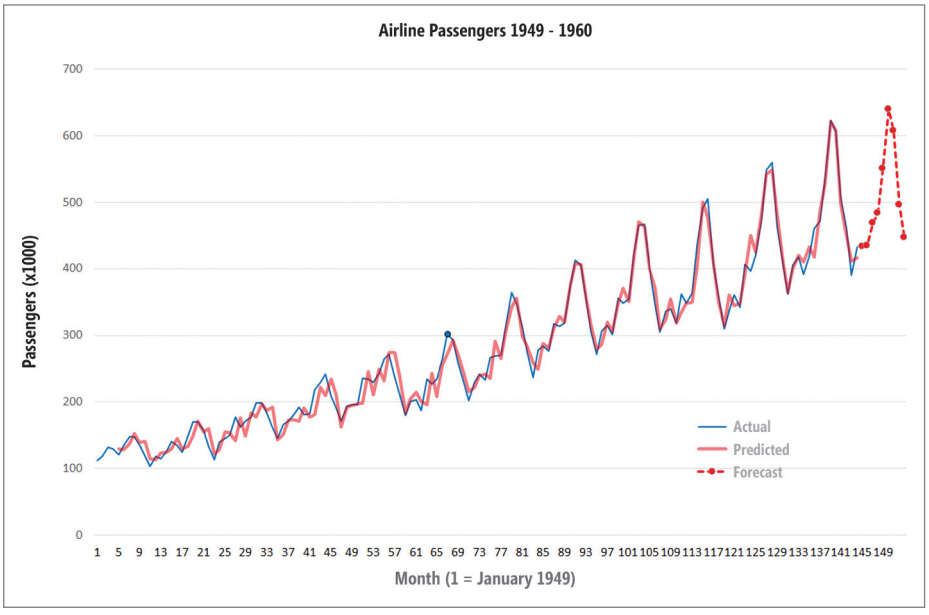


Figure 2 **Time-Series Regression Line Chart**

You can also use a programmatic approach for spotting anomalous data with time-series regression. For example, you could flag any time value where the actual data value and the predicted value differed by more than some fixed threshold, such as four times the standard deviation of the predicted versus actual data values.

The Demo Program

To code the demo program, I launched Visual Studio and created a new C# console application and named it NeuralTimeSeries. I used Visual Studio 2015, but the demo program has no significant .NET Framework dependencies, so any recent version will work fine.

In addition to making predictions for times beyond the training data range, time-series regression analyses can be used to identify anomalous data points.

After the template code loaded into the editor window, I right-clicked on file Program.cs in the Solution Explorer window and renamed the file to NeuralTimeSeriesProgram.cs, then allowed Visual Studio to automatically rename class Program for me. At the top of the template-generated code, I deleted all unnecessary using statements, leaving just the one that references the top-level System namespace.

The overall program structure, with a few minor edits to save space, is presented in **Figure 3**.

Figure 3 **NeuralTimeSeries Program Structure**

```
using System;
namespace NeuralTimeSeries
{
  class NeuralTimeSeriesProgram
  {
    static void Main(string[] args)
    {
      Console.WriteLine("Begin times series demo");
      Console.WriteLine("Predict airline passengers ");
      Console.WriteLine("January 1949 to December 1960 ");

      double[][] trainData = GetAirlineData();
      trainData = Normalize(trainData);
      Console.WriteLine("Normalized training data:");
      ShowMatrix(trainData, 5, 2, true);  // first 5 rows

      int numInput = 4; // Number predictors
      int numHidden = 12;
      int numOutput = 1; // Regression

      Console.WriteLine("Creating a " + numInput + "-" + numHidden +
        "-" + numOutput + " neural network");
      NeuralNetwork nn = new NeuralNetwork(numInput, numHidden,
        numOutput);

      int maxEpochs = 10000;
      double learnRate = 0.01;
      double[] weights = nn.Train(trainData, maxEpochs, learnRate);

      Console.WriteLine("Model weights and biases: ");
      ShowVector(weights, 2, 10, true);

      double trainAcc = nn.Accuracy(trainData, 0.30);
      Console.WriteLine("\nModel accuracy (+/- 30) on training " +
        "data = " + trainAcc.ToString("F4"));

      double[] future = new double[] { 5.08, 4.61, 3.90, 4.32 };
      double[] predicted = nn.ComputeOutputs(future);
      Console.WriteLine("January 1961 (t=145): ");
      Console.WriteLine((predicted[0] * 100).ToString("F0"));

      Console.WriteLine("End time series demo ");
      Console.ReadLine();
    } // Main

    static double[][] Normalize(double[][] data) { . . }

    static double[][] GetAirlineData() {. . }

    static void ShowMatrix(double[][] matrix, int numRows,
      int decimals, bool indices) { . . }

    static void ShowVector(double[] vector, int decimals,
      int lineLen, bool newLine) { . . }

  public class NeuralNetwork { . . }
} // ns
```

The demo uses a simple single-hidden-layer neural network, implemented from scratch. Alternatively, you can use the techniques presented in this article along with a neural network library such as Microsoft Cognitive Toolkit (CNTK).

The demo begins by setting up the training data, as shown in **Figure 4**.

Here, the rolling-window data is hardcoded with a window size of 4. Before writing the time-series program, I wrote a short utility program to generate the rolling-window data from the raw data. In most non-demo scenarios you'd read raw data from a text file, and then programmatically generate rolling-window data, where the window size is parameterized so you could experiment with different sizes.

The Normalize method just divides all data values by a constant 100. I did this purely for practical reasons. My first attempts with non-normalized data led to very poor results, but after normalization, my results were much better. In theory, when working with neural networks, your data doesn't need to be normalized, but in practice normalization often makes a big difference.

Figure 4 **Setting up the Training Data**

```
double[][] trainData = GetAirlineData();
trainData = Normalize(trainData);
Console.WriteLine("Normalized training data:");
ShowMatrix(trainData, 5, 2, true);

Method GetAirlineData is defined as:

static double[][] GetAirlineData()
{
  double[][] airData = new double[140][];
  airData[0] = new double[] { 112, 118, 132, 129, 121 };
  airData[1] = new double[] { 118, 132, 129, 121, 135 };
...
  airData[139] = new double[] { 606, 508, 461, 390, 432 };
  return airData;
}
```

The neural network is created like so:

```
int numInput = 4;
int numHidden = 12;
int numOutput = 1;
NeuralNetwork nn =
  new NeuralNetwork(numInput, numHidden, numOutput);
```

The number of input nodes is set to four because each rolling window has four predictor values. The number of output nodes is set to one because each set of window values is used to make a prediction for the next month. The number of hidden nodes is set to 12 and was determined by trial and error.

The neural network is trained and evaluated with these statements:

```
int maxEpochs = 10000;
double learnRate = 0.01;
double[] weights = nn.Train(trainData, maxEpochs, learnRate);
ShowVector(weights, 2, 10, true);
```

The Train method uses basic back-propagation. There are many variations, including using momentum or adaptive learning rates to increase training speed, and using L1 or L2 regularization or drop-out to prevent model over-fitting. The helper method ShowVector displays a vector with real values formatted to 2 decimals places, 10 values per line.

> My first attempts with non-normalized data led to very poor results, but after normalization, my results were much better.

After the neural network time-series model has been created, its prediction accuracy is evaluated:

```
double trainAcc = nn.Accuracy(trainData, 0.30);
Console.WriteLine("\nModel accuracy (+/- 30) on " +
  " training data = " + trainAcc.ToString("F4"));
```

For time-series regression, deciding whether a predicted value is correct or not depends on the problem being investigated. For the airline passenger data, method Accuracy marks a predicted passenger count as correct if the unnormalized predicted count is plus or minus 30 of the actual raw count. For the demo data, the first five predictions, for t = 5 to t = 9 are correct, but the prediction for t = 10 is incorrect:

```
 t  actual  predicted
= = = = = = = = = = =
 5   121     129
 6   135     128
 7   148     137
 8   148     153
 9   136     140
10   119     141
```

The demo program finishes by using the last four passenger counts (t = 141 to 144) to predict the passenger count for the first time period beyond the range of the training data (t = 145 = January 1961):

```
double[] predictors = new double[] { 5.08, 4.61, 3.90, 4.32 };
double[] forecast = nn.ComputeOutputs(predictors);
Console.WriteLine("Predicted for January 1961 (t=145): ");
Console.WriteLine((forecast[0] * 100).ToString("F0"));
Console.WriteLine("End time series demo");
```

Notice that because the time-series model was trained using normalized data (divided by 100), the predictions will also be normalized, so the demo displays the predicted values times 100.

Neural Networks for Time-Series Analyses

When you define a neural network you must specify the activation functions used by the hidden-layer nodes and by the output-layer nodes. Briefly, I recommend using the hyperbolic tangent (tanh) function for hidden activation, and the identity function for output activation.

When using a neural network library or system such as Microsoft CNTK or Azure Machine Learning, you must explicitly specify the activation functions. The demo program hardcodes these activation functions. The key code occurs in method ComputeOutputs. The hidden node values are computed like so:

```
for (int j = 0; j < numHidden; ++j)
  for (int i = 0; i < numInput; ++i)
    hSums[j] += this.iNodes[i] * this.ihWeights[i][j];

for (int i = 0; i < numHidden; ++i)  // Add biases
  hSums[i] += this.hBiases[i];

for (int i = 0; i < numHidden; ++i)   // Apply activation
  this.hNodes[i] = HyperTan(hSums[i]); // Hardcoded
```

Here, function HyperTan is program-defined to avoid extreme values:

```
private static double HyperTan(double x) {
  if (x < -20.0) return -1.0; // Correct to 30 decimals
  else if (x > 20.0) return 1.0;
  else return Math.Tanh(x);
}
```

A reasonable, and common, alternative to using tanh for hidden-node activation is to use the closely related logistic sigmoid function. For example:

```
private static double LogSig(double x) {
  if (x < -20.0) return 0.0; // Close approximation
  else if (x > 20.0) return 1.0;
  else return 1.0 / (1.0 + Math.Exp(x));
}
```

Because the identity function is just f(x) = x, using it for output-node activation is just a fancy way of saying don't use any explicit activation. The demo code in method ComputeOutputs is:

```
for (int j = 0; j < numOutput; ++j)
  for (int i = 0; i < numHidden; ++i)
    oSums[j] += hNodes[i] * hoWeights[i][j];

for (int i = 0; i < numOutput; ++i)  // Add biases
  oSums[i] += oBiases[i];

Array.Copy(oSums, this.oNodes, oSums.Length);
```

The sum of products for an output node is copied directly into the output node without applying an explicit activation. Note that the oNodes member of the NeuralNetwork class is an array with one cell, rather than a single variable.

The choice of activation functions affects the code in the back-propagation algorithm implemented in method Train. Method Train uses the calculus derivatives of each activation function. The derivative of y = tanh(x) is (1 + y) * (1 - y). In the demo code:

```
// Hidden node signals
for (int j = 0; j < numHidden; ++j) {
  derivative = (1 + hNodes[j]) * (1 - hNodes[j]); // tanh
  double sum = 0.0;
  for (int k = 0; k < numOutput; ++k)
    sum += oSignals[k] * hoWeights[j][k];
  hSignals[j] = derivative * sum;
}
```

If you use logistic sigmoid activation, the derivative of y = logsig(x) is y * (1 - y). For output activation, the calculus derivative of y = x is just the constant 1. The relevant code in method Train is:

```
for (int k = 0; k < numOutput; ++k) {
  errorSignal = tValues[k] - oNodes[k];
  derivative = 1.0;  // For Identity activation
  oSignals[k] = errorSignal * derivative;
}
```

Obviously, multiplying by 1 has no effect. I coded as I did to act as a form of documentation.

> A reasonable, and common, alternative to using tanh for hidden-node activation is to use the closely related logistic sigmoid function.

Wrapping Up

There are many different techniques you can use to perform time-series regression analyses. The Wikipedia article on the topic lists dozens of techniques, classified in many ways, such as parametric vs. non-parametric and linear vs. non-linear. In my opinion, the main advantage of using a neural network approach with rolling-window data is that the resulting model is often (but not always) more accurate than non-neural models. The main disadvantage of the neural network approach is that you must experiment with the learning rate to get good results.

Most time-series regression-analysis techniques use rolling-window data, or a similar scheme. However, there are advanced techniques that can use raw data, without windowing. In particular, a relatively new approach uses what's called a long short-term memory neural network. This approach often produces very accurate predictive models. ■

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

Thanks to the following Microsoft technical experts who reviewed this article: *John Krumm, Chris Lee and Adith Swaminathan*

2017 Orlando

ROYAL PACIFIC RESORT
AT UNIVERSAL ORLANDO
NOVEMBER 12-17

Visual Studio® LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS

Coding in Paradise

Grab your flip flops, and your laptops, and make plans to attend Visual Studio Live! (VSLive!™), the conference more developers rely on to expand their .NET skills and the ability to build better applications.

Over six full days of unbiased and cutting-edge education on the Microsoft Platform, developers, engineers, designers, programmers and more will soak in the knowledge on everything from Visual Studio and the .NET framework, to AngularJS, ASP.NET and Xamarin.

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 SPONSORS
Google Cloud okta

SUPPORTED BY
Visual Studio msdn magazine Redmond Channel Partner Redmond Magazine

NEW: HANDS-ON LABS

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

Only $645 through October 13

Whether you are an

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

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

REGISTER NOW

REGISTER BY OCTOBER 13 & SAVE $300!*

Use promo code ORLOCT4

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

LIVE! 360

TECH EVENTS WITH PERSPECTIVE

5 GREAT CONFERENCES, 1 GREAT PRICE

Visual Studio Live! Orlando is part of Live! 360, the Ultimate Education Destination. This means you'll have access to four (4) other co-located events at no additional cost:

- Visual Studio LIVE! — EXPERT SOLUTIONS FOR .NET DEVELOPERS
- SQL Server LIVE! — TRAINING FOR DBAs AND IT PROS
- TECHMENTOR — IN-DEPTH TRAINING FOR IT PROS
- Office & SharePoint LIVE! — ON-PREMISE, CLOUD & CROSS-PLATFORM TRAINING
- Modern Apps LIVE! — MOBILE, CROSS-DEVICE & CLOUD DEVELOPMENT

Five (5) events and hundreds of sessions to choose from—mix and match sessions to create your own, custom event line-up—it's like no other conference available today!

TURN THE PAGE FOR MORE EVENT DETAILS →

VSLIVE.COM/ORLANDOMSDN

PRODUCED BY

VIRTUALIZATION & Cloud Review | Visual Studio MAGAZINE | 1105 MEDIA YOUR GROWTH. OUR BUSINESS.

Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS

AGENDA AT-A-GLANCE

ALM / DEVOPS | CLOUD COMPUTING | NATIVE CLIENT | SOFTWARE PRACTICES | VISUAL STUDIO / .NET FRAMEWORK | WEB CLIENT

START TIME	END TIME	NEW Full Day Hands-On Labs: Sunday, November 12, 2017	
9:00 AM	6:00 PM	**VSS01** Full Day Hands-On Lab: Busy Developer's HOL on Angular - *Ted Neward*	**VSS02** Full Day Hands-On Lab: From 0-60 in a day with

START TIME	END TIME	Pre-Conference Workshops: Monday, November 13, 2017		
8:30 AM	5:30 PM	**VSM01** Workshop: Distributed Cross-Platform Application Architecture - *Jason Bock & Rockford Lhotka*	**VSM02** Workshop: Add Intelligence to Your Solutions with AI, Bots, and More - *Brian Randell*	**VSM03** Workshop: Designing, Developing,
6:30 PM	8:00 PM	Dine-A-Round Dinner @ Universal CityWalk - 6:30pm - *Meet at Conference Registration Desk to walk over with the group*		

START TIME	END TIME	Day 1: Tuesday, November 14, 2017		
8:00 AM	9:00 AM	Visual Studio Live! KEYNOTE: To Be Announced		
9:15 AM	10:30 AM	**VST01** Front-end Web Development in 2017 for the Front-endally Challenged Microsoft Developer - *Chris Klug*	**VST02** Go Mobile with C#, Visual Studio, and Xamarin - *James Montemagno*	**VST03** Microservices with Azure Container Service & Service Fabric - *Vishwas Lele*
10:30 AM	11:00 AM	Networking Break • Visit the EXPO - *Pacifica 7*		
11:00 AM	12:00 PM	**LIVE! 360 KEYNOTE: To Be Announced** - *Pacifica 6*		
12:00 PM	12:45 PM	Lunch • Visit the EXPO		
12:45 PM	1:30 PM	Dessert Break • Visit the EXPO		
1:30 PM	2:45 PM	**VST05** ASP.NET Core MVC—What You Need to Know - *Philip Japikse*	**VST06** Optimizing and Extending Xamarin.Forms Mobile Apps - *James Montemagno*	**VST07** Tactical DevOps with Visual Studio Team Services - *Brian Randell*
2:45 PM	3:15 PM	Networking Break • Visit the EXPO - *Pacifica 7*		
3:15 PM	4:30 PM	**VST09** Angular(2)—The 75-Minute Crash Course - *Chris Klug*	**VST10** Building a Cross-Platform Mobile App Backend in the Cloud - *Nick Landry*	**VST11** Database Lifecycle Management and the SQL Server Database - *Brian Randell*
4:40 PM	5:00 PM	**VST13** Fast Focus: Aurelia vs. Just Angular - *Chris Klug*	**VST14** Fast Focus: Tips & Tricks for Xamarin Development - *James Montemagno*	**VST15** Fast Focus on Azure Functions
5:10 PM	5:30 PM	**VST17** Fast Focus: Web Security 101 - *Brock Allen*	**VST18** Fast Focus: Cross-Platform Code Reuse - *Rockford Lhotka*	**VST19** Fast Focus: Exploring Microservices in a Microsoft Landscape - *Marcel de Vries*
5:30 PM	7:30 PM	Exhibitor Reception - *Pacifica 7*		

START TIME	END TIME	Day 2: Wednesday, November 15, 2017		
8:00 AM	9:15 AM	**VSW01** User Authentication for ASP.NET Core MVC Applications - *Brock Allen*	**VSW02** Cloud Oriented Programming - *Vishwas Lele*	**VSW03** Overcoming the Challenges of Mobile Development in the Enterprise - *Roy Cornelissen*
9:30 AM	10:45 AM	**VSW05** Building AngularJS Component-Based Applications - *Miguel Castro*	**VSW06** Building Modern Web Apps with Azure - *Eric D. Boyd*	**VSW07** Creating a Release Pipeline with Team Services - *Esteban Garcia*
10:45 AM	11:30 AM	Networking Break • Visit the EXPO - *Pacifica 7*		
11:30 AM	12:30 PM	**LIVE! 360 KEYNOTE: To Be Announced** - *Pacifica 6*		
12:30 PM	1:30 PM	Birds-of-a-Feather Lunch • Visit the EXPO		
1:30 PM	2:00 PM	Dessert Break • Visit the EXPO		
2:00 PM	3:15 PM	**VSW09** Securing Web APIs in ASP.NET Core - *Brock Allen*	**VSW10** A/B Testing, Canary Releases and Dark Launching, Implementing Continuous Delivery on Azure - *Marcel de Vries*	**VSW11** The Zen of UI Automation Testing
3:15 PM	4:00 PM	Networking Break • Visit the EXPO • Expo Raffle @ 3:30 p.m. - *Pacifica 7*		
4:00 PM	5:15 PM	**VSW13** Build Object-Oriented Enterprise Apps in JavaScript with TypeScript	**VSW14** Lock the Doors, Secure the Valuables, and Set the Alarm - *Eric D. Boyd*	**VSW15** Unit Testing & Test-Driven Development (TDD) for Mere Mortals - *Benjamin Day*
8:00 PM	10:00 PM	Live! 360 Dessert Luau - *Wantilan Pavilion*		

START TIME	END TIME	Day 3: Thursday, November 16, 2017		
8:00 AM	9:15 AM	**VSH01** HTTP/2: What You Need to Know - *Robert Boedigheimer*	**VSH02** PowerApps and Flow Part II: Package, Embed, and Extend Your Applications - *Manas Maheshwari & Pratap Ladhani*	**VSH03** Exploring C# 7 New Features - *Adam Tuliper*
9:30 AM	10:45 AM	**VSH05** ASP.NET Tag Helpers - *Robert Boedigheimer*	**VSH06** Storyboarding 101 - *Billy Hollis*	**VSH07** .NET Standard—From Noob to Ninja - *Adam Tuliper*
11:00 AM	12:00 PM	Visual Studio Live! Panel: Native or Web—Does it Matter? - *Brian Randell (Moderator), Damian Brady, Jeremy Clark, Esteban Garcia, Billy Hollis, & Adam Tuliper*		
12:00 PM	1:00 PM	Lunch on the Lanai - *Lanai / Pacifica 7*		
1:00 PM	2:15 PM	**VSH09** I See You: Watching the User with Reactive Forms - *Deborah Kurata*	**VSH10** Continuous Integration and Deployment for Mobile Using Azure Services - *Kevin Ford*	**VSH11** Deploying Straight to Production: A Guide to the Holy Grail - *Damian Brady*
2:30 PM	3:45 PM	**VSH13** Angular Routing - *Deborah Kurata*	**VSH14** XAML Inception—Deep Composition for Better UI - *Billy Hollis*	**VSH15** Application Insights: Measure Your Way to Success - *Esteban Garcia*
4:00 PM	5:00 PM	Next? Visual Studio Live! Networking Event - *Brian Randell (Moderator), Damian Brady, Jeremy Clark, Esteban Garcia, Billy Hollis, & Deborah Kurata*		

START TIME	END TIME	Post-Conference Workshops: Friday, November 17, 2017	
8:00 AM	5:00 PM	**VSF01** Workshop: Angular Fundamentals - *John Papa*	**VSF02** Workshop: Building, Running & Microservices with Docker Containers on Azure

Speakers and sessions subject to change

Presented in Partnership with Magenic

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

LIVE! 360
TECH EVENTS WITH PERSPECTIVE

ROYAL PACIFIC RESORT AT UNIVERSAL ORLANDO
NOVEMBER 12-17

WEB SERVER	MODERN APPS LIVE!
Xamarin and Xamarin.Forms - *Roy Cornelissen*	
	Pre-Con Workshops: Monday, Nov. 13
Service Oriented Technologies: & Implementing WCF and the Web API - *Miguel Castro*	**MAM01** Workshop: Building Modern Mobile Apps - *Brent Edwards & Kevin Ford*
	Dine-A-Round Dinner
	Day 1: Tuesday, November 14, 2017
	Modern Apps Live! KEYNOTE PANEL: Industry Trends, Technology, and Your Career - *Matt Lockhart (Moderator)*
VST04 What's New in Visual Studio 2017 - *Robert Green*	**MAT01** Modern App Development: Transform How You Build Web and Mobile Software - *Rockford Lhotka*
	Networking Break • Visit the EXPO - *Pacifica 7*
	LIVE! 360 KEYNOTE
	Lunch • Visit the EXPO
	Dessert Break • Visit the EXPO
VST08 To Be Announced	**MAT02** Architecture: The Key to Modern App Success - *Brent Edwards*
	Networking Break • Visit the EXPO - *Pacifica 7*
VST12 Top 10 Entity Framework Core Features Every Developer Should Know - *Philip Japikse*	**MAT03** Focus on the User Experience #FTW - *Jim Barrett*
VST16 Fast Focus: Busting .NET Myths - *Jason Bock*	**MAT04** Fast Focus: Hybrid Web Frameworks - *Allen Conway*
VST20 Fast Focus: Dependency Injection in 20 Minutes - *Miguel Castro*	**MAT05** Fast Focus: Web Assembly - *Jason Bock*
	Exhibitor Reception - *Pacifica 7*
	Day 2: Wednesday, November 15, 2017
VSW04 Building Apps with Microsoft Graph and Visual Studio - *Robert Green*	**MAW01** Manage Distributed Teams with Visual Studio Team Services and Git - *Brian Randell*
VSW08 Bots are the New Apps: Building Bots with ASP.NET Web API & Language Understanding - *Nick Landry*	**MAW02** DevOps, Continuous Integration, the Cloud, and Docker - *Dan Nordquist*
	Networking Break • Visit the EXPO - *Pacifica 7*
	LIVE! 360 KEYNOTE
	Birds-of-a-Feather Lunch
	Dessert Break • Visit the EXPO
VSW12 To Be Announced	**MAW03** Security with Speed for Modern Developers - *Michael Lester*
	Networking Break • Visit the EXPO • Raffle @ 3:30 p.m.
VSW16 PowerApps, Flow, and Common Data Service: Empowering Businesses with the Microsoft Business Application Platform - *Charles Sterling*	**MAW04** Coding for Quality and Maintainability - *Jason Bock*
	Live! 360 Dessert Luau - *Wantilan Pavilion*
	Day 3: Thursday, November 16, 2017
VSH04 Top 10 Ways to Go from Good to Great Scrum Master - *Benjamin Day*	**MAH01** Modern Web Development: Building Server Side Using ASP.NET Core, MVC, Web API, and Azure - *Allen Conway*
VSH08 Devs vs. Ops: Making Friends with the Enemy - *Damian Brady*	**MAH02** Modern Web Development: Building Client Side Using TypeScript and Angular - *Allen Conway*
	Modern Apps Live! Panel: Mobile Development Technologies - *Rockford Lhotka (Moderator), James Montemagno, Kevin Ford*
	Lunch on the Lanai - *Lanai / Pacifica 7*
VSH12 Design Patterns: Not Just for Architects - *Jeremy Clark*	**MAH03** Using All That Data: Power BI to the Rescue - *Scott Diehl*
VSH16 DI Why? Getting a Grip on Dependency Injection - *Jeremy Clark*	**MAH04** Modern Mobile Development: Build a Single App For iOS, Android, and Windows with Xamarin Forms - *Kevin Ford*
	Next? Modern Apps Live! Networking Event - *Rockford Lhotka (Moderator)*
	Post-Con Workshops: Friday, Nov. 17
Continuously Deploying - *Marcel de Vries & Rene van Osnabrugge*	**MAF01** Workshop: Modern App Deep Dive—Xamarin, Responsive Web, UWP - *Kevin Ford, Brent Edwards, Allen Conway*

Check Out These Additional Sessions for Developers at Live! 360

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

Office & SharePoint Live! features 15+ developer sessions, including:

- **NEW!** Full Day Hands-On Lab: Developing Extensions for Microsoft Teams - *Paul Schaeflein*
- Workshop: Mastering the SharePoint Framework- *Andrew Connell*
- TypeScript for SharePoint Developers -*Rob Windsor*
- Building Office Add-ins for Outlook with Angular - *Andrew Connell*
- Developing SharePoint Framework Components Using Visual Studio - *Paul Schaeflein*
- What Every Developer Needs to Know about SharePoint Development Online or On-Prem - *Robert Bogue*
- Build a Complete Business Solution Using Microsoft Graph API through Client Side Web Parts - *Julie Turner*

SQL Server LIVE!
TRAINING FOR DBAs AND IT PROS

SQL Server Live! features 30+ developer sessions, including:

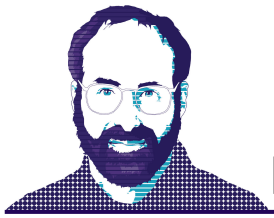
- **NEW!** Full Day Hands-On Lab: Developer Dive into SQL Server 2016 - *Leonard Lobel*
- Turbo Boost - SQL Tricks Everybody MUST Know - *Pinal Dave*
- Advanced SSIS Package Authoring with Biml - *Tim Mitchell*
- Graph DB Support in SQL Server 2017 - *Karen Lopez*
- Big Data Technologies: What, Where and How to Run Them on Azure - *Andrew Brust*
- Top Five SQL Server Query Tuning Tips - *Janis Griffin*
- Workshop: Big Data, BI, and Analytics on The Microsoft Stack - *Andrew Brust*

TECHMENTOR
IN-DEPTH TRAINING FOR IT PROS

TechMentor features 20 + developer sessions, including:

- **NEW!** Full Day Hands-On Lab: Ethical Hacking with Kali Linux - *Mike Danseglio & Avril Salter*
- Workshop: Windows Security—How I Do It! - *Sami Laiho*
- Hardware, Camtasia, and a Storyline: Creating Your Own User Training - *Greg Shields*
- Make Your PowerShell Scripts Bulletproof with Pester - *Melissa Januszko*
- Controlling Your Azure Spend - *Timothy Warner*
- PowerShell Scripting Secrets - *Jeffery Hicks*
- In-Depth Introduction to Docker - *Neil Peterson*

VSLIVE.COM/ORLANDOMSDN



A Measure of Displeasure

I've just solved a *major* problem in my never-ending crusade against sucky software. I can recognize sucky software, explain why it's bad, and what it should be instead. But I've never been able to quantify an app or Web site's level of suckiness. Until now.

Anyone who's done any sort of research knows that, "If you can't measure it, you can't manage it." If you don't have some sort of metric to quantify how well you're doing, you can't tell if an intervention you make was effective or not. That's why studies in the treatment of pain are so difficult. You can't just slap a meter on someone to see how much they're hurting.

So, how to quantify a bad UX? This app sucked, but how badly? More than this other app, or less? I tried some multi-factor rubrics: frequency of crashes, amount of data lost, length of learning curve, extra keystrokes over the required number and so on. I found all these metrics too complicated. A measurement for ease of use has to be easy to use itself, no?

The revelation hit me a month ago. I was trying to connect my new iPhone to Harvard's secure Wi-Fi network, which was a huge pain in the ass. I had to call tech support (on my Android phone) and work through it with them for about half an hour. The complexity and opaqueness of this supposedly easiest of devices astonished me, and I found myself saying "WTF?" again and again. Curmudgeon that I am, I kept track. By the time I'd finished the process and mercifully released the poor tech support agent, I'd said "WTF?" six separate times.

There's my metric. I hereby decree that henceforward, all bad software shall be rated on the PWS scale, which stands for Plattski's WTF Score. It's the number of times a user says "WTF?" when installing or using an app.

Like the Apgar score for newborn babies (see wikipedia.org/wiki/Apgar_score), the PWS is a snap to calculate and compare. Connecting my iPhone to Harvard's Wi-Fi network scored a six. Connecting my Android scored only a three. My Android was twice as easy to use as my iPhone. Who knew?

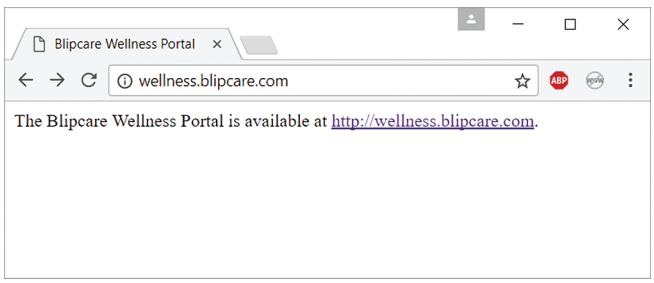


Figure 1 **Isn't wellness.blipcare.com Where I Already Am?**

I've found that the PWS works best at the extremes of its range. The difference between a five and a six isn't huge. But the difference between a three and an eight certainly is.

Here's another real example. I was setting up two different weight scales that automatically log their readings over Wi-Fi to Microsoft HealthVault. The first came from Blipcare, and its software stunk. For example, clicking the login button on the main site (blipcare.com), redirected me to wellness.blipcare.com (**Figure 1**), where the Web page greeted me with this gem: "The Blipcare Wellness Portal is available at http://wellness.blipcare.com." Whoa, isn't that where I already am? Didn't your own login button just send me here? WTF? (On further inspection, the link in **Figure 1** points to https://wellness.blipcare.com/portal/f?p=100. Without the portal suffix, you don't get the site. This WTF is well deserved.)

Muddling past that, the so-called wizard takes me to a page that offers to set up notification rules. I can be alerted by a High, Medium or Low level notification, signified by red, orange or green color if my weight "satisfies a defined rule." I just want to record my damn weight, not get a full TSA terrorism alert (bit.ly/2v2VF2w) when I eat a basket of onion rings. WTF?

I don't have space here to list them all, but by the time I'd gotten the Blipcare scale hooked up, I said WTF nine separate times. Nine! I declare this the maximum for any app. When the PWS hits 10, you throw the damn thing away. And the pox on anyone who would release an app that bad.

On the other hand, I found Fitbit's Aria scale pleasantly easy to connect. It didn't redirect me anywhere cryptic. The instructions showed a picture of a critical step, which Blipcare omitted. I said WTF only once during that installation.

I've been using the PWS ever since. Try it, and tell me what you think. The best I've found so far is Amazon's One-Click ordering system, which scores a PWS of zero. At least, until the bill comes. Then I say it a lot. ■

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.*

GrapeCity | Developer Solutions

Build Better Applications

For more information, visit GrapeCity.com or call +1-800-240-7500

Achieve more with GrapeCity Developer Solutions.

Developers seek out GrapeCity and our great line of products because we get the fast-moving pace of the developer world. We live, breathe, and work within the developer universe. Their community is our community.

We're trusted partners, problem solvers, and team members.

Because GrapeCity is comprised of developers, we understand the unique challenges developers and their enterprises face. Our team and tools empower development by decreasing time, money, and security risks so the focus remains on developing innovative business solutions. GrapeCity's high-quality, fast, flexible UI controls and solutions have small footprints, key features, and universal APIs – all designed to reduce the learning curve and enhance the end product. And with direct access to global product experts, developers are never alone in the goal to develop meaningful, intuitive applications for enterprises and beyond.

ComponentOne
.NET UI CONTROLS

ActiveReports
REPORTING SOLUTIONS

Spread
SPREADSHEET SOLUTIONS

Wijmo
JAVASCRIPT UI CONTROLS

© 2017 GrapeCity, Inc. All rights reserved. All other product and brand names are trademarks and/or registered trademarks of their respective holders.

SYNCFUSION
DATA PLATFORM

Unique, One-Stop Solution for All Your Data Science Needs

- Dashboard Platform
- Big Data Platform
- Data Integration Platform
- Report Platform

Take data from any source, transform it visually, build models, and deploy interactive dashboards with ease.

Before spending millions, try us!

- License your entire team for $4,000 or less per year.
- No per-user or per-server charges.
- Completely free Community License also available.

www.syncfusion.com/MSDNdata

Syncfusion®