THE MICROSOFT JOURNAL FOR DEVELOPERS

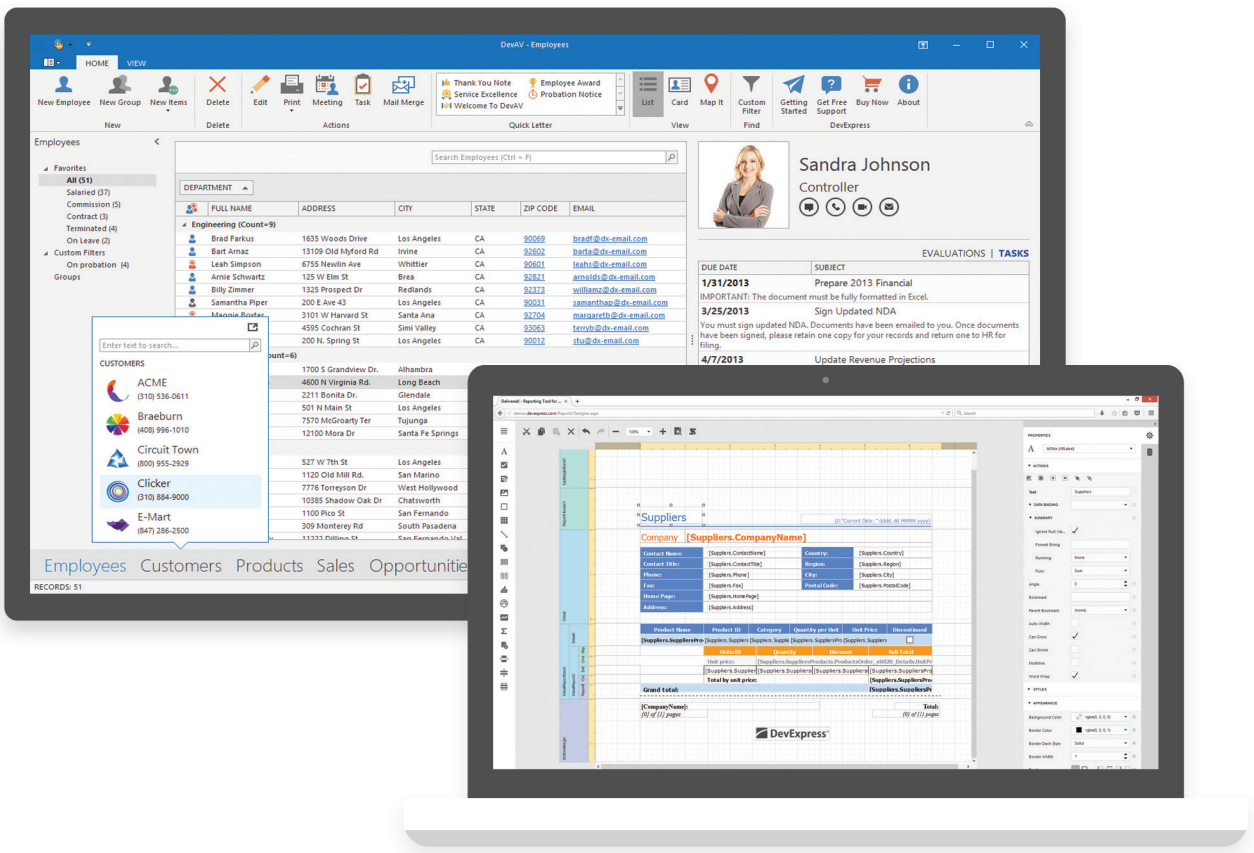
OCTOBER 2018 VOL 33 NO 10

msdn magazine

DevExpress®

Your Next Great App Starts Here

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



Experience the DevExpress Difference
Download Your Free 30-Day Trial Today

devexpress.com/trial

THE MICROSOFT JOURNAL FOR DEVELOPERS

OCTOBER 2018 VOL 33 NO 10

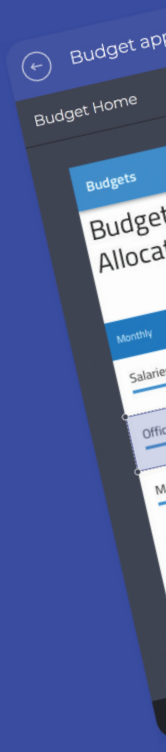
msdn magazine

COLUMNS

Microsoft

Indigo.Design

A unified platform for Visual Design, UX Prototyping, Code Generation & App Development



Design to Match Your Brand

Create best-in-class UI designs using the expressive Indigo Design System with Sketch UI Kits. With 50+ components, 30+ UI patterns and complete app scenarios, getting started is a breeze.

Share, Collaborate & Test

Import Sketch files into the cloud and share user flows and interactions on any device. Record & playback usability studies and get real-time reporting & analytics to see how users interact with your designs.

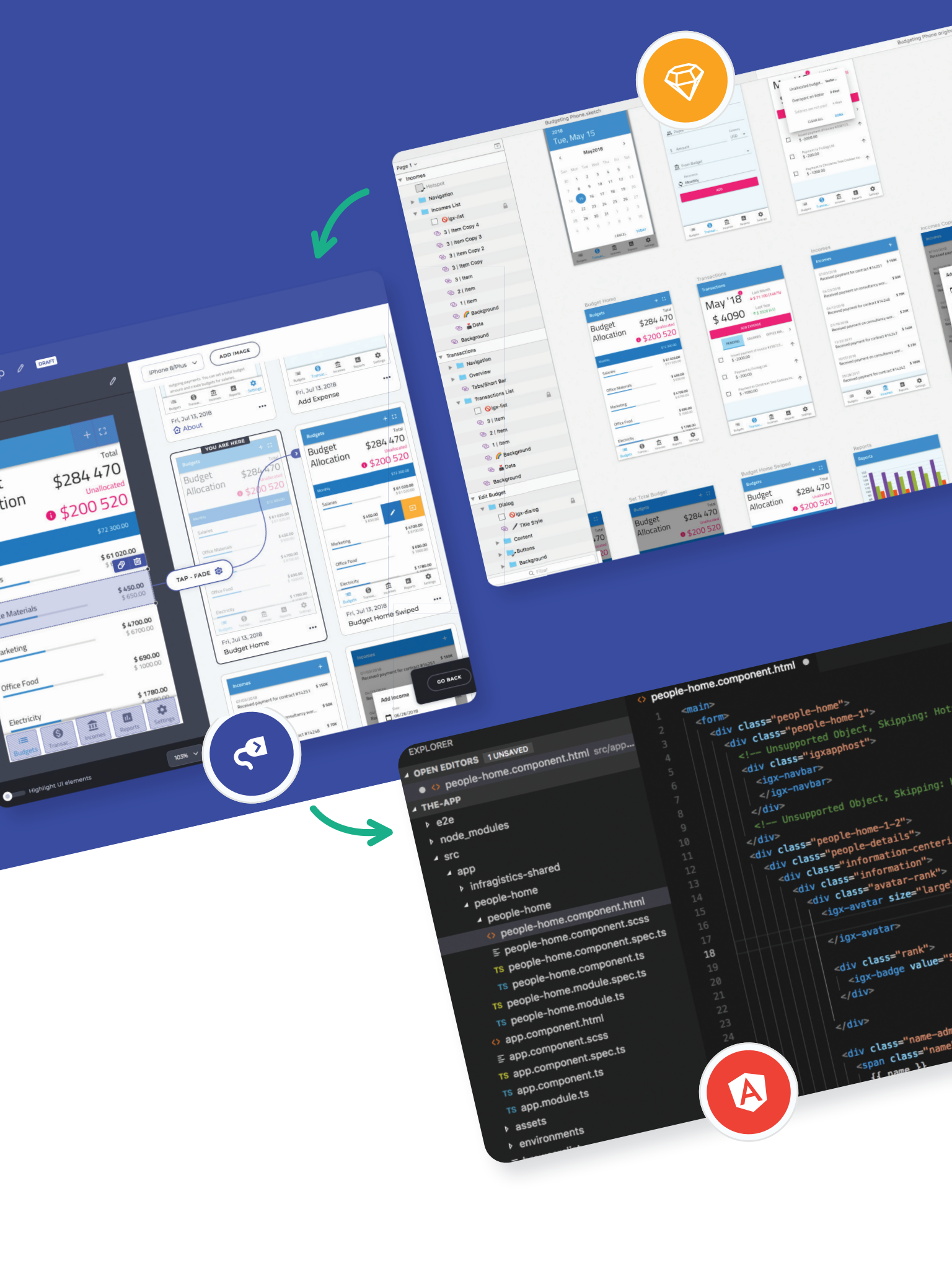
Runnable Code with 1-Click!

Everything you craft in Sketch from the design system matches to our Ignite UI for Angular components. With the click of a button, generate high-quality HTML, CSS, and Angular code from your design with no compromise.

Get it today for only $99/month: indigo.design

INFRAGISTICS®

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


Budgeting Phone.sketch
Page 1
Incomes
Navigation
Incomes List
Transactions
Edit Budget
Dialog
Budget Allocation
$284 470
$200 520
May '18
$ 4090
Tue, May 15
ADD IMAGE
iPhone 8/Plus
DRAFT
YOU ARE HERE
TAP - FADE
GO BACK
Budget Home
Budget Home Swiped
Add Expense
Add Income
About
Highlight UI elements
EXPLORER
OPEN EDITORS 1 UNSAVED
people-home.component.html src/app...
THE-APP
e2e
node_modules
src
app
infragistics-shared
people-home
people-home.component.html
people-home.component.scss
people-home.component.spec.ts
people-home.component.ts
people-home.module.spec.ts
people-home.module.ts
app.component.html
app.component.scss
app.component.spec.ts
app.component.ts
app.module.ts
assets
environments
people-home.component.html
<main>
<form>

<!-- Unsupported Object, Skipping: Hot

<igx-navbar>
</igx-navbar>

<!-- Unsupported Object, Skipping:

<div class="information-centeri

<igx-avatar size="large"
</igx-avatar>

<igx-badge value="

<div class="name-adr
<span class="name
{{ name }}

msdn magazine

OCTOBER 2018 VOLUME 33 NUMBER 10

ID STATEMENT MSDN Magazine (ISSN 1528-4859) is published 13 times a year, monthly with a special issue in November by 1105 Media, Inc., 6300 Canoga Avenue, Suite 1150, Woodland Hills, CA 91367. Periodicals postage paid at Woodland Hills, CA 91367 and at additional mailing offices. Annual subscription rates payable in US funds are: U.S. $35.00, International $60.00. Annual digital subscription rates payable in U.S. funds are: U.S. $25.00, International $25.00. Single copies/back issues: U.S. $10, all others $12. Send orders with payment to: MSDN Magazine, P.O. Box 3167, Carol Stream, IL 60132, email MSDNmag@1105service.com or call 866-293-3194 or 847-513-6011 for U.S. & Canada; 00-1-847-513-6011 for International, fax 847-763-9564. POSTMASTER: Send address changes to MSDN Magazine, P.O. Box 2166, Skokie, IL 60076. Canada Publications Mail Agreement No: 40612608. Return Undeliverable Canadian Addresses to Circulation Dept. or XPO Returns: P.O. Box 201, Richmond Hill, ON L4B 4R5, Canada.

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

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

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

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

LEADTOOLS® V20

WEB MOBILE DESKTOP SERVER

WHAT DOES YOUR CODE LOOK LIKE?

SEARCH ocrdemo.cs

```cs
using System;
using Leadtools;
using Leadtools.Ocr;
using Leadtools.Document.Writer;

namespace LEADOcrSimpleDemo
{
   class Program
   {
      static void Main(string[] args)
      {
         //Set the LEADTOOLS evaluation license
         RasterSupport.SetLicense(@"License\LEADTOOLS.LIC",
            System.IO.File.ReadAllText(@"License\LEADTOOLS.LIC.KEY"));
         //Specify input and output parameters
         string inputFile = @"C:\Users\Public\Documents\LEADTOOLS Images\OCR1.TIF";
         string outputFile = @"C:\Users\Public\Documents\LEADTOOLS Images\OCR1.PDF";

         //Call OCRImage method
         OCRImage(inputFile, outputFile);
      }

      static void OCRImage(string inputFile, string outputFile)
      {
         Console.WriteLine($"Loading and recognizing {inputFile}");

         //Initiate the LEADTOOLS OCR Engine
         using (IOcrEngine ocrEngine = OcrEngineManager.CreateEngine(OcrEngineType.LEAD, false))
         {
            //Startup the LEADTOOLS OCR Engine
            ocrEngine.Startup(null, null, null, null);
            //Run the AutoRecognizeManager and specify PDF format
            ocrEngine.AutoRecognizeManager.Run(inputFile, outputFile, DocumentFormat.Pdf, null, null);
            Console.WriteLine($"OCR output saved to {outputFile}");
         }
      }
   }
```

Above is an entire OCR application using LEADTOOLS

That's it. Clean and simple code that produces a fast and accurate OCR app backed by the most powerful OCR SDK. Download our free evaluation and see how easy it is to use LEADTOOLS for your document, medical, and multimedia projects.

Microsoft® .NET · X · Windows · Java · Linux · iOS · macOS · Android · HTML5

Get Started Today

DOWNLOAD OUR FREE EVALUATION

LEADTOOLS.COM

MICHAEL DESMOND

Open Soars

When Mark Michaelis approached me about writing an article on how to contribute to Microsoft open source software (OSS) projects, I was a bit skeptical. After all, *MSDN Magazine* is committed to publishing code-level how-to articles for working developers engaged with the Microsoft tool stack. The other articles in this issue have titles like "Deploying to Azure App Service and Azure Functions," "Logging SQL and Change-Tracking Events in EF Core" and "Face Detection Using the Eigenfaces Algorithm on the GPU."

A couple things swayed my thinking. First, Michaelis stands out as one of our most successful and valuable authors. His articles have consistently been among the most widely read in the magazine, and if he thinks a Microsoft OSS story is a good idea, I'm going to hear him out. Second, the idea was timely given the clear and deepening commitment Microsoft has made to OSS over the years, and perhaps most notably on June 4, 2018, when Microsoft purchased GitHub.

In his article, Michaelis provides a roadmap for developers looking to contribute to Microsoft OSS projects.

I thought it telling that Microsoft Chief Executive Officer Satya Nadellah took a moment in his blog post announcing the acquisition to offer a plea to those perhaps watching the GitHub acquisition with a jaundiced eye (bit.ly/2Npif2Q). He wrote:

"We have been on a journey with open source, and today we are active in the open source ecosystem, we contribute to open source projects, and some of our most vibrant developer tools and frameworks are open source. When it comes to our commitment to open source, judge us by the actions we have taken in the recent past, our actions today, and in the future."

It's a fair ask, and one Michaelis seemed aware of when I reached out to him for this column. He said that the sentiment toward Microsoft among open source developers, which once was broadly negative, has "turned around." The numbers tell the story: At the moment of this writing, Microsoft hosts 1,971 OSS repositories on GitHub, including strategic technologies like .NET Framework, .NET Core and Xamarin.

In his article, Michaelis provides a roadmap for developers looking to contribute to Microsoft OSS projects. He addresses how Microsoft encourages newcomers to get involved, and provides insight into the mechanics of Microsoft-hosted OSS projects. For young coders eager to contribute, Michaelis says the first step is to become a Git expert. From there, he says, developers can worry about finding a good technical fit and advancing their personal and professional goals.

Enlisting an employer can sometimes make sense, says Michaelis: "The easiest thing is to ensure that the OSS project, and specifically the feature or bug you're working on, is something that the company needs. If you have alignment, getting company support is relatively easy," Michaelis says.

But he cautions that aligning with your day job may be a poor fit if you hope to build technical cred in an OSS project without regard for company objectives and projects. It may be best to do the side work on your own time, rather than your company's.

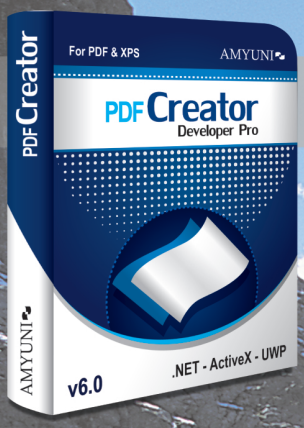
"Perhaps some more realistic goals are, submit changes that you can point to when interviewing, fix an issue or feature that you need for your own work, or focus on an area that you need to improve your understanding anyway," Michaelis says.

Ultimately, his advice to those looking to make a mark in the OSS space is simple: "Be passionate and willing to burn the midnight oil, as you would expect of anyone trying to rise to the top."

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

Switch to Amyuni® PDF

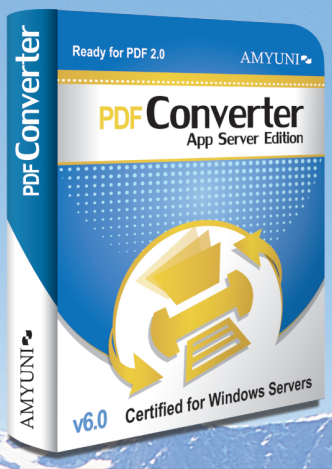
AMYUNI



Create and Process PDFs in .NET, COM/ActiveX and UWP

NEW v6.0

- Edit, process and print PDF 2.0 documents
- Create, fill-out and annotate PDF forms with Javascript support
- Fast and lightweight 64-bit components
- Universal Apps DLLs enable publishing C#, C++, CX or JS apps to Windows Store
- Plus a number of exciting features in v6.0



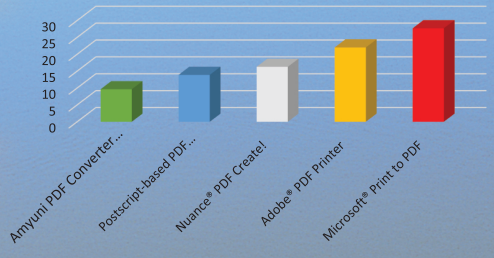
High Performance PDF Printer for Desktops and Servers

Print to PDF in a fraction of the time needed with other tools. WHQL certified for all Windows platforms. Version 6.0 updated for Server 2016.

Complete Suite of PDF, XPS and DOCX 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 and DOCX printer driver
- Export PDF documents into other formats such as images, HTML5 or editable DOCX
- Import/Export XPS files using our native libraries

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



Other Developer Components from Amyuni®

- 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 3 engine for accurate character recognition
- Javascript Engine: Integrate a full JS interpreter into your PDF processing applications
- USB Mobile Monitor: Mirror the display of your Windows or Linux system onto your Android device

Windows Server 2016 Certified

Windows Server 2012 Certified

Windows

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

JULIE LERMAN

Logging SQL and Change-Tracking Events in EF Core

The addition of the flexible logging APIs in .NET Core has been a boon to development. EF Core ties into those APIs, allowing you to expose a variety of logging and debugging information coming from EF Core.

In this month's column, I want to show you some of the types of information you can get from EF Core at run time, a few of the ways to surface that information and, as usual, pass along various tips and tricks I've learned along the way.

Although the logging capability is available for anything that sits on top of the .NET Core SDK, you'll find that most of the documentation is for how ASP.NET Core uses it. See, for example, the official Microsoft doc on ASP.NET Core Logging at bit.ly/2OiS4HD.

> In order to surface EF Core logging information, you need to tie an ILoggerFactory to a DbContext.

You can target the output to a variety of destinations, thanks to the ILoggerFactory, which can use built-in providers such as the ConsoleLoggerProvider and the DebugLoggerProvider, and can even integrate with Windows events, AzureAppServices and more. The providers are encapsulated in various extensions, such as Microsoft.Extensions.Logging.Console. Because my goal here is to show you the type of information EF Core can expose, not how to integrate with the various providers, I do recommend looking at the aforementioned ASP.NET Core Logging article, as well as Mark Michaelis's article, "Logging with .NET Core," at msdn.com/magazine/mt694089.

Here, I'll only use the providers that output to the console and the debugger.

The logging extensions are dependent on. NET Standard. That means you can use the logging in .NET Framework 4.6.1 and higher apps or in .NET Core and ASP.NET Core. ASP.NET Core has the logging built in and it's a lot easier to tap into it there. But I'll use a .NET Framework app and configure the logging directly in a DbContext class in order to demonstrate surfacing EF Core logging, as there are already so many examples of using logging from ASP.NET Core.

Some of the technology in this article is in preview; all information is subject to change.

Code download available at msdn.com/magazine/1018magcode.

EF Core Logging in General

In order to surface EF Core logging information, you need to tie an ILoggerFactory to a DbContext. The DbContext will then share information with that logger factory. ASP.NET Core does this for you, but outside of ASP.NET Core this step must be done explicitly.

First you need to define a LoggerFactory object, and this is where you can specify which providers that LoggerFactory should use. Given that I already have a reference to Microsoft.Extensions.Logging.Console and a using statement to that namespace in my DbContext class, this is the simplest way to define a LoggerFactory that will output to a console window.

The different providers have their own constructor definitions. For example, the ConsoleProvider has four different constructors that take different objects to express settings for the provider. One of those constructors takes a lambda expression to surface a logging level and a logging category. Logging levels help organize different classes of logging details. One level consists of logs containing some basic information. Another holds more detailed logs to help in debugging a problem. The APIs that are sharing their information with the logger determine in which level their data belongs. For example, the database commands surfaced by EF Core are tagged as both Information and Debug log level. The change tracker details surfaced by EF Core are tagged as the Debug level. I'll show you some examples of these.

The categories are determined by the type of information being shared. EF Core 2.1 exposes 12 different categories of logging information. The EF Core LoggerCategory class at bit.ly/2KBLWs1 is a handy reference, where you'll find a list of categories that derive from it, as shown in **Figure 1**.

Figure 1 **EF Core LoggerCategory Classes**

ChangeTracking	Migrations
Database	Model
Database.Command	Model.Validation
Database.Connection	Scaffolding
Database.Transaction	Query
Infrastructure	Update

Notice there are even log events exposed for migrations commands.

The log levels and categories make it easy to filter what's output by the logger—I'll use that to show you just the SQL and ChangeTracker details.

Examining EF Core's Generated SQL

Logging the SQL commands generated by EF Core activity may be the most common logging task you'll want to perform with EF Core. Of course, you always have the option of using SQL profiling tools such as the SQL Profiler for Windows, EF Prof (efprof.com) and the (in preview) profiling feature of the cross-platform SQL Operations Studio (bit.ly/2MnfN94). But you may not always have access to these tools, especially if you want to trigger logs from a user's app, not on a developer machine.

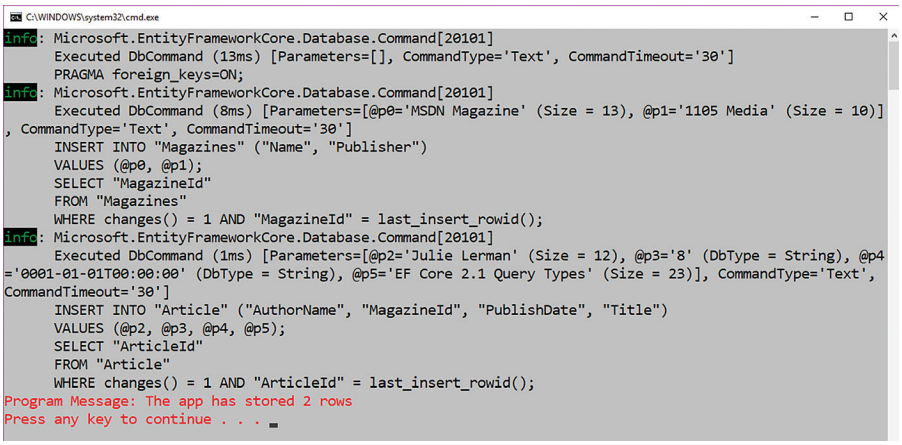


Figure 2 **Log Output When Filtering on Information Level Database.Commands**

A logger factory can be composed of one or more providers, which allows you to spit out logs to multiple destinations concurrently.

So, let's start by creating a logger factory that spits out the SQL sent to the database. The SQL commands are exposed through the Database.Command category.

I'll create a LoggerFactory named DbCommandConsoleLoggerFactory and then explain the code:

```
public static readonly LoggerFactory DbCommandConsoleLoggerFactory
  = new LoggerFactory (new [] {
      new ConsoleLoggerProvider ((category, level) =>
        category == DbLoggerCategory.Database.Command.Name &&
        level == LogLevel.Information, true)
    });
```

Notice that the LoggerFactory is static. The EF team (and the docs) advise that you use the same logger for every instance of a context during the application lifetime. Otherwise, you can encounter some strange side effects, as well as slow EF Core's processing dramatically. Making this logger factory static ensures that it will remain in scope and available for every new instance of the context throughout the app's lifetime.

A logger factory can be composed of one or more providers, allowing you to spit out logs to multiple destinations concurrently, but I'll only add one. This constructor takes an array of different logger providers. After defining the array via new [], I add only a single provider—a ConsoleLoggerProvider. Of the four available constructors for that provider, I'll use the one that lets me filter using a lambda composed from the category and level of the log. The first parameter is the predicate I'll use for filtering. I'm providing variable names—category and level—to the values of the expression and then building the filter from those to indicate that the category (a string) should be equal to the name of the Database.Command class and the level should be equal to the LogLevelInformation enum.

Database.Command outputs SQL plus a few other details through the Debug level, but only SQL commands through the Information level. So you're better off specifying just one of the LogLevels or you'll get redundant commands. Debug will output commands as Executing, whereas Information will output them as Executed.

The second parameter is a Boolean for the IncludeScope option. Log scopes allow you to group logs, but as far as I know, EF Core doesn't use them.

With a LoggerFactory defined, it must then be tied to a DbContext, which you can do within the OnConfiguring method. UseLoggerFactory is a method of DbContextOptionsBuilder. If you're defining other options, such as UseSqlite, you can append UseLoggerFactory. Otherwise, you can just call it directly from the optionsBuilder. Either way, you then pass in the factory that's defined in your class:

```
optionsBuilder.UseLoggerFactory(DbCommandConsoleLoggerFactory)
```

Logging the SQL commands generated by EF Core activity may be the most common logging task you'll want to perform with EF Core.

By default, EF Core will protect you from exposing sensitive data, such as filter parameter values. When I'm debugging my own code, I often add the EnableSensitiveDataLogging method to expose that:

```
optionsBuilder.UseLoggerFactory(
  DbCommandConsoleLoggerFactory).EnableSensitiveDataLogging();
```

Figure 3 **Database.Commands from LogLevel.Debug**

With the context now set up to expose this data, I have a console application that creates some data and saves it to the database to which my context is mapped. All I'm doing is creating a single magazine and adding a new article to its articles collection:

```
using (var context = new PublicationsContext())
{
  var mag = new Magazine("MSDN Magazine", "1105 Media");
  var article = new Article("EF Core 2.1 Query Types","Julie Lerman");
  mag.Articles.Add(article);
  context.Magazines.Add(mag);
  var results=context.SaveChanges();
  Console.ForegroundColor = ConsoleColor.Red;
  Console.WriteLine($"Program Message: The app has stored {results} rows");
}
```

The logger outputs three sets of Information followed by my Program Message output (see **Figure 2**). Note that I'm mixing logging and Console WriteLine messages here specifically for this simple demo. But you shouldn't do this in a production application because there is a chance of processing conflicts and logging messages not reaching the console.

> If I change the Visual Studio Output window to display Debug output, I can see the SQL commands output by the logger in between the Visual Studio debug messages.

Each message from the logger is prefaced by a line that says:

```
"info: Microsoft.EntityFramewrokCore.Database.Command[20101].
```

20101 is the unique EventId assigned to Executed DbCommand events.

The first line is a special command SQLite sends by default for each newly opened connection to ensure that foreign keys are honored. The next command inserts the new magazine and returns its newly generated primary key. The last command inserts the article along with the magazine's key value into the MagazineId foreign key column and returns *its* newly generated primary key. Finally, the console app outputs my own message, which I specified with Console.WriteLine.

Notice that the info items say "Executed DbCommand," even relaying the time the command took to execute and return the results to EF Core to continue processing those results. Because I enabled sensitive data logging, you can also see the parameter values that were passed in.

If I were to change the LogLevel from Information to Debug, the output would be slightly different, as shown in **Figure 3**. The flags are dbug rather than info. The SQL is still logged but as "Executing DbCommand" with an event id of 20100, in contrast to the Executed DbCommand events where the event ids were 20101. Moreover, Database.Command relayed two additional events when the data readers were disposed after receiving the new PrimaryKeys in the results from the SELECT statements.

Let's look at a DebugLoggerProvider outputting the same Database.Commands.

First, I'll create another LoggerFactory, this time one that just pushes logs through a DebugLoggerProvider. This provider has only two constructors, but one of them is similar to the ConsoleLoggerProvider's constructor—taking in a filter lambda expression but without the IncludeScope parameter. My new LoggerFactory looks pretty close to the other:

```
public static readonly LoggerFactory DbCommandDebugLoggerFactory
  = new LoggerFactory (new [] {
      new DebugLoggerProvider(
        (category, level) => category == DbLoggerCategory.Database.Command.Name &&
                             level == LogLevel.Information)
      );
```

After modifying the optionsBuilder.UseLoggerFactory to use this object, my console displays only the Program Message and Press any key output.

But if I change the Visual Studio Output window to display Debug output, I can see the SQL commands output by the logger in between the Visual Studio debug messages. **Figure 4** displays some of this output. Notice that the EF Core logs don't have the "info" flags or the Event Ids, but do list the category followed by the name of the log level.

Now let's take a look at the ChangeTracking category. How the EF Core change tracker works is very interesting to me—dare I say fascinating? I hope you agree. All of the events in this category fall into the debug log level.

I'll create another LoggerFactory using the ConsoleLoggerProvider again, but this time my filter combines Database.Commands and ChangeTracking events. By filtering on Debug level, I can get not only all of the ChangeTracking events, but also the

DevExpress®

From Desktops to Web and Mobile

Your Next Great App Starts Here

Experience the DevExpress difference and see why your peers consistently vote our products #1. With our Universal Subscription, you will build your best, see complex software with greater clarity, increase your productivity and create stunning applications for Windows, Web and your Mobile world.

DevExpress Universal ships with 500+ UI controls.
It also includes our royalty-free reporting and dashboard platform.

WIN ASP MVC WPF UWP JS

Download your free 30-day trial today.

devexpress.com/try

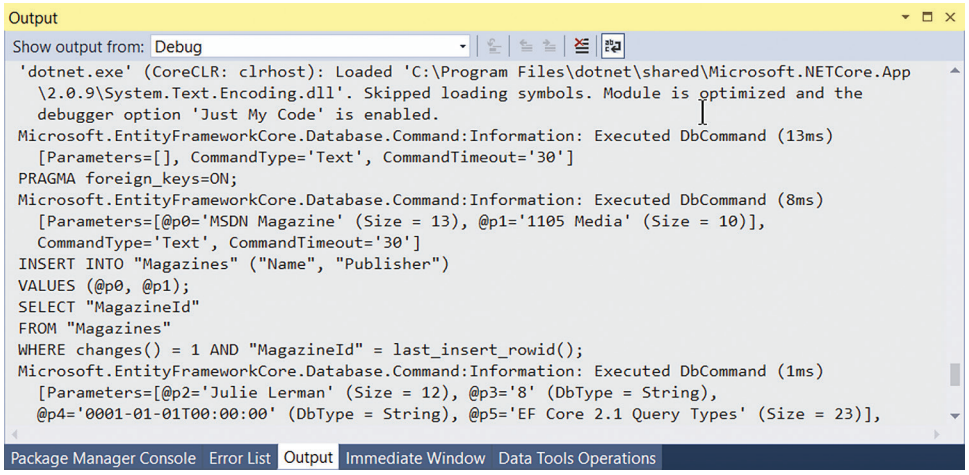


Figure 4 **Database.Commands Written out to the Debug Window**

SQL captured by the commands. Remember that debug raises Executing DbCommand events and a few others:

```
public static readonly LoggerFactory
ChangeTrackingAndSqlConsoleLoggerFactory
  = new LoggerFactory(new[] {
      new ConsoleLoggerProvider (
        (category, level) =>
        (category == DbLoggerCategory.ChangeTracking.Name |
        category==DbLoggerCategory.Database.Command.Name)
        && level==LogLevel.Debug ,true)
  });
```

After switching the optionsBuilder to use this logger factory, the output is quite different! The first part of that output is shown in **Figure 5**. If you're not familiar with how the change tracker goes about its business, this may be especially interesting to you. Note that the second debug log in **Figure 5** switches the context name with the entity name—a known bug who's fix will appear in EF Core 2.2.

> The ChangeTracking details for interacting with data coming from the database, especially if you modify and save it, are really impressive.

As before, each of the entries is prefaced by the category with the relevant Event Id. The events begin when my method calls context.Magazines.Add.

The first event is that the context generated a temporary key value (EventId 10808) for the new magazine and then it began tracking the key (EventId 10806). Next, the context repeated the same two steps (EventIds 10808 and 10806 again) for the new article. The temporary key values are the same because they're different entity types and -2147482647 is the starting point for temporary key values.

After adding the magazine and its article to the context, I then call SaveChanges. This triggers DetectChanges and you can see an event for when it started and when it ended. Because there are no changes to the entities from the time they were tracked, there's nothing to discover, so this is followed immediately by "DetectChanges completed." As a note, because these are new entities, even if something had changed, their state would still be Added.

After the context finishes detecting changes, it starts sending the SQL to the database. So next in the log are the special SQLite commands for the foreign keys setting, followed by the command to insert the magazine and return its new MagazineId.

Then the change tracker kicks in again, replacing the temporary value generated by the database for the MagazineId with the new database-generated value—fixing up first Magazine.MagazineId and then Article.MagazineId:

```
dbug: Microsoft.EntityFrameworkCore.ChangeTracking[10803]
      Foreign key property 'Magazine.MagazineId' detected as changed from
      '-2147482647' to '17' for entity with key '{MagazineId: 17}'.
dbug: Microsoft.EntityFrameworkCore.ChangeTracking[10803]
      Foreign key property 'Article.MagazineId' detected as changed from
      '-2147482647' to '17' for entity with key '{ArticleId:
      -2147482647}'.
```

Now article has the correct foreign key value for MagazineId and the next log event is EF Core sending the SQL to insert the article into the database.

Finally, two more change tracking events are logged as EF Core revises the known state of both the magazine and the article it's tracking to Unchanged:

```
dbug: Microsoft.EntityFrameworkCore.ChangeTracking[10807]
      The 'Magazine' entity with key '{MagazineId: 17}' tracked by
      'PublicationsContext' changed from 'Added' to 'Unchanged'.
dbug: Microsoft.EntityFrameworkCore.ChangeTracking[10807]
      The 'Article' entity with key '{ArticleId: 17}' tracked by
      'PublicationsContext' changed from 'Added' to 'Unchanged'.
```

I love seeing this laid out so clearly. Over the decade I've been working with Entity Framework, I've spent a lot of time debugging through code and digging very deeply through runtime debuggers and watch windows to understand how the change tracker works.

The ChangeTracking details for interacting with data coming from the database, especially if you modify and save it, are really

Figure 5 **First Part of ChangeTracking and Database.Command Category Events**

```
dbug: Microsoft.EntityFrameworkCore.ChangeTracking[10808]
      'PublicationsContext' generated temporary value '-2147482647' for
      the 'MagazineId' property of new 'Magazine' entity.
dbug: Microsoft.EntityFrameworkCore.ChangeTracking[10806]
      Context 'Magazine' started tracking '{MagazineId: -2147482647}'
      entity with key 'PublicationsContext'.
dbug: Microsoft.EntityFrameworkCore.ChangeTracking[10808]
      'PublicationsContext' generated temporary value '-2147482647' for
      the 'ArticleId' property of new 'Article' entity.
dbug: Microsoft.EntityFrameworkCore.ChangeTracking[10806]
      Context 'Article' started tracking '{ArticleId: -2147482647}'
      entity with key PublicationsContext'.
dbug: Microsoft.EntityFrameworkCore.ChangeTracking[10800]
      DetectChanges starting for 'PublicationsContext'.
dbug: Microsoft.EntityFrameworkCore.ChangeTracking[10801]
      DetectChanges completed for 'PublicationsContext'.
```

File Format APIs

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

Try risk free - 30 day trial

ASPOSE

Download a Free Trial at

https://downloads.aspose.com

Aspose.Words

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

Aspose.Cells

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

Aspose.Pdf

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

Aspose.Slides

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

Aspose.Email

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

Aspose.BarCode

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

Aspose.Imaging

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

Aspose.Tasks

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

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

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

impressive. Let's look at that with a new method, GetAndUpdateSomeData in **Figure 6**, which retrieves a magazine with an article, makes an edit and then saves the changes. The method also writes out notices ("Query," "Edit" and "Save") so in the log I can see when I ran the query, when I made an edit and when I called SaveChanges.

I won't list the entire log, but it first shows the SQL to query for the magazine with MagazineId 1. The log serves as a reminder that in EF Core, some Includes (notably for collections) are now broken down into separate queries on the database because it's generally more efficient with respect to the size of the results and the effort of materializing the objects. After the magazine is returned, another event—that the context started tracking that Magazine—is displayed. Next, EF Core sends SQL to retrieve the articles for all of the magazines I filtered on, which will return only one article because that's all the data I've got (just the one with MagazineId=1). The next event is that the context has started tracking that article entity. After this, there are two events indicating that the data reader was disposed—one for each of the two queries.

Now things get more interesting, as you can see in **Figure 7**, which shows most of the rest of the log.

First notice my "Edit" message is followed directly by a "Save" message. Even though the context is currently tracking the article I'm editing, it has no way of updating state information in real time. It waits until the change tracker's DetectChanges method is called, either explicitly in code or by another EF Core method such as SaveChanges. So it's not until after I call SaveChanges that DetectChanges starts doing its job. The first thing it sees is that AuthorName has changed. The 18082 event relays that information and indicates that because of that change, it's planning to mark its entity (the Article whose ArticleId is 1) as modified. The next event, 18087, states that it just changed that entity (ArticleId: 1) from Unchanged to Modified. This is amazing detail and when you're debugging a problem or some unexpected behavior, it will be extremely helpful. I'm really grateful for this! (Hat tip to Arthur Vickers on the EF Team, as logging was his baby!)

Figure 6 **A New Method for Exploring the Logging**

```
private static void GetAndUpdateSomeData()
{
  using (var context = new PublicationsContext())
  {
    Console.WriteLine("Query");
    var mag = context.Magazines.Include(m=>m.Articles)
      .FirstOrDefault(m => m.MagazineId == 1);
    Console.WriteLine("Edit");
    mag.Articles.FirstOrDefault().AuthorName += " and Friends";
    Console.WriteLine("Save");
    var results = context.SaveChanges();
    Console.ForegroundColor = ConsoleColor.Red;
    Console.WriteLine($"Program Message: The app has updated {results} rows");
  }
```

Figure 7 **ChangeTracking Logs During an Update**

```
Edit
Save
dbug: Microsoft.EntityFrameworkCore.ChangeTracking[10800]
      DetectChanges starting for 'PublicationsContext'.
dbug: Microsoft.EntityFrameworkCore.ChangeTracking[10802]
      Unchanged 'Article.AuthorName' detected as changed from 'Julie Lerman' to
     'Julie Lerman and Friends' and will be marked as modified
      for entity with key '{ArticleId: 1}'.
dbug: Microsoft.EntityFrameworkCore.ChangeTracking[10807]
      The 'Article' entity with key '{ArticleId: 1}' tracked by
      'PublicationsContext' changed from 'Unchanged' to 'Modified'.
dbug: Microsoft.EntityFrameworkCore.ChangeTracking[10801]
      DetectChanges completed for 'PublicationsContext'.
dbug: Microsoft.EntityFrameworkCore.Database.Command[20100]
      Executing DbCommand [Parameters=[], CommandType='Text',
      CommandTimeout='30'] PRAGMA foreign_keys=ON;
dbug: Microsoft.EntityFrameworkCore.Database.Command[20100]
      Executing DbCommand [Parameters=[@p1='1' (DbType = String),
      @p0='Julie Lerman and Friends' (Size = 24)], CommandType='Text',
      CommandTimeout='30']
      UPDATE "Article" SET "AuthorName" = @p0
      WHERE "ArticleId" = @p1;
      SELECT changes();
dbug: Microsoft.EntityFrameworkCore.Database.Command[20300]
      A data reader was disposed.
dbug: Microsoft.EntityFrameworkCore.ChangeTracking[10807]
      The 'Article' entity with key '{ArticleId: 1}' tracked by
      'PublicationsContext' changed from 'Modified' to 'Unchanged'.
```

> After the context finishes detecting changes, it starts sending the SQL to the database.

After the change tracker finishes this work, you then see the update method. Again, the fact that I've enabled sensitive data is why you can see all of the parameters of the SQL command.

The "SELECT changes" at the end of the SQL is the SQLite way of returning the count of rows that were affected.

Finally, the change tracker cleans up the state of its tracked objects, marking the article as Unchanged and ready for its next adventure.

Don't Forget About Other Logger Categories

It looks like I'd have to fill this magazine to show you the ins and outs of the other 10 DbLoggerCategory types that EF Core exposes. Now that you have a handle on how to expose that data, you can experiment with the different categories and see what kinds of information they share. Alternatively, you can just have EF Core spit out all of its logging data by defining a logging provider that doesn't filter on anything. Rather than a predicate expression for the filter, just return true as I've done in this Too Much Information (TMI) factory:

```
public static readonly LoggerFactory TMIConsoleLoggerFactory
  = new LoggerFactory(new[] {
      new ConsoleLoggerProvider ((category, level) => true, false)
  });
```

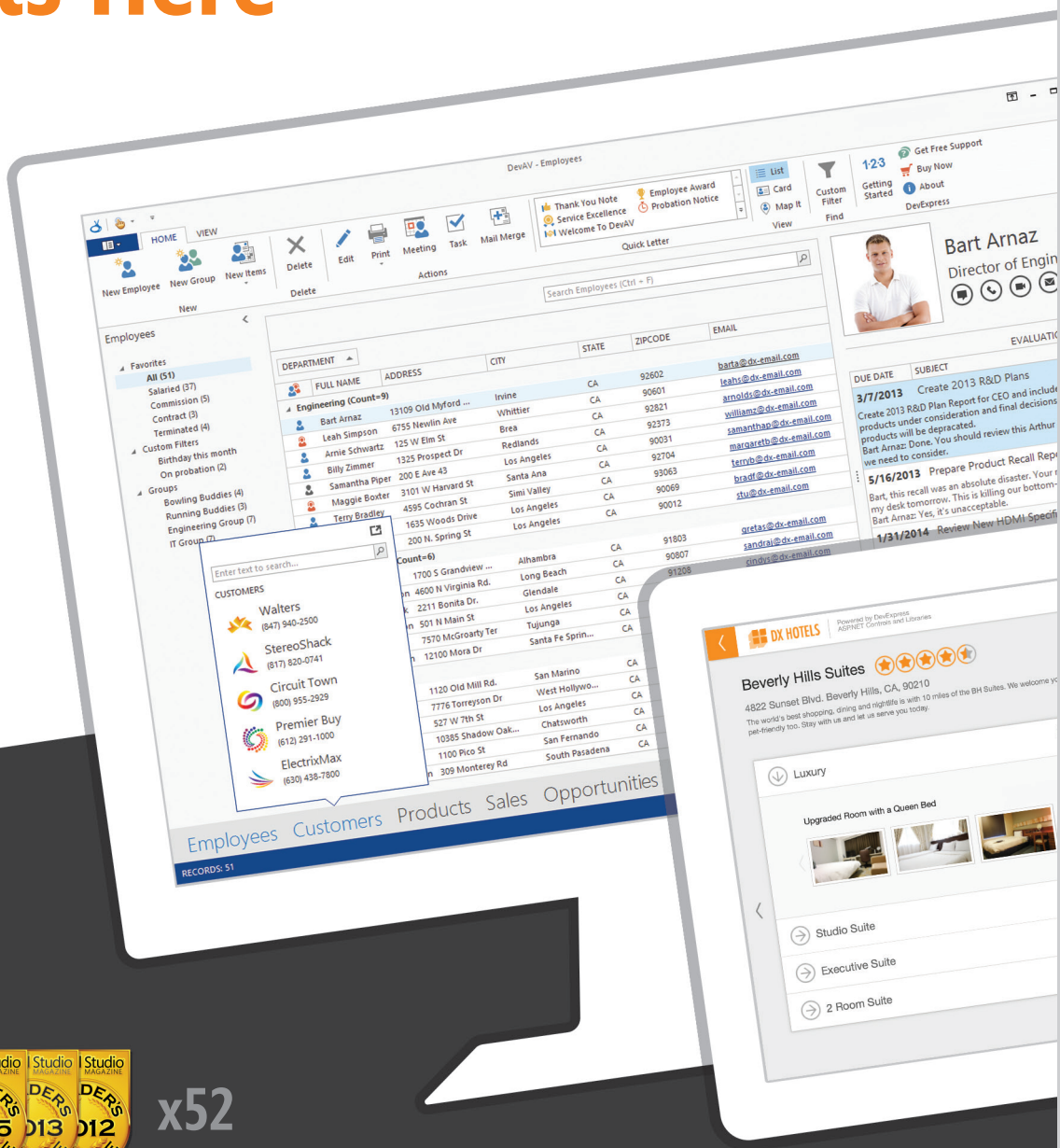
While logging is incredibly helpful for debugging problems, I think it's equally important for gaining insight into how an API works. And the better you understand how it works, the less likely it is you'll create problems that will need to be debugged. ■

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

Thanks to the following Microsoft technical expert for reviewing this article: *Arthur Vickers*

DevExpress®

Your Next Great App
Starts Here



Visual Studio Magazine Readers Choice 2017 · Visual Studio Magazine Readers Choice 2016 · Visual Studio Magazine Readers Choice 2015 · Visual Studio Magazine Readers Choice 2013 · Visual Studio Magazine Readers Choice 2012

x52

When Only the Best Will Do

With the DevExpress Universal Subscription, you'll deliver amazing, high-impact user experiences for Windows®, the web and your mobile world. Get started today and see why your peers consistently vote DevExpress products #1.

Download your free 30-day trial at **devexpress.com** and let's build great apps together.

UI Controls for

DESKTOP / WEB / MOBILE

Unleash the UI Superhero in you and create solutions that fully address customer requirements today and leverage your code investments to build next generation touch-enabled solutions for tomorrow. Whether building an Office® inspired app or a data centric analytics dashboard, DevExpress Universal ships with everything you'll need to build your best, without limits or compromise.

And with industry recognized support services, we'll be with you every step of the way — making certain that our products fulfill their promise. Visit **devexpress.com/support** or email **support@devexpress.com** anytime, with any DevExpress product question.

Geoffrey Jones
Code21 Solutions Pty Ltd

We have been using DevExpress products for over 5 years and been delighted with the results. DevExpress have helped us to develop an enterprise-ready labour management solution with WinForms and ASP.NET modules. As a small team, it would not have been possible for us to develop an application with a polished UI, highly scalable architecture and flexible customization options without the assistance of DevExpress.

Office-Inspired Apps

Touch-Enabled Windows & Web Apps

HTML5 Mobile Apps

Reporting-Dashboard & Analytics Apps

Experience the DevExpress Difference Risk Free

We are so confident in our tools that we back them with a 60-day unconditional money-back guarantee. If in your first 60 days of ownership you find that our products do not meet your business needs, you can return them to us for a full, no questions asked refund.

To get started and to choose a subscription that's right for you, email **info@devexpress.com** or call **+1 (818) 844-3383**.

60 DAY MONEY-BACK GUARANTEE

Grid Controls

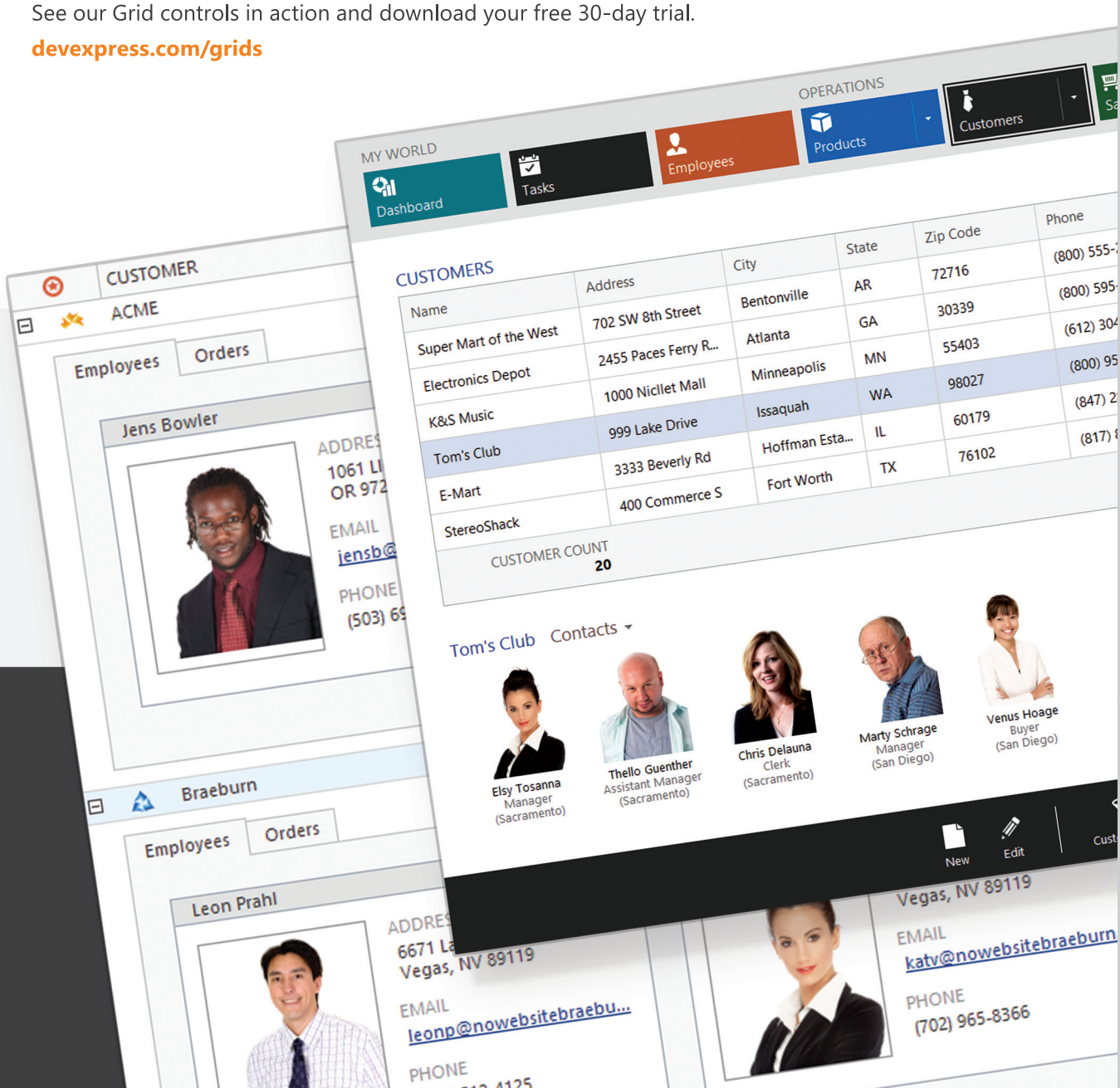
Blazing fast, feature-complete and just plain beautiful.

DevExpress Grid controls are Outlook® inspired record editing and data shaping components, allowing your end-users to manage and arrange information on-screen as business requirements dictate. Our data grids ship with dozens of industry leading features including integrated master-detail support and multiple data layout options such as Card and Windows Explorer views. And whether you target desktops or touch devices, all DevExpress Grid Controls support a broad range of user interaction options.

WIN WPF ASP MVC JS UWP

Your Next Great App Starts @DevExpress

See our Grid controls in action and download your free 30-day trial.
devexpress.com/grids



Reporting

Inform and analyze with absolute ease.

The ever changing needs of the enterprise require that reporting platforms offer simple and straightforward end-user customization options so that consumers can freely manipulate report output for maximum clarity. Flexibility is key and ease-of-use paramount.

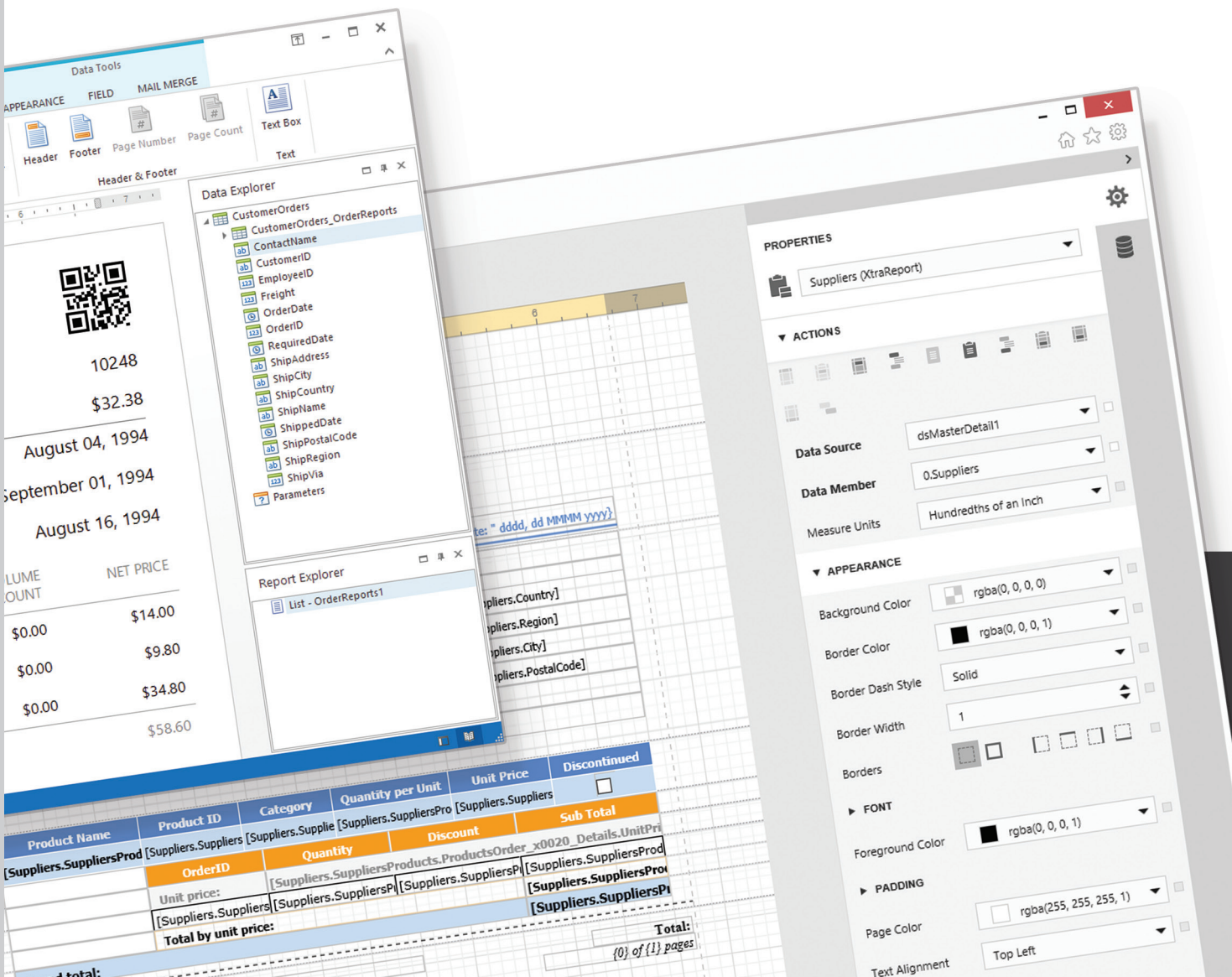
The DevExpress Reporting Suite for .NET was built with your end-users in mind and engineered so you can get the most out of your reporting and data analytics investments. Our Reporting Suite helps you overcome the limitations associated with traditional report tools by providing a fully integrated set of productivity tools, report wizards, pre-built report templates and end-user report designers for both Windows and the web.

WIN WPF ASP MVC

Your Next Great Report Starts @DevExpress

See how to solve your Windows® and web reporting requirements forever.

devexpress.com/reports



Spreadsheets (XLS, XLSx, CSV)

The power of Excel® at your fingertips.

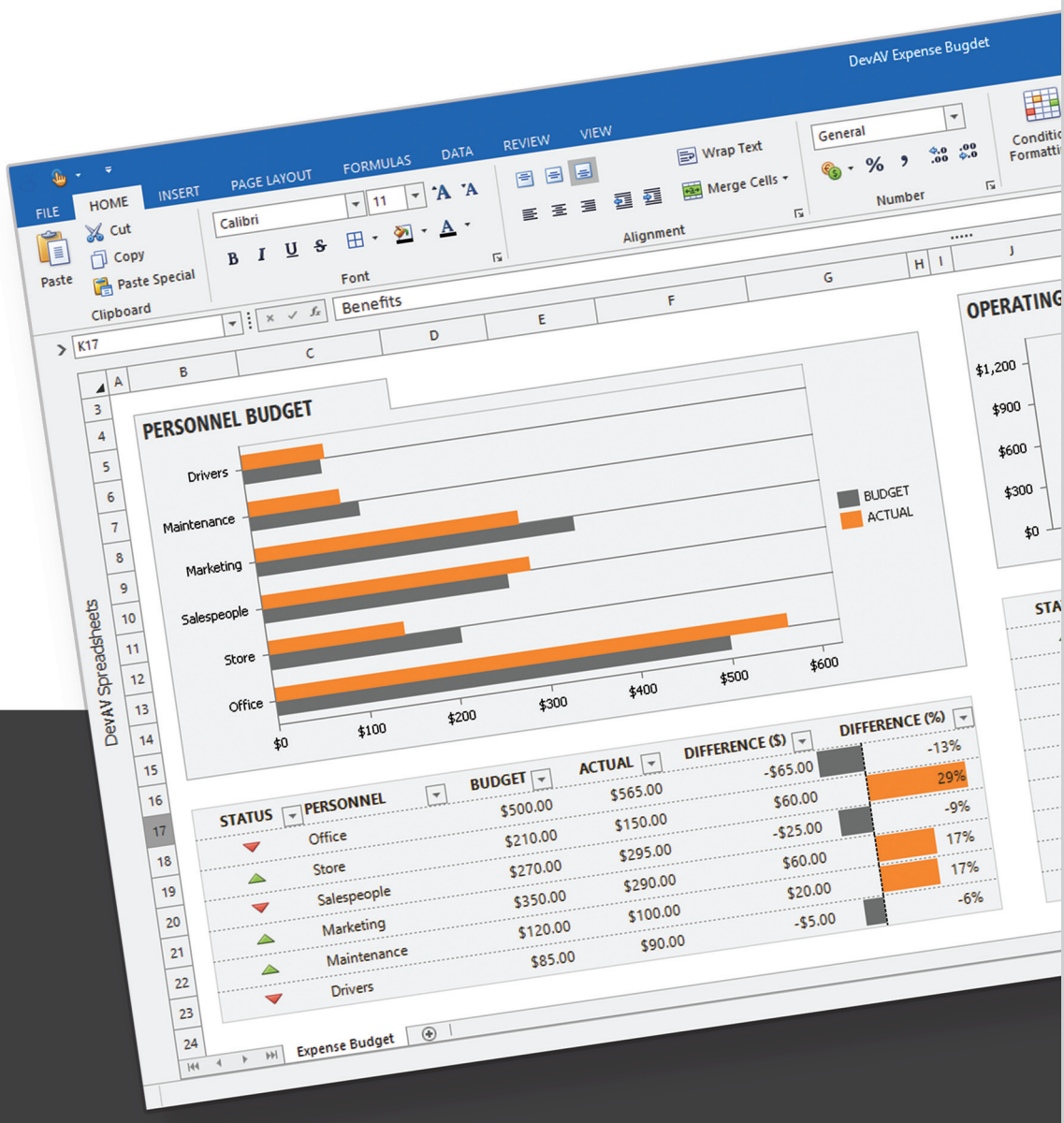
It's no secret...spreadsheets are an important part of business and if you need to incorporate the UX and functionality end-users have come to expect from Microsoft Excel®, DevExpress Spreadsheet is the tool for you. The component library ships with dozens of easy-to-implement features including chart support and fully integrates with other DevExpress components like our Office® inspired Ribbon.

WIN WPF ASP MVC

Your Next Great Spreadsheet Starts @DevExpress

See how you can harness the power of spreadsheets in your next .NET app.

devexpress.com/spreadsheet



Word Processing (DOC, DOCx, RTF)

Create the perfect WYSIWYG document or report.

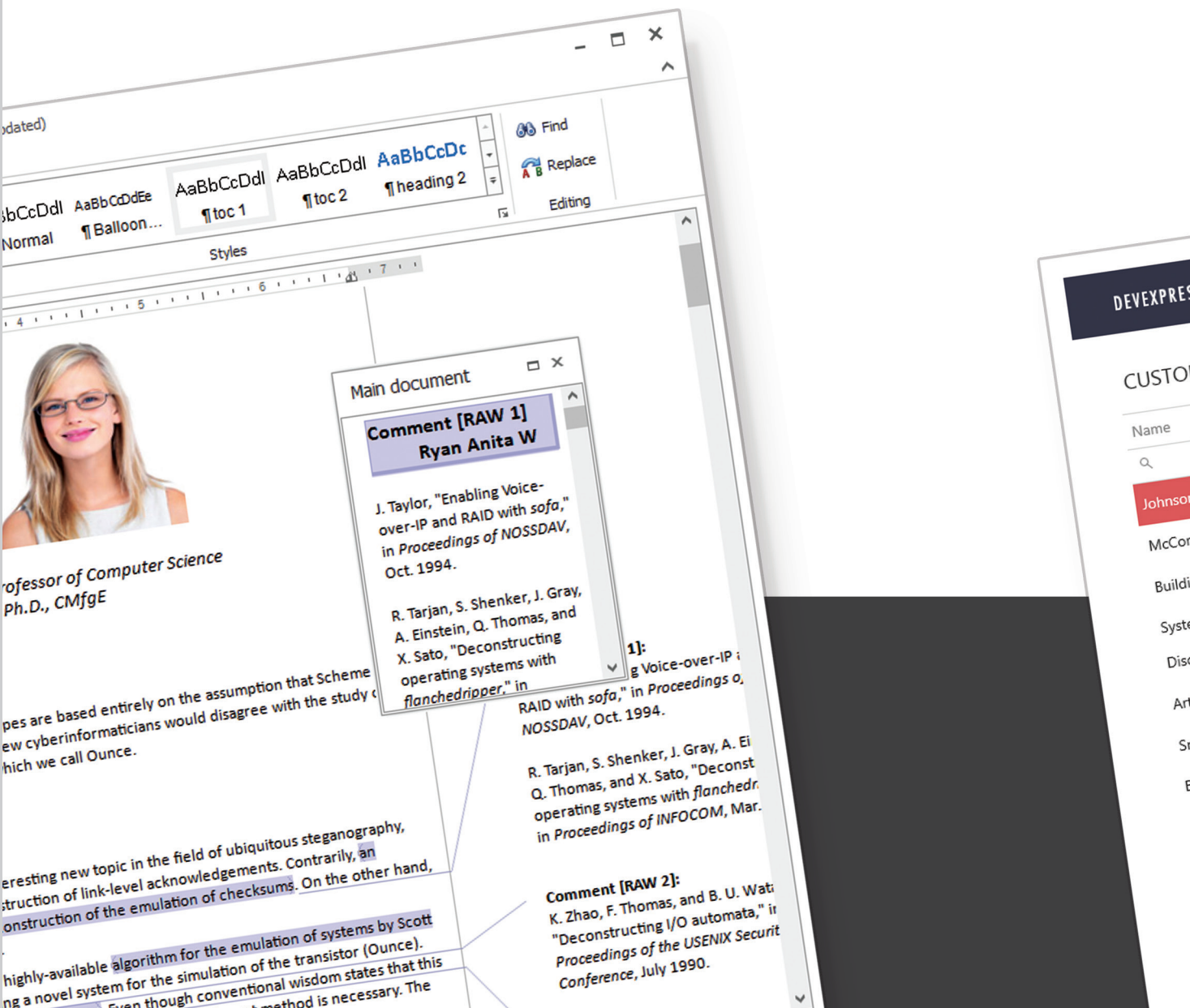
When rich text editing is a must and you need to replicate key features found inside Microsoft Word® (from mail merge and tables, to floating objects and document comments), look no further than the DevExpress Rich Text Editor Control. The library's comprehensive feature set includes printing and PDF export support and the availability of DevExpress Visual Studio project templates means you can incorporate our rich text editors in your next .NET app within minutes of install.

WIN WPF ASP MVC

Your Next Great Text Editor Starts @DevExpress

See how you can introduce Word® compatible rich text editing in your app.

devexpress.com/word



Charting & Analytics

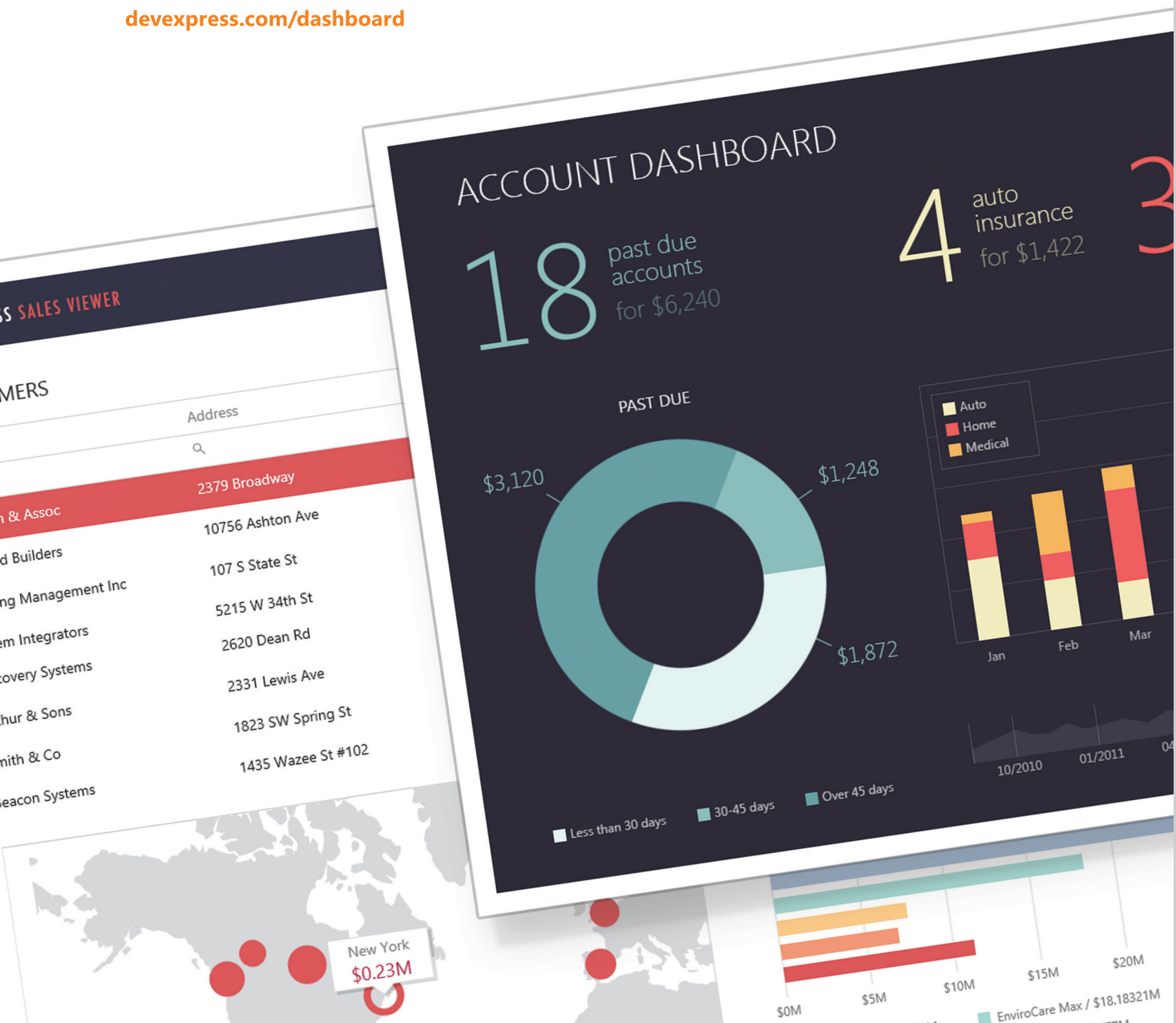
From zero to dashboard in record time.

With our integrated suite of charts, pivot grids, and multi-purpose dashboard widgets, you'll create information-rich decision support systems that are optimized for Windows, the web and your favorite mobile device. The UI components inside our subscriptions help you deliver adaptable, interactive and touch-enabled user experiences that can address a broad range of use-case scenarios. And perhaps best of all, the dashboards you build with DevExpress Universal can be distributed royalty-free.

WIN WPF ASP MVC JS UWP

Your Next Great Dashboard Starts @DevExpress

See how you can build future proof, enterprise-grade decision support systems.
devexpress.com/dashboard



Introduction to Reinforcement Learning

In previous articles, I've mentioned both supervised learning and unsupervised learning algorithms. Beyond these two methods of machine learning lays another type: Reinforcement Learning (RL). Formally defined, RL is a computational approach to goal-oriented learning through interaction with the environment under ideal learning conditions.

Like other aspects of AI, many of the algorithms and approaches actively used today trace their origins back to the 1980s (bit.ly/2NZP177). With the advent of inexpensive storage and on-demand compute power, reinforcement learning techniques have re-emerged.

Arguably the most famous use of RL to date is the AlphaGo Zero program from DeepMind Technologies Ltd. Unlike AlphaGo, which learned from records of both amateur and professional human players, AlphaGo Zero had no access to human-based training datasets. Instead, the system learned entirely from playing games against itself. After several weeks, AlphaGo zero achieved master-level status and, after 40 days, became the best Go player in the world. Keep in mind that it did all this with no human intervention. More information about the project is at bit.ly/2K9NpW8. RL has also been used to train algorithms to play classic arcade games and achieve the highest possible scores with little or no human participation.

In this article, I'll discuss the foundations of RL and build upon this in future articles.

Core Components of an RL Model

RL focuses on training a model using a reward system in a manner very similar to how animals and humans learn based on experience—by trial and error. Because RL mimics human cognitive behavioral systems, there's quite a lot of overlap with cognitive psychology. RL algorithms also deploy a fair bit of game-theory mathematics and strategies, as well.

The first two elements in any RL model are the environment and the agent. Using the example of a game of chess, the environment is the game board and the agent is the player. The state of the environment is the placement of the pieces on the chessboard. The agent observes the state of the environment and applies an action to the environment to receive a reward (see **Figure 1**). In the game of chess, this action could be moving a pawn forward or capturing the opponent's piece. Any move in a game of chess automatically changes the state of the environment. The agent may take any of the valid moves available to the player in a game of chess.

The agent receives a reward from the environment based on the action it took. The reward is a numerical value analogous to the score in a video game. Agents choose among the available options based on the anticipated reward. Agents are programmed

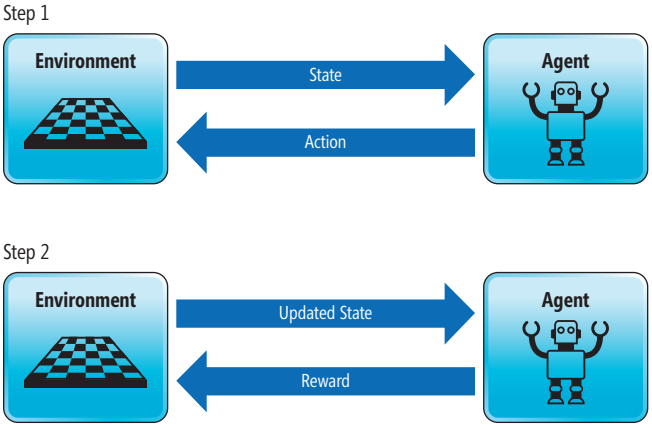


Figure 1 **Essential Components to a Reinforcement Learning System**

to maximize their reward. For example, moving a piece forward may yield a reward of 1, while capturing an enemy piece may yield a reward of 5. Given these choices, the algorithm will choose capturing an enemy piece over simply moving forward every time. Such a reward structure will yield an aggressive agent, as it will always choose capturing an opponent's piece, even at the expense of losing the game. Clearly, this is not the optimal outcome. Therefore, the accuracy of the environment and reward structure are critical to the success of a model trained via RL.

A Smaller Problem Space

Building a model that captures the complexity of chess to train an agent falls far outside the scope of this column. For simplicity, I will reduce the environment down to a 4x4 grid. The agent will control the movement of a character in this grid. Some cells of the grid are walkable, while others lead to the agent falling into the water and, thus, killing the character and resetting the simulation. The agent has a goal cell to reach and will be rewarded with a score of 1 for finding a safe path to it. Moving around the board returns a reward value of 0. For this example, I'll use the Frozen Lake environment from OpenAI Gym (bit.ly/2v1csWA).

To get started, open a Python 3 Jupyter notebook on your preferred platform. I covered how to get started with Jupyter notebooks in an earlier column (msdn.com/magazine/mt829269). Create an empty cell and enter and execute the following code to import the appropriate libraries and set up the environment for the agent in which to train:

```
import gym
import numpy as np

env = gym.make('FrozenLake-v0')
```

Now, create a new cell, enter the following code, and then execute:

```
print("observation_space")
print(env.observation_space)
print("action_space")
print(env.action_space)
```

The result should indicate that there are 16 discrete values in the observation space, just what you'd expect in a 4x4 grid. It also shows that there are four discrete values in the action space, meaning that the agent can move up, down, left or right.

A Random Approach

Now that the environment is set up, it's time to start interacting with it. Add the following code to a new cell below the current one:

```
env.reset()
done = False

while done == False:

  action = env.action_space.sample()
  observation, reward, done, info = env.step(action)
  print(action, observation, reward, done, info)
```

The output should look like the following:

```
2 4 0.0 False {'prob': 0.3333333333333333}
3 4 0.0 False {'prob': 0.3333333333333333}
2 8 0.0 False {'prob': 0.3333333333333333}
0 4 0.0 False {'prob': 0.3333333333333333}
1 5 0.0 True {'prob': 0.3333333333333333}
```

This algorithm randomly explores the space by picking a direction from the action space at random. The step function accepts the action as a parameter and returns the observation, reward, done and info. Observation, in this case, represents the location on the grid. Reward is the amount of reward the step generated. Done is a Boolean value indicating whether the simulation ended. In this example, it means that our player has fallen through the ice and we need to restart the simulation. The final outbound parameter is info and is primarily used for diagnostic purposes. Given that zero reward points were earned means that the agent didn't reach the goal.

Of course, it's highly unlikely that the agent will just randomly come across the solution. Perhaps trying multiple times will yield a successful outcome. The following code, which should be entered and executed, will run the simulation 100 times:

```
print("starting")
for i in range(100):
    env.reset()
    done = False
    while done == False:
        action = env.action_space.sample()
      observation, reward, done, info = env.step(action)
      if(reward > 0):
        print("success")
        print(reward)
print ("done")
```

This, too, will not likely yield a successful outcome. Alter the code so that the simulation runs 1,000 times and run again. With 1,000 runs, there should be around 10 successful runs, give or take. Statistically speaking, this means that a random approach to the problem works about 1 percent of the time. Clearly, the random approach alone will not suffice.

Introducing Q-Learning

As you've seen, the random "try and fail" approach yields terrible results. Or does it? In higher-order animals, memory plays a crucial role in learning. A try-and-fail approach will be far more successful if the results from previous attempts are captured. The previous code starts anew every iteration, effectively relying on chance for success.

One particularly useful approach is Q-Learning. The Q-Learning method learns a strategy to guide an agent on which action to take under what circumstances. In its simplest configuration, the Q-Learning algorithm may be implemented as a table, called a Q-Table. In this Frozen Lake environment, there are 16 possible states, representing a location on the 4x4 grid and four possible actions for each direction. This results in a 16x4 table of Q-values. Each row of the table has four columns, the highest value on the row represents the best action to take. Therefore, the table contains rows that represent every possible state with columns indicating which direction has the highest likelihood of success.

Enter the code in **Figure 2** into a blank cell and execute (it should take a moment to run).

To create a Q-Table, first create a table with 16 rows and four columns. Recall that there will be one row for every possible state and one column for every possible action. The Q-table is initialized with values of zero in every cell, a task made easy using the zeros method. As the agent explores the environment and observes the results, the Q-Table gets updated. At each step of the simulation, the code selects the action (column) with the highest value for the given state (row), after which the code observes the rewards earned for various actions and then updates the table accordingly. The more the code runs the simulation, the more experience is stored in the Q-Table. Eventually, the values converge on an ideal and this results in a more accurate model. The code in **Figure 2** will run the simulation 2,000 times.

While much of the code is fairly straightforward and matches the description just provided, you may find the following line from the code snippet somewhat puzzling:

```
Qtable[s,a] = Qtable[s,a] + lr*(r + gamma*np.max(Qtable[s1,:]) - Qtable[s,a])
```

This code implements the Bellman Equation and stores the results in the Q-Table. A full discussion of the Bellman Equation falls outside of the scope of this column. For now, understand that the Bellman Equation represents the rewards associated with a sequence of actions. A complete walk-through of this equation may be found on YouTube at bit.ly/2n1eEc5.

Figure 2 **Implementing the Q-Table**

```
Qtable = np.zeros([env.observation_space.n,env.action_space.n])
lr = .8
gamma = .95
num_episodes = 2000
rList = []
for i in range(num_episodes):
  s = env.reset()
  rAll = 0
  d = False
  j = 0
  while j < 99:
    j+=1
    a = np.argmax(Qtable[s,:] + np.random.randn(1,env.action_space.n)*(1./(i+1)))
    s1,r,d,_ = env.step(a)
    Qtable[s,a] = Qtable[s,a] + lr*(r + gamma*np.max(Qtable[s1,:]) - Qtable[s,a])
    rAll += r
    s = s1
    if d == True:
      break
  rList.append(rAll)
```

Enter the following code in a new cell and execute it:

```
print( "score: " +  str(sum(rList)/num_episodes) )
```

The resulting score is an average of the rewards earned over 2,000 iterations and roughly equates to accuracy. This score should fall within a range of 0.42 and 0.54, meaning that the algorithm will successfully navigate its way through the environment somewhere between 42 percent and 54 percent of the time. Not perfect, but certainly better than the 1 percent success rate produced by random guessing with no memory of past actions.

Examining the Q-Table More Closely

Finally, it may help to take a closer look at the Q-Table. Enter "Qtable" into an empty cell and execute it. The output will be a 16x4 table of numbers between zero and one. Notice that some rows are entirely filled with zeros. Enter "Qtable[1,:]" into an empty cell and execute the code. The output will look like this:

```
array([0.00081276, 0.00239483, 0.00018525, 0.15080315])
```

This result means that the agent learned that the fourth action (going up) is the most likely path to reward when the environment is in state one. Rows filled with zeros indicate that the algorithm never filled in the value and it stayed at the default. Such a case would happen when the iteration ended, as the agent either fell into a hole or reached the goal.

Wrapping Up

In this article, I explored the key components of RL and two methods to explore an environment. The agent learned how to navigate an ice field with no human guidance. Everything the agent did, it learned on its own. In both scenarios, the agent explored its environment randomly. In the second example, the agent used a Q-Table, or a look-up table, that provided guidance on what move to make based on the current state of the environment. This gave the agent a sort of memory where it could learn from past decisions. As you saw, the result was a significant boost in success rate.

RL is among the most intriguing spaces in artificial intelligence right now. As AlphaGo Zero has demonstrated, human input is no longer an essential component for machine learning. While the potential applications for this technology are many, there are significant challenges to overcome before using it in any real-world project.

Special thanks to Arthur Juliani and his blog post, "Simple Reinforcement Learning with Tensorflow Part 0: Q-Learning with Tables and Neural Networks" (bit.ly/2OxySXQ), which provided an example of the Bellman equation implemented in Python. ■

Frank La Vigne *works at Microsoft as an AI Technology Solutions Professional, where he helps companies achieve more by getting the most out of their data with analytics and AI. He also co-hosts 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 expert for reviewing this article: *Andy Leonard*

LIVE! 360
TECH EVENTS WITH PERSPECTIVE

NEW! IN 2018

Artificial Intelligence LIVE!

AI FOR DEVELOPERS AND DATA SCIENTISTS

ROYAL PACIFIC RESORT AT UNIVERSAL ORLANDO
December 2-7, 2018

Full Agenda Available Now!
ATTENDAILIVE.COM

A New AI Event for Developers and Data Scientists

Artificial Intelligence Live! is an innovative, new conference for current and aspiring developers, data scientists, and data engineers covering artificial intelligence (AI), machine learning, data science, Big Data analytics, IoT & streaming analytics, bots, and more. You can expect real-world training on the languages, libraries, APIs, tools and cloud services you need to implement real AI and machine learning solutions, today and into the future.

TRACK TOPICS INCLUDE:

- Artificial Intelligence (AI)
- Machine Learning
- Data Science
- Big Data analytics
- IoT & Streaming Analytics
- Bots
- & More....

REGISTER NOW

SAVE $400 WITH SUPER EARLY BIRD PRICING!
REGISTER BY OCTOBER 5
Use Promo Code MSDN

A Part of Live! 360: The Ultimate Education Destination

6 CONFERENCES, 1 GREAT PRICE

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

ATTENDAILIVE.COM

EVENT PARTNERS
Microsoft
Magenic

SILVER SPONSOR
Progress
GOVERLAN REACH

PREMIER & ALLIANCE MEDIA PARTNERS
netda
UserGroup.TV

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

PRODUCED BY
CONVERGE 360
An 1105 MEDIA company

AGENDA-AT-A-GLANCE

AI Application Development	Analytics	Bots	Data Science and Machine Learning	Internet of Things	Deep Learning

START TIME	END TIME		
Artificial Intelligence Live! Full Day Hands-On Lab: Sunday, December 2, 2018			
7:30 AM	9:00 AM	Registration • Coffee and Morning Pastries	
9:00 AM	6:00 PM	**AIS01** Hands-On Lab: Build a Bot in a Day Using the Microsoft Bot Framework, Cognitive Services and Azure *- Brian Randell*	
2:00 PM	7:00 PM	Pre-Conference Registration - Royal Pacific Resort Conference Center	
Artificial Intelligence Live! Pre-Conference Workshops: Monday, December 3, 2018			
7:00 AM	8:30 AM	Registration • Coffee and Morning Pastries	
8:30 AM	5:30 PM	**AIM01** Workshop: So You Want to do Data Science, What Now? *- Matt Winkler & Euan Garden*	**AIM02** Workshop: Moving from BI to AI: Artificial Intelligence Skills for Business Intelligence Professionals - *Jen Stirrup*
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	
Artificial Intelligence Live! Day 1: Tuesday, December 4, 2018			
7:00 AM	8:00 AM	Registration • Coffee and Morning Pastries	
8:00 AM	9:00 AM	**ARTIFICIAL INTELLIGENCE LIVE! & SQL SERVER LIVE! KEYNOTE: To Be Announced**	
9:15 AM	10:30 AM	**AIT01** Make Your App See, Hear and Think with Cognitive Services *- Roy Cornelissen*	**AIT02** Introduction to Azure Databricks *- Andrew Brust*
10:30 AM	11:00 AM	Networking Break • Visit the EXPO - *Pacifica 7*	
11:00 AM	12:00 PM	**LIVE! 360 KEYNOTE: To Be Announced** - *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	1:50 PM	**AIT03** Fast Focus: Externalizing Conversational Context *- Heather Downing*	**AIT04** Fast Focus: Jupyter Notebooks for Artificial Intelligence in Azure *- Jen Stirrup*
2:00 PM	2:20 PM	**AIT05** Fast Focus: Supervised VS Unsupervised Machine Learning Approach *- Leila Etaati*	**AIT06** Fast Focus: Career Planning for the Next Era of Analytics *- Jen Underwood*
2:20 PM	2:45 PM	Networking Break • Visit the EXPO - *Pacifica 7*	
2:45 PM	4:00 PM	**AIT07** Speak to Me: Voice App Conversation Practices *- Heather Downing*	**AIT08** Machine Learning with Azure Databricks *- Leila Etaati*
4:15 PM	5:30 PM	**AIT09** Taking IoT to the Edge *- Eric D. Boyd*	**AIT10** Data Wrangler Magic Wand; Power Query in Excel or Power BI *- Reza Rad*
5:30 PM	7:30 PM	Exhibitor Reception - *Pacifica 7*	
Artificial Intelligence Live! Day 2: Wednesday, December 5, 2018			
7:30 AM	8:00 AM	Registration • Coffee and Morning Pastries	
8:00 AM	9:15 AM	**AIW01** Building for Alexa with Web API *- Heather Downing*	**AIW02** Deep Learning in Microsoft Azure: CNTK, CaffeOnSpark and Tensorflow *- Jen Stirrup*
9:30 AM	10:45 AM	**AIW03** Practical Internet of Things for the Microsoft Developer *- Eric D. Boyd*	**AIW04** The Case for R in AI - Statistical Perspective to Data, Inference and Prediction as Used by AI Applications - *Jen Stirrup*
10:45 AM	11:30 AM	Networking Break • Visit the EXPO - *Pacifica 7*	
11:30 AM	12:30 PM	**LIVE! 360 KEYNOTE: Enterprise Transformation (And You Can Too)** *Donovan Brown, Principal DevOps Manager, Cloud Developer Advocacy Team, Microsoft*	
12:30 PM	1:30 PM	Birds-of-a-Feather Lunch	
1:30 PM	2:00 PM	Dessert Break • Visit the EXPO	
2:00 PM	3:15 PM	**AIW05** Building a Holographic Assistant with Bot Framework, LUIS & Mixed Reality *- Nick Landry*	**AIW06** Data Visualization Super Power *- Reza Rad*
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	**AIW07** ML for the .NET Developer *- Matt Winkler & Euan Garden*	**AIW08** Intelligent Reports with Azure Cognitive Services and Power BI *- Leila Etaati*
7:30 PM	9:00 PM	Live! 360 Dessert Luau - *Wantilan Pavilion*	
Artificial Intelligence Live! Day 3: Thursday, December 6, 2018			
7:30 AM	8:00 AM	Registration • Coffee and Morning Pastries	
8:00 AM	9:15 AM	**AIH01** Managing the Conversation Flow within a Bot *- Rabeb Othmani*	**AIH02** Scaling Machine Learning in Azure *- Matt Winkler & Euan Garden*
9:30 AM	10:45 AM	**AIH03** Machine Learning the Easy Way: Azure Cognitive Services *- Eric Potter*	**AIH04** Chest X-ray Image Analysis with Deep Learning *- Vishwas Lele*
11:00 AM	12:00 PM	**ARTIFICIAL INTELLIGENCE LIVE! & SQL SERVER LIVE! PANEL DISCUSSION: Keeping Pace with AI and Machine Learning While Maintaining Your Day Job** *Andrew Brust (moderator); Thomas LaRock, Jen Stirrup, Jen Underwood, & Stacia Varga*	
12:00 PM	1:00 PM	Lunch on the Lanai - *Lanai / Pacifica 7*	
1:00 PM	2:15 PM	**AIH05** DevOps for AI *- Matt Winkler & Euan Garden*	**AIH06** How To Avoid Building Bad Predictive Models *- Jen Underwood*
2:30 PM	3:45 PM	**AIH07** Getting Started with Deep Learning *- Seth Juarez*	**AIH08** Innovations in Automating Analytics and Machine Learning *- Jen Underwood*
4:00 PM	5:00 PM	Next? Live! 360 Networking Event *Heidi Araya, Andrew Brust, Jeremy Clark, Ben Curry, Benjamin Day, Peter De Tender, Brent Edwards, Kevin Ford, Grant Fritchey, Esteban Garcia, Seth Juarez, Dave Kawula, Cristal Kawula, Sami Laiho, Thomas LaRock, Vishwas Lele, Rockford Lhotka, Karen Lopez, Matthew McDermott, Brian Randell, Paul Schaeflein, Jen Underwood, Rob Windsor*	
Artificial Intelligence Live! Post-Conference Workshop: Friday, December 7, 2018			
7:30 AM	8:00 AM	Registration • Coffee and Morning Pastries	
8:00 AM	5:00 PM	**AIF01** Workshop: Practical Artificial Intelligence for the Working Software Engineer *- Seth Juarez*	

Speakers and sessions subject to change

Deploying to Azure App Service and Azure Functions

Stuart Leeks

Microsoft Azure App Service is a Platform-as-a-Service solution that lets you build and deploy Web, mobile and API apps running on any platform. It's also the platform for Azure Functions, which provides an event-driven, serverless compute experience. In this article I'll mostly refer to Azure App Service, but the deployment techniques discussed are equally applicable to Azure Functions.

When it comes to deploying your application, App Service offers a wide range of options from which to choose. In this article I'll take a quick look at the different deployment options and then spend a bit of time delving into the more recent addition of zipdeploy, as well as the Run From Package approach (which is in Preview at the time of this writing).

A Tour of App Service Deployment Options

I've been using App Service for years (since before it was called App Service), but while writing this article I discovered a deployment option I didn't know about: Cloud Sync! So, let's start with that before taking a look at other, more traditionally developer-focused approaches that are part of App Service.

Run From Package is in Preview at the time of this writing, so information regarding this approach is subject to change.

This article discusses:

- Deployment options for Azure App Service and Azure Functions
- How to use deployment slots for atomic deployments
- The new Run From Package approach to deployments
- Using Project Kudu

Technologies discussed:

Azure App Service, Azure Functions, Visual Studio Team Services, Project Kudu

Cloud Sync With Cloud Sync you can connect App Service to your OneDrive or Dropbox account and it will create a folder for your site. You can edit your site content locally and have the OneDrive or Dropbox clients sync to cloud storage. When you want to update your site from cloud storage, you can click the Sync button in the portal. This flow is probably a good fit if you're working on a site with static content and want a way to update with a single click. A full walk-through of Cloud Sync for deployment can be found at bit.ly/2w7C1VK.

Good Ol' FTP If you've been writing Web apps for a while, there's a fair chance that at some point you've used the venerable File Transfer Protocol (FTP) as a way to (ahem) transfer files to a Web server. With App Service, you get an FTP endpoint for your site, which allows you to use your favorite FTP client to connect to and synchronize your content with the site/wwwroot directory. (The endpoint is enabled by default, but you can disable if you wish.)

More details of using FTP for deployments (including using or enforcing FTP over SSL) can be found at bit.ly/2Bw9NtV.

zipdeploy One of the really appealing aspects of zipdeploy is its simplicity. You've probably been creating .zip files for years, and it's just as simple to create .zips across both platforms and continuous integration (CI)/continuous deployment (CD) services such as Visual Studio Team Services (VSTS):

```
# Bash
zip -r <file-name>.zip .
# PowerShell
Compress-Archive -Path * -DestinationPath <file-name>.zip
```

Once you have your .zip file you can deploy it in multiple ways. For example, you can navigate to the Project Kudu site, "https://<app_name>.scm.azurewebsites.net/ZipDeploy," and drag the .zip file, or you can use the Azure CLI:

```
az webapp deployment source config-zip --resource-group myResourceGroup
--name <app_name> --src clouddrive/<filename>.zip
```

Or you can POST an HTTP request with the .zip file to the REST API:

```
curl -X POST -u <deployment_user> --data-binary @"<zip_file_path>"
https://<app_name>.scm.azurewebsites.net/api/zipdeploy
```

All of these methods (including how to invoke zipdeploy from PowerShell) are covered in more detail at bit.ly/2tQwaGD.

When you deploy via a .zip file, you're engaging Kudu's deployment mechanism, which will add the files in the .zip to the site/wwwroot folder of your site in App Service. For more information on Kudu, see "Project Kudu" within this article.

It's also possible to have Kudu execute a deployment script. To do this, you create a .deployment file in the root of the .zip that tells Kudu what script to run. This script can perform an npm install, a dotnet build or whatever steps you require as part of the deployment.

If you're working with a C# project and zipping the source, you can set the SCM_DO_BUILD_DURING_DEPLOYMENT environment variable (via App Settings) and Kudu will automatically build the .sln or .csproj file, providing it's in the root of the folder.

Full details of the .deployment approach can be found at bit.ly/2MosYe4.

Local Git Deployment Git has become a widely adopted tool among developers. One of the features of Git as a version control system is its distributed nature. This means that when you're working with a local repository on your developer machine, the way to synchronize changes to a remote server (such as VSTS, GitHub or BitBucket) is by adding a Git remote and pushing your changes to it.

When you enable the Local Git Deployment option in App Service, a Git repository is created for you. You can configure this as remote from your local Git repository and then trigger a deployment via the "git push" command.

As with zipdeployment, you can execute build steps during deployment. With Local Git Deployment, Kudu will automatically look for .sln/.csproj files in the root and build them. For other languages (or for more control), you can use the .deployment file to control the build process.

If you have richer requirements for build, you can use VSTS to perform the build, as shown in **Figure 1**. This provides access to the build features built into VSTS, as well as custom tasks available in the marketplace.

More details on using Local Git Deployment can be found at bit.ly/2BljdIx.

Continuous Deployment The Continuous Deployment option in App Service is similar to the Local Git deployment. The difference is that instead of the code being pushed to a Git repository that Kudu hosts for you, you push code to your Git repository in VSTS, GitHub or BitBucket. When you configure this option, App Service sets up webhooks to be notified when you push changes to your repository, and this is the trigger for a deployment.

As with Local Git Deployment, you can choose either Kudu builds or VSTS build pipelines.

More details on Continuous Deployment can be found at bit.ly/2v0S23k.

Other Deployments

The options I discussed are those that are directly integrated into App Service. Providers of build and CI/CD platforms may offer their own tasks to enable you to deploy to App Service. Such tasks all build on top of one of the App Service mechanisms described, so any discussion around these App Service methods will apply to those tasks. For example, AppVeyor has a task for deploying to App Service using zipdeployment (bit.ly/20K535j), so the considerations for zipdeployment will apply to the AppVeyor task.

Addressing Challenges

When you deploy your application to App Service, you're modifying the site/wwwroot folder contents. These are the files that are used to serve your site, so there are some challenges you may face. If your site is in use, you may find that some of the files are locked when the deployment attempts to copy new files over. Additionally, because the files aren't all updated at exactly the same moment, it's possible to get unpredictable behavior during deployment.

Using Deployment Slots (in Standard and Premium tiers of App Service) is a great way to overcome these challenges. When you create a deployment slot, you create a new site within your Web App that has its own URL and site/wwwroot content, but with a single operation you can swap your test site slot to become your production slot.

If you create a "test" slot, the flow for deployment might be:

- Deploy to your test slot
- Validate that the new version is functioning correctly
- Swap the test slot to your production slot

If you don't want to perform the validation step in the middle, you can even configure the slot to Auto Swap to production once the deployment has finished.

More details on Deployment Slots can be found at bit.ly/2L4rslt.

A New Alternative

It's common to evolve your deployment as your project progresses, and the range of deployment options available to you in App

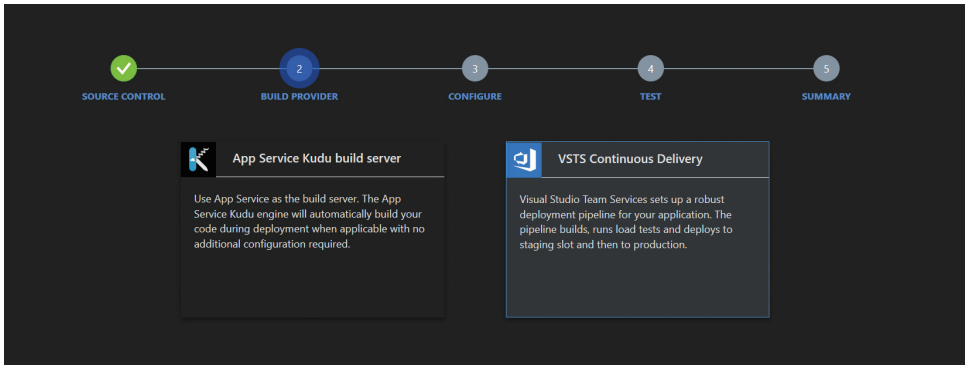


Figure 1 **Choosing Between Kudu and Visual Studio Team Services (VSTS) to Execute the Build Stage**

Project Kudu

Kudu is a built-in component of App Service that combines behind-the-scenes functionality with features that are surfaced in the Kudu portal.

Behind the Scenes Earlier I mentioned how Kudu exposes an endpoint that you can invoke to trigger a zipdeployment. Management of deployments to your App Service is a behind-the-scenes feature of Kudu; for example, when you do a "git push" of your code to the App Service Git endpoint, it's Kudu that takes that code and applies it to your App Service file system.

Kudu also enables the WebJobs capability of App Service (bit.ly/20HHuKz); WebJobs provide the ability to have background tasks that execute in the same context as your Web App. These tasks can be manually triggered, run on a schedule, or can continuously run and retrieve messages from a queue and so forth.

Another behind-the-scenes feature that Kudu provides is Site Extensions. Site Extensions allow you to add extra functionality to your site and can be a combination of configuration changes to your site, a UI that's exposed within the Kudu Portal and WebJobs. You can install Site Extensions from the gallery shown in the Kudu portal, and even write your own (bit.ly/2PioL9z).

Kudu Portal The Kudu portal is quite handy. If your App Service site URL is https://mysite.azurewebsites.net, you can access the Kudu portal at https://mysite.scm.azurewebsites.net (if you're already authenticated with your Azure Portal account, you'll be automatically signed in, otherwise you'll be prompted to authenticate). You can also navigate to Kudu in the Azure Portal by clicking on the Advanced Tools item in the Development Tools section for your App Service or Function App.

After signing in you'll be greeted by the Kudu portal's light but functional UI. Common tasks are in the menu at the top, and links to the Kudu REST API are shown in the body of the homepage. While a full walk-through of Kudu is outside the scope of this article, I want to review a few of the cool features. (You'll also find a lot of great content at the project's Wiki at bit.ly/2KZoQM1.)

One of the features I've used a lot is the Debug Console. When you click on the menu item, you get a choice of command prompt (CMD) or PowerShell. This gives you a combination of folder/file browser interface and an embedded console (and a nice feature is that the current location of each is kept in sync as you navigate), and there's also a simple file editor built in. I've found this to be an extremely useful tool, especially when building WebJobs and Site Extensions.

Another feature that has helped me out on many occasions is the Environment page. This is simply a way to see the App Settings, environment variables and so forth that are configured for your site, but it's handy when you're scripting Kudu-based deployments or even just for a sanity check on the configuration your site is seeing. And there's plenty more in Kudu you'll find useful, from listing processes to triggering a process minidump. If you haven't yet taken a look at Kudu, it's highly recommended.

Service gives you a lot of flexibility over the lifetime of your project. For example, using the .deployment file to specify the build/deployment steps to execute can be hugely productive, but as your project progresses you might want to move to an approach that splits the build process out in order to control it via VSTS or the like. As previously mentioned, all the deployment approaches discussed so far work by syncing content to the site/wwwroot folder of your App Service.

App Service (and Functions) have a new deployment method called Run From Package (in preview at the time of writing) that takes zipdeployment a step further. Unlike all the deployment options discussed so far (including zipdeploy), Run From Package doesn't take the new content and sync it with the site/wwwroot folder. Instead, with Run From Package you provide a .zip file that's mounted as a read-only file system at wwwroot.

Not every application will work with Run From Package; historically, the wwwroot folder has been writable (and ASP.NET Apps in particular are used to writing to the App_Data folder). Also, because the .zip file is mounted, you can't use the .deployment file approach to run build/deployment steps; the .zip file needs to contain the final set of files required to run your site. This can actually be a benefit, though: Because there's no step that syncs files or restores packages, you know exactly what files will be present for a given .zip file and there are no dependencies on npm or NuGet services. As long as you retain the .zip files, reverting back to a previous version of your site is as simple as pointing to the .zip file for that version.

Another key benefit is deployment consistency: Because the .zip file is mounted, there's no syncing of files and the deployment step is atomic. This immediately avoids the concerns I discussed in the "Addressing Challenges" section.

For applications that require loading a large number of files, there can be an initial performance hit when a new host starts running your code (often referred to as a "cold start"); for example, with a Node.js application that has thousands of files in npm_modules. The cold start problem is particularly noticeable in Azure Functions on the dynamic billing plan because it scales aggressively to handle incoming demand, which causes new hosts to be introduced. There are existing ways to mitigate this cold start issue (such as funcpack or webpack), but the way that Run From Package is implemented solves this problem for you.

Wrapping Up

It's clear that App Service enables a wide range of approaches for deploying your application. Across various projects I've worked on with Azure App Service and Azure Functions, I've tried quite a few of these approaches, often using more than one on an individual project as the project's needs evolved over time. The addition of the Run From Package deployment method now provides a simple yet powerful new addition to those deployment choices. For anyone who wants immutable deployments with a robust rollback plan, Run From Package is well worth a look. ■

Stuart Leeks *is a Web and cloud geek who works at Microsoft as a software engineer helping customers and partners with their cloud adoption. You can find him on twitter @stuartleeks and blogging at blogs.msdn.microsoft.com/stuartleeks.*

Thanks to the following Microsoft technical expert for reviewing this article: *David Ebbo*

APPDYNAMICS

Transform the way modern business is built and run

- Application Performance
- End User Monitoring
- Infrastructure Visibility
- Business Performance
- Cloud Migration

appdynamics.com

AppDynamics is now part of Cisco. CISCO

Face Detection Using the Eigenfaces Algorithm on the GPU

Kishore Mulchandani

Computer vision is an area of computer science that involves the identification or labeling of regions in an image. Images are large 2-D arrays of pixels that can be operated upon in parallel. GPUs are ideally suited for accelerating algorithms that entail large amounts of data parallelism, a kind of parallelism in which different portions of a dataset can be operated upon at the same time on multiple processors. Therefore, it should be possible to apply GPUs to the task of implementing efficient computer vision algorithms. In this article I'll use the GPU to implement facial detection, a popular problem in computer vision.

Face detection, as the name suggests, discovers areas in a photo or image that correspond to a human face. This is a critical prerequisite of applications that deal with applying a filter or edit to a face in an image. Automatic blurring, masking, highlighting and enhancing are examples that require knowing which pixels should be modified to complete the operation.

There are several algorithms currently in use to detect faces, but in this article, I'm going to utilize the eigenfaces method, one of the earliest published face detection methods. It's also computationally intensive. My goal is to demonstrate the use of GPUs to accelerate a computationally demanding algorithm, and to present a high-level introduction to programming on the GPU.

The article assumes some familiarity with C-like languages and general concepts of linear algebra and image processing.

This article discusses:

- Computer vision and face detection
- The eigenfaces algorithm
- Programming the GPU
- Implementing face detection with OpenCL

Technologies discussed:

C++, OpenCL, GPU

Code download available at:

msdn.com/magazine/1018magcode

Eigenfaces

From linear algebra you know that if you have a collection of n-dimensional points, you can find some principal components, or eigenvectors, for this collection. These eigenvectors also form the basis of the space of these points. The eigenvector of the highest importance is the one with the largest eigenvalue. Any new point that belongs to this collection should also have a higher projection value on the most important eigenvector.

The eigenfaces method proceeds by converting a collection of faces into n-dimensional points. This is done by converting each two-dimensional image into a one-dimensional vector. The entire collection is packed into a large matrix, where each column is the 1-D vector. Eigenvectors and eigenvalues are found for the square matrix formed by multiplying this matrix with its transpose. The

Figure 1 **The First 100 Images from the Labeled Faces in the Wild Dataset**

discovery of eigenvectors for a collection of points is also called principal component analysis, and the eigenvectors are called principal components.

The starting point for any face detection algorithm is a good database of faces. There are a few available on the Internet, but for this article I made use of the publicly available "Labeled Faces in the Wild" collection (vis-www.cs.umass.edu/lfw). A derivative of the collection converted to grayscale is available for download at conradsanderson.id.au/lfwcrop and is also provided with the download accompanying this article. This collection consists of 13,233 64x64 grayscale images of faces. **Figure 1** shows a subset of the images. Note that all the images have been cropped to include only the face.

A packed matrix, in which each column in the matrix is a 1-D face image from the dataset, is shown in **Figure 2**. Notice the high level of intensity match along the rows of this image.

After the eigenvectors and eigenvalues are found, the most important eigenvector when reshaped as a 2-D image looks like the blurry face in **Figure 3**.

Once the first eigenface has been calculated, the next step is to search the test images for regions where the projection with the eigenface reaches a large value. This large value indicates the test image region and the eigenface are near each other in the n-dimensional space of faces. The search regions will be 64x64 pixel areas starting at every pixel of the test image. Each search is independent of the search at other starting pixels, meaning there's a high level of data parallelism.

As noted earlier, GPUs excel at accelerating parallel computations. But that doesn't mean every algorithm should be implemented on the GPU. The system architecture still has a relatively slow connection between system memory and GPU memory. The data that needs to be operated on must go through a PCI Express bus, which even at PCI-e Gen 4 per lane can transfer only up to 32GT/s, relatively a much slower transfer rate than that between the CPU and system memory. This means that applications that can't tolerate high latency won't run effectively on the GPU. The size of the data set must be large enough to justify the overhead of copying the data from the system memory to the GPU memory and back. If an application has a significantly large workload in terms of data size, the operations to be performed are similar in nature, and all the data needed to perform the task can fit in the GPU memory, then the GPU is an ideally suited processor for the application. As I'll explain, in the case of face detection, the GPU is ideal for part of the processing.

The face detection algorithm has two distinct phases. The first phase consists of creating an eigenface matrix for a collection of faces. Because this collection doesn't typically change, the computation for the eigenface matrix calculation needs to be performed only once per collection. The second phase is the projection phase, in which a test region is projected on the first eigenface to see how large the projection is. This phase tends to happen quite frequently. For example, in a security camera application this phase may occur at 30 frames per second, producing large amounts of data. The resolution of the images could also be very large as even the cheapest webcams nowadays support HD resolutions. Moreover, face detection programs are often running with other graphics operations, such as blurring or highlighting, and the results of the face detection are being consumed on the GPU by another part of the program.

Thus, phase one is not a good candidate for GPU implementation, given the low frequency with which it needs to be performed, while phase two is an excellent match for a GPU implementation as it satisfies all the criteria.

In the following sections I'll show you how to do the phase two of face detection, but first let me expand on how programming the GPU differs from that on the CPU.

Programming the GPU

The paradigm of programming the GPU is quite different from that of the CPU. To the OS a GPU appears as a device, and its installed drivers manage the execution of work on this device. The OS and the GPU drivers work together to help complete the tasks. At the lowest level, a sequence of commands is sent to the GPU device through its command queues. The commands may involve setting

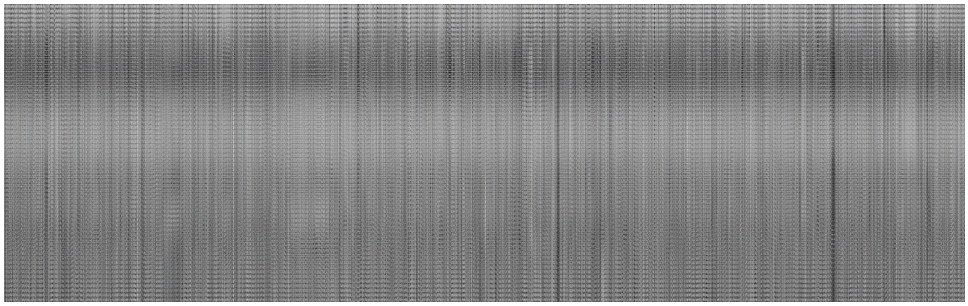


Figure 2 **Scaled Representation of the Big Matrix with 4,096 Rows and 13,233 Columns**

state, copying data between CPU memory and GPU memory and running code. GPUs have their own instruction language, quite different from the CPU language. Luckily, that's not something you need to know unless you're developing GPU compilers. Most programmers use one or more APIs that GPU vendors implement in their drivers, such as Direct3D, OpenCL and OpenGL. These APIs hide the low-level complexity while providing portability and compatibility across different vendors and families of architectures.

> For this article I've chosen to use OpenCL, a standard that's supported by all the major CPU and GPU vendors.

For this article I've chosen to use OpenCL, a standard that's supported by all the major CPU and GPU vendors. OpenCL has been around since 2009, has had several versions, the latest being version 2.2, and it continues to evolve (khronos.org/opencl). A very important fact to note about OpenCL is that it can target not only the GPU, but also the CPU. This is a very big plus for OpenCL as it means you can make use of all the compute resources on a system. For this reason, OpenCL is known as a parallel programming API for heterogeneous systems. An OpenCL host program can explore the availability and capabilities of the computational devices before deciding which ones to use.

Fortunately, the programming model and abstractions are very similar among the various APIs. This means you should be able to map the concepts learned in OpenCL to other APIs without difficulty. Let's now dive into an OpenCL implementation for face detection.

Setting Up the Environment

To develop with OpenCL, in addition to the runtime support in the driver, you need the SDK from the appropriate vendor. See the documentation in the digital download accompanying this article.

Because OpenCL is an open standard, any vendor can provide their own version of the runtime optimized for their hardware.

The OpenCL programming vocabulary consists of platforms, devices and command queues. A platform is synonymous with an implementation available on a system. A device refers to a piece of hardware capable of being scheduled independently from other devices. Every computer has at least one CPU device. Most workstations or gaming-class computers will have a discrete GPU that will be listed as an additional device. High-end engineering workstations often have multiple GPUs, with one attached to display while others are reserved for computational use. Each device can have multiple command queues into which a program will insert commands.

Every OpenCL program, running on a host, begins by discovering the capabilities of the system on which it's running. The host program tries to identify the platforms available on the system, then it enumerates the devices. The program can specifically request a device type, for instance a CPU device type or a GPU device type. Once a device has been chosen, you create a context and a command queue for each device you intend to use. Here are some of the calls made to perform the aforementioned operations (for clarity I've excluded the parameters required for the calls):

```
clGetPlatformIDs(...); // Gets the ids of the platforms available
clGetDeviceIDs(...); // Gets the devices with certain capabilities
clCreateContext(...); // Creates a context on the device
clCreateCommandQueueWithProperties(...); // Creates a command queue with
                                         // certain properties
```

Once the command queue is set up, the host program creates buffers on the device that can be read-only, write-only or read-write. The buffers must be initialized with data from system memory before they can be used to do anything meaningful:

```
clCreateBuffer(...);  // Creates a read only buffer on the device
clEnqueueWriteImage(...);  // Queues a buffer copy from the system to
                           // device memory
```

The data tends to be large buffers or arrays of integer and floating-point values and is usually copied asynchronously. The buffers where the computational results are to be stored are similarly created on the device. OpenCL has special data types called Images. Many of the intensive tasks usually originate in the form of images or 3-D datasets where these special datatypes come in handy. These types support various sampling operations, such as nearest sample or averaging.

The programs, or kernels as they're referred to in OpenCL terminology, must be compiled and linked and queued in the command queue. I'll elaborate on this aspect later, as a CPU programmer might wonder why the kernels are compiled at run time:

```
clCreateProgramWithSource(...); // Create a program from kernel source
clBuildProgram(...); // The program is compiled; errors from compilation
                     // are reported
clCreateKernel(...); // The kernel is created from the program
```

Once the kernel has been created, the arguments for the kernel are set up, and the kernel is then queued for execution in the command queue:

```
clSetKernelArg(...); // Set the value of a particular argument
clEnqueueNDRangeKernel(...); // Queue the kernel
```

There are two kernels in the accompanying source code, one that does the projection of a region on the eigenface, and the other that

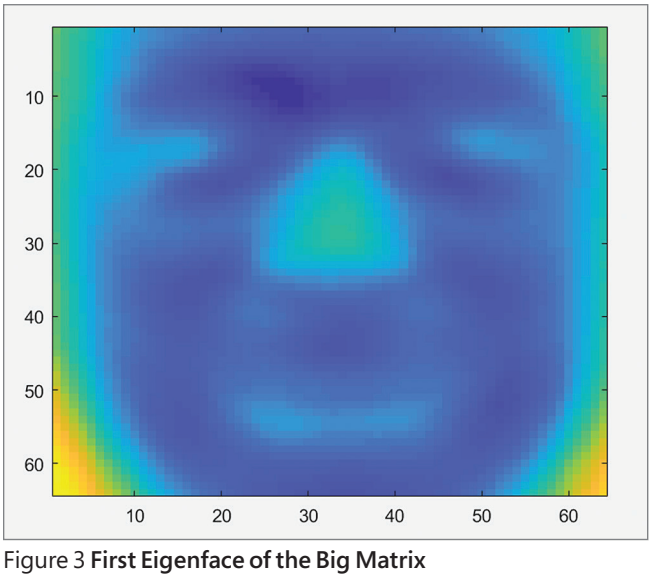
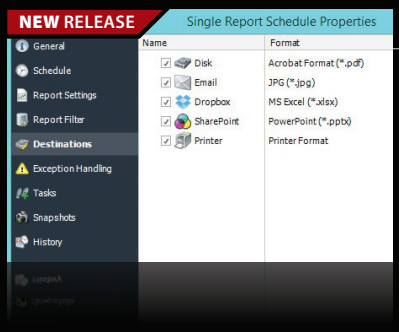


Figure 3 **First Eigenface of the Big Matrix**

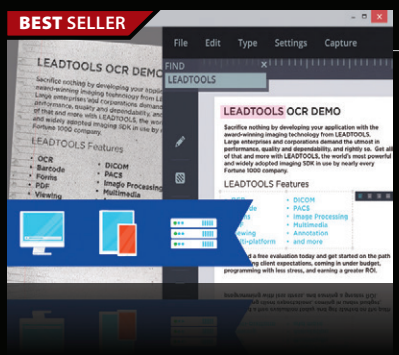


PBRS (Power BI Reports Scheduler) | from $8,132.21

Christiansteven

Schedules & delivers Power BI reports to unlimited recipients with one Power BI License.

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



LEADTOOLS Document Imaging SDKs V20 | from $2,995.00 SRP

LEADTOOLS
THE WORLD LEADER IN IMAGING SDKs

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

- Universal document viewer & conversion framework for PDF, Office, CAD, TIFF & more
- OCR, MICR, OMR, ICR and Forms Recognition supporting structured & unstructured forms
- PDF SDK with text edit, hyperlinks, bookmarks, digital signature, forms, metadata
- Barcode Detect, Read, Write for UPC, EAN, Code 39, Code 128, QR, Data Matrix, PDF417
- Zero-footprint HTML5/JavaScript UI Controls & Web Services

DevExpress DXperience 18.1 | 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

Help & Manual Professional | from $586.04

ecsoftware

Help and documentation for .NET and mobile applications.

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

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

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

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

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

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

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

GSA Schedule Contract GS-35F-0188R

finds local peaks from the results of the projection. The projection operation is essentially a vector dot product.

Large values for the dot product imply more similarities between the region in the image and the eigenface. These large values or peaks are found using the second kernel. Because the results of the first kernel must be available before the second kernel can use them, OpenCL events are used to synchronize. The host program can wait on one or more events using the following API call:

```
clWaitForEvents(...);
```

Once both the kernels have been executed on the device, the results of the second kernel will be in a device buffer that's then copied back to the host memory where it can be used further.

Now let's look at one of the kernels that's launched in **Figure 4**. This kernel is called for every pixel in the test image. It first finds the sum of the 4,096-dimensional vector (64x64) of the region starting at that pixel and then subtracts a factor of the sum from each pixel of the region. It then computes the dot product of this region with the eigenface that's passed into the kernel as an argument.

Compiling the OpenCL Kernel

As mentioned previously, the OpenCL compiler is invoked at run time by the host program. The compiler is supplied with the entire source of the kernel as a string. Frequently, kernels are rather tiny pieces of code that compile quite fast, and unless your application is sensitive to every extra millisecond it takes to get the results back, you probably won't care about it. But for complex applications with hundreds of kernels, the delay could be noticeable. Fortunately, intermediate language (IL) options exist that can reduce the overall compilation times. (IL also helps in protecting the intellectual property as embedded strings of kernel code might give away proprietary information.) The benefit of the runtime compilation of kernels is that it makes the application resilient to changes in hardware. It's quite common for PC users to upgrade their GPUs during the lifespan of their computers. A new GPU with a new architecture might have a completely new instruction set, leaving precompiled binaries from an older GPU unable to work.

Piecing It Together

The eigenface image is precomputed and supplied as a binary file containing single-precision float data in a row-major order. The test images are also converted to floating-point grayscale images. The test images have the following format:

```
Height: 4 byte unsigned integer
  // The number of rows in the image
Width: 4 byte unsigned integer
  // The number of columns in the image
Intensity Buffer:   Height * width single precision floating point buffer
  // Data buffer
```

With such a simple format, it should be straightforward to generate test images that can be consumed by my program. The results of the computation are also saved out as buffers of floating-point data. Additionally, a text file with coordinates of the top-left corners of all rectangles containing faces is generated. This data can be imported into any program. I save the data for consumption in MATLAB (bit.ly/2isajNJ).

The program can be easily modified to run on the CPU rather than the GPU.

Results of the Run

I ran the program on several test images. Because the code currently searches for only 64x64 regions, if the test images have much larger or smaller faces, I wouldn't expect it to find good matches. However, when the test images do have approximately the same size as the eigenface, unless the faces are rotated, it should do well. Results of my run on one famous image are shown in **Figure 5**. Faces were identified using the eigenfaces method with unnormalized projection of region on the first eigenface.

I got quite a few false positives and false negatives. Several faces were missed, and many non-face areas were highlighted. Some of this is a result of the local maxima search method I used to detect the high values (peaks) in the projection image. I simply searched the neighborhood of 97x97 pixels to see if I'd find any higher value. The result was several areas were highlighted where there were no faces but still had a higher value than its local neighborhood. This method could be improved upon to reduce the false positives.

Keep in mind that the quality of results depends on the quality of the example images, which includes things such as the image resolution, the lighting conditions and the variability in the intensities.

Figure 4 **The OpenCL Kernel for Creating an Eigenface Projection on a Region in the Test Image**

```
__kernel void eigenFaceSearch(
    __read_only image2d_t testImage,        // Input test image
    __write_only image2d_t projectionImage, // Output projection image
    int rows,                               // Number of rows in the test image
    int cols,                               // Number of columns in the test image
    __constant float* eigenFace,            // The first eigenface buffer
    sampler_t sampler                       // Sampler for reading the image data
    )
  {
    int myCol = get_global_id(0);           // Column index of test image pixel
    int myRow = get_global_id(1);           // Row index of test image pixel
    float4 dotp = {0.0f, 0.0f, 0.0f, 0.0f}; // Resulting dot project variable
    float4 pixel = {0.0f, 0.0f, 0.0f, 0.0f};// Variable for reading the pixel
    float sum = 0.f;                        // Sum of the pixels in the test region
    float meanVal=0.f;
    float patch[4096];                      // Local buffer to hold the mean
                                            // subtracted region
    int2 myCoords;                          // Local variable for holding current
                                            // pixels coords
    myCoords.x = myCol;
    myCoords.y = myRow;
    if (!(( myCol > cols-64) || (myRow > rows - 64))) { // Bounds check
      for (int i = 0; i < 64; i++) { // Loop over every pixel in test region
        int2 coords;
        coords.y = myRow + i;
        for (int j = 0; j < 64; j++) {
          coords.x = myCol + j;
          // Read the pixel at coords, from the test image using the sampler
          pixel = read_imagef(testImage, sampler, coords);
          patch[i*64+j] = pixel.x;  // Take the first component as it is a single
                                    // float buffer
          sum += patch[i*64+j];     // Accumulate the sum
        }
    }
    meanVal = sum / 256.;           // An intensity factor proportional to sum.
    for (int i= 0; i < 4096; i++){  // Remove from each pixel
      patch[i] -= meanVal;          // Per component subtract of intensity factor
    }
    // Perform the dot product
    for (int i= 0; i < 4096; i++){
      dotp.x += patch[i] * eigenFace[i]; // Do the component-wise dot product
    }
    // Write back the dot product into the projection Image
    write_imagef(projectionImage, myCoords, dotp);
  }
}
```

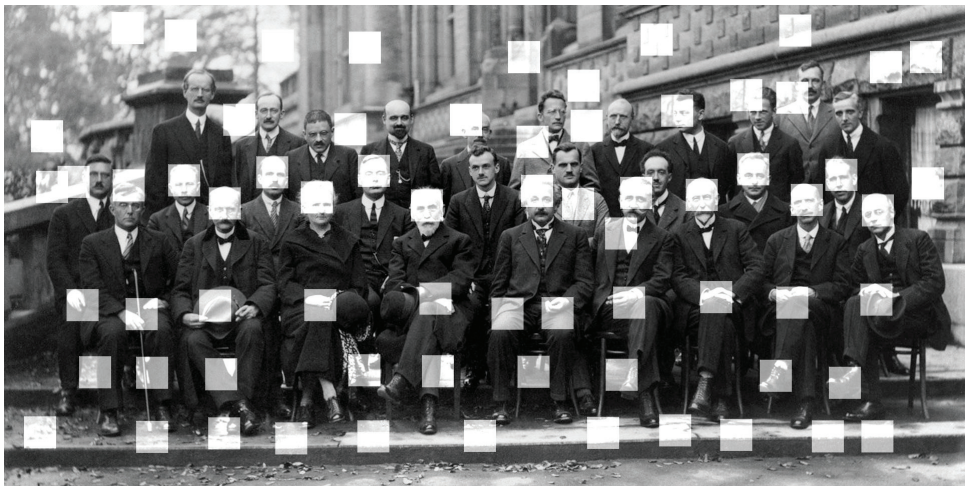


Figure 5 **A Photograph Taken at the Fifth Solvay Conference held in 1927**

OpenCL on GPU vs. CPU

There are several parallel processing technologies on the CPU front, from SIMD vectorization, to explicit multithreading, to compiler pragmas in the OpenMP style parallelization of loops. To truly compare what's faster or has higher throughput would be very difficult unless every possible combination of technologies was tested with extensively optimized implementations. That said, given how easy it is to switch from one device type to another in code, I decided to give it a try.

I ran the program on both device types, the CPU and the GPU, and from two different "platforms." The system I ran this on was a gaming laptop with an Intel Core i7-8750H CPU @2.20 GHz with 16GB RAM with an NVIDIA GeForce GTX 1060 GPU. I could've chosen a CPU device on any of the three platforms, or a GPU device type on one from NVIDIA. Note that a GPU vendor might supply a CPU implementation along with its GPU implementation, in which case it's less likely to be optimized.

As expected, the GPU performed better, with the CPU version taking 12 to 21 times longer, as shown in **Figure 6**. The profiling didn't include any I/O but did include the time to copy the buffers from the system to the device memory. All times are in milliseconds.

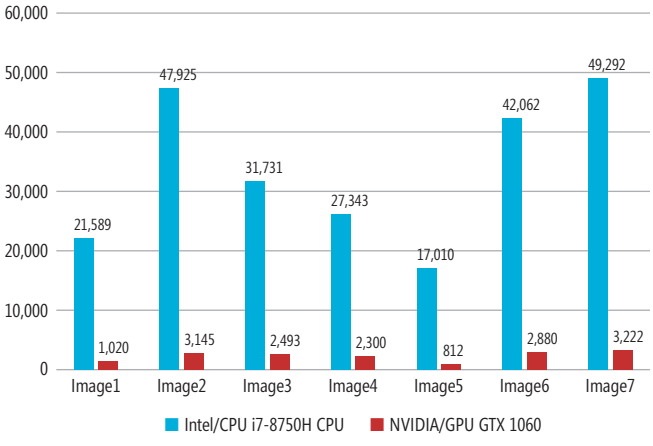


Figure 6 **Comparison of GPU and CPU Eigenface Search Implementation Times Using OpenCL**

The kernels were written with the intent to learn and not to get the best possible performance, so take these comparisons with a grain of salt. Indeed, the performance tuning of kernels is an involved process, and an important step prior to deploying a program. Pipelining buffer copies, reducing the number of registers in kernels and keeping memory accesses coherent are some techniques employed to optimize the kernels.

The fact that the same OpenCL host program can simultaneously queue work for all the different devices available on a system means you can further increase throughput by employing all these devices. If you're running face detection on video security footage, for example, you could divide each frame or subframe among the different device queues proportionate to their throughputs. The system memory and system buses are still shared and could become bottlenecks, so, unfortunately, you can't scale the performance indefinitely by adding more devices.

Eigen Anything

Though this article focused on detecting the faces of people, there's nothing that prevents this approach from being applied to detecting other entities or objects. The trick is to collect a good number of test images, do some preprocessing to make them all the same size, register the location of some salient feature and then calculate the eigenvectors. Once the eigenvectors have been created, the search code remains unchanged. So, if you're developing code to detect hands for a nail polish-visualizing application, you could take a database of hands and generate the "eigenhands" and search in an image for where the hands might be. With augmented reality applications on the rise, these types of applications are likely to appear.

Wrapping UP

For a programmer delving into GPU development for the first time, there could be a steep learning curve, but the rewards can be well worth the investment. GPUs have significant amounts of compute power that can be harnessed with programming languages such as OpenCL. This article serves as an introduction to GPU programming while demonstrating the popular real-world problem of finding faces in an image using the eigenfaces approach. This approach can be extended to other shapes and objects. ■

Kishore Mulchandani *develops 3-D graphics applications and does performance improvements in parallel code running on CPUs and GPUs. High-performance computing, benchmark development for graphics, 3-D shape reconstruction from 2-D image data using computer vision, and virtual reality edutainment applications are some of his latest interests and endeavors. He can be reached at Kishore@vanishinglines.com.*

Thanks to the following Microsoft technical experts for reviewing this article: *James McCaffrey, Ravi Shankar Kolli*

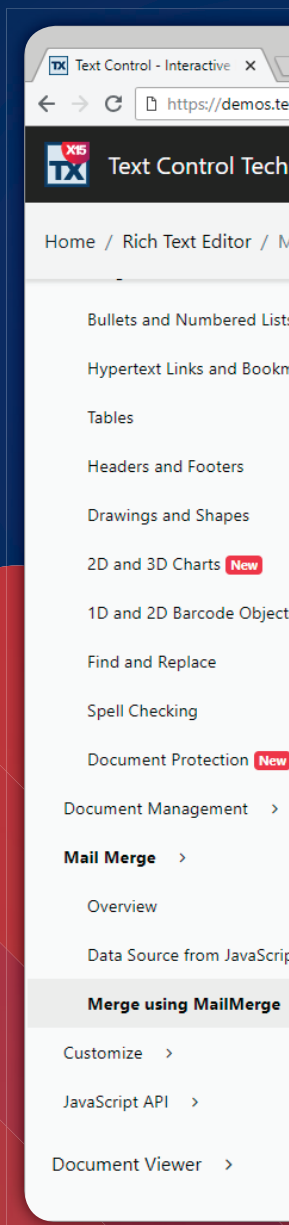
TEXT CONTROL

Integrate Documents and Reporting in Any Platform

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

See our technology live:

demos.textcontrol.com



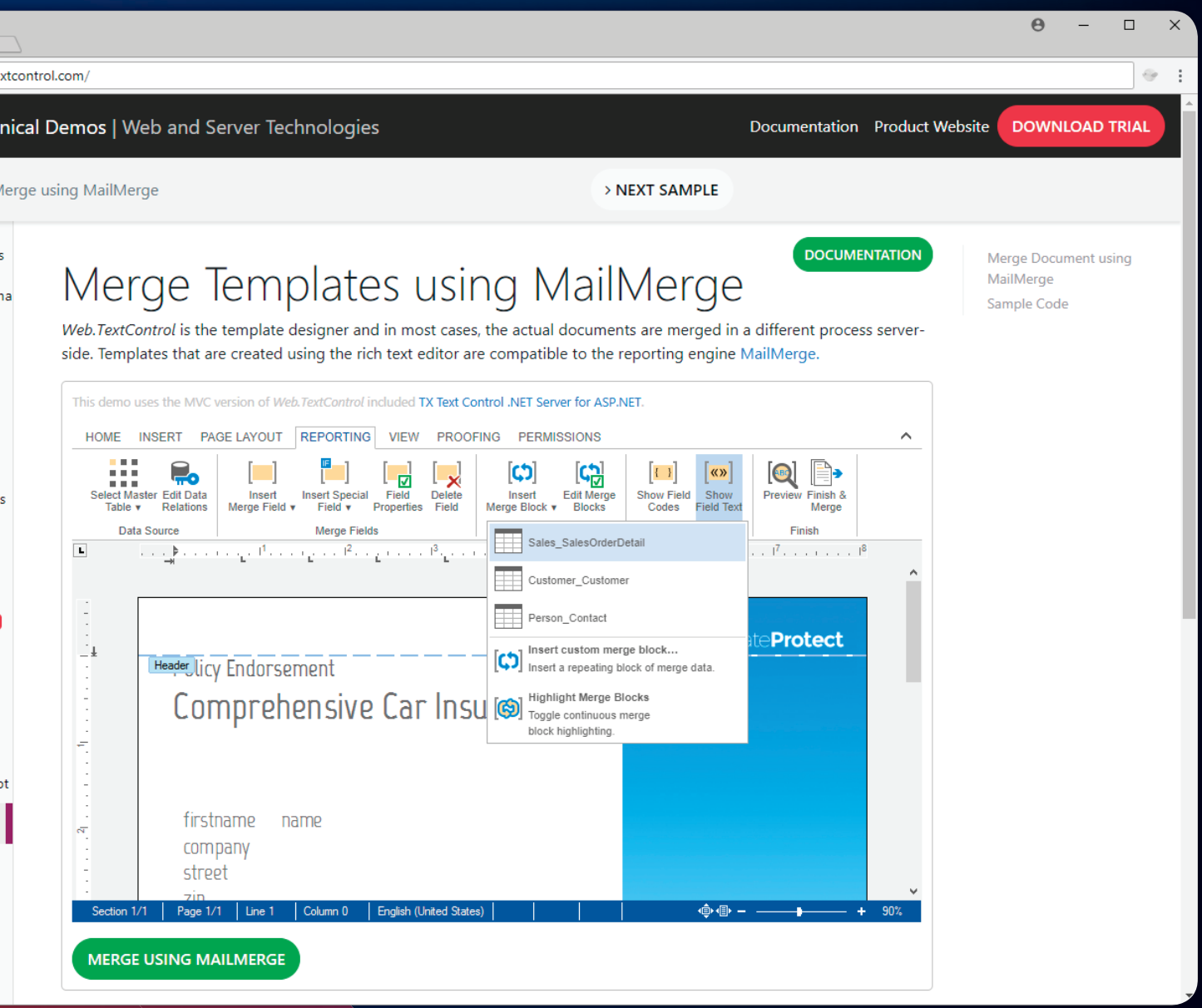
WE ARE CHANGING THE WAY YOU LOOK AT REPORTING

Visual Studio
Partner

SEE THIS LIVE

TX Text Control X15 Technical Demos

Evaluate our technology and test the most sophisticated, cross-browser, and true WYSIWYG, rich text editor. Merge MS Word compatible templates with JSON data and create pixel-perfect Adobe PDF documents on-the-fly. See what's possible today!



How to Contribute to Microsoft Open Source Software Projects

Mark Michaelis

Here's a fact: Microsoft hosts around 2,000 open source software (OSS) repositories on GitHub, including some rather large ones like the .NET Compiler Platform, also known as "Roslyn," which has as many as 4 million lines of code. For many developers, the prospect of submitting code changes to a project that runs on millions of computers might seem daunting. Fortunately, you don't need a Ph.D. in programming languages and compilers to make your mark on Microsoft OSS projects. There are opportunities to contribute that span a broad spectrum of difficulty and experience, from beginner to expert.

I got my start in March of 2018 working with the .NET Core team to add a new set of APIs. I was able to jump on board thanks to my connections at Microsoft, specifically with Program Manager Kathleen Dollard. At the time I wondered, "How hard would it be for someone that wasn't well connected at Microsoft to get involved?" With this question in mind, I decided to do some research and find out. In this article I explore the topic of contributing to Microsoft OSS and what it takes for newcomers to get involved.

This article discusses:

- Getting started with a Microsoft open source software (OSS) project
- Programming and workflow guidelines for OSS projects
- Engaging in design discussions on strategic projects like C#, CoreFX and Dotnet CLI repositories

Technologies discussed:

.NET Core, GitHub

Getting Started: Documentation and Pull Request Review

Perhaps the best place to begin is with documentation. If you navigate to any of the .NET documentation pages (for example, bit.ly/2LAv7hA) you'll notice at the bottom of each page that there's a footer soliciting feedback, as shown in **Figure 1**.

From here you can click "Product feedback," submit a new issue, or browse and search existing issues. Even better, the second button takes you directly to the GitHub issue list for the specific page you were browsing, so you can either create a GitHub issue or navigate to the documentation source code itself (such as github.com/dotnet/docs) and fix the issue directly. Frequently, just updating the documentation and submitting a pull request (PR) is easier than the work required to document the problem.

I've spoken with team members directly and they emphasize that all submissions are welcome, even spelling and grammar corrections. These changes may not be exciting, but they can make the difference between a successful API and an unsuccessful one.

Further, the docs team is one of the fastest to respond to raised issues and PRs, with staff assigned to each area to address contributions no matter how small. One reason documentation editing is easy: It typically doesn't require you to clone the repo to submit changes. Rather, you can use the GitHub Web-based editing UI, which automatically forks and submits the PR for you.

PR review is also an important way to contribute. Every project needs PR feedback and the Microsoft team is grateful for PR review contributions. I know I've appreciated—and learned from—reviews

of works I've submitted to .NET. The biggest lesson for me is that I can't jump in and make a significant contribution casually or in the slivers of time between doing other things. Code at this level requires careful thought, careful implementation and significant collaboration. In the end, I'm grateful for both the PRs rejected and the PRs accepted. PR reviews are a great place to step in and help with open source development.

First-Time Approachable Issues

For those ready to take on more than documentation, Microsoft offers some guidance. Issues that are tagged with descriptors like "first issue" and "easy" are great candidates for developers who are new to the game. Microsoft even asks active project members to steer clear of first-time approachable issues until near the end of a release, thus keeping easier issues available for developers new to the project and lowering the complexity bar for getting started. Furthermore, first-time approachable issues frequently document links that describe where to look for the problem, rather than leaving new developers flailing in an attempt to locate a specific defect in the haystack that's a large repo. For the Roslyn team, for example, a good first issue must include:

- Links to the file where the fix is likely to be necessary
- Identification of where the tests need to go
- Setup instructions for getting started with Roslyn
- General contribution policy

The commitment to favor contributions from newcomers extends to how PRs are accepted. For PRs tagged as a "good first issue," Microsoft gives preference to new contributors, accepting their PRs over those from existing contributors. What's more, the Microsoft staffer who tags an issue as a "good first issue" will directly answer questions from people working to fix the issue. That bit of hand-holding can be important in the first stages of involvement.

Clearly, Microsoft goes out of its way to get first-time contributors engaged. Roslyn, for example, is a complicated, 4-million-line code base that's not for the faint of heart. By engaging new contributors, Microsoft enables an active community of external developers in its OSS efforts.

Regular Contributions

Once you have your first PR accepted, you're likely going to take on more complex issues and features. There are issue tags like "help wanted" and "up for grabs" that indicate that an issue is likely a good target for the community—although not necessarily ideal for first-timers. (See up-for-grabs.net to browse through the list of projects and corresponding issues tags such as for first-time approachable or help wanted.) Issues labeled with these tags likely involve more work or greater knowledge of the project than a newcomer can provide; however, they're well defined and don't need extensive collaboration with the project team. On the other hand, there's a defined workflow that you would be wise to follow:

- Contributions beyond the level of a bug fix should be discussed with the team and within the scope of the roadmap to avoid being declined
- Contributions should be against master—not against an experimental feature branch
- PRs must merge easily with the tip of the master branch
- Contributors should make sure to sign the Contributor License Agreement (see cla2.dotnetfoundation.org)

As developers experienced with Git would expect, be sure that you work in a local fork (cloned to your computer) and then submit code for consideration via a PR. Of course, you can create a branch locally, but when you submit your PR you'll be submitting to master.

There are a few programming and workflow guidelines to keep in mind. First, there's the coding style to consider. While you can find the C# Coding Style at bit.ly/2woQv3u, the general summary is to follow the standard of the existing file. This is true even if the existing file differs from the documented standard. This means that until you're coding entire files (or adding items for which there's no precedence already in the file), the guideline is easy—follow the example of the rest of the file. Even without precedence, however, there are only 16 items listed in the C# coding style document, none of which are particularly surprising. These items include:

- Curly braces on their own line
- Fours spaces for indentation (not tabs)
- _camelCase for internal private fields and PascalCasing for constant local variables and fields
- Avoid this. unless required
- Always specify the visibility (that is, use private even if the member defaults to private)
- Avoid more than one empty line to break up the code
- Only use var when the assigned type is obvious (see itl.tc/UsingVar), with the exception of Roslyn projects, which use var everywhere
- Specify fields at the top within a type declaration

Generally, you'll find an .editorconfig (see editorconfig.org) settings file for each directory, enforcing these standards. Be sure to use the file to ensure you follow the guidelines and avoid having your PR blocked.

For those coding in Visual Basic, follow the spirit and intent of the C# guidelines.

Although not mentioned in the list, unit tests are critical to producing the required level of quality.

Design Help

Some repositories like language, CoreFX, and Dotnet CLI require a significantly greater level of experience and expertise and, as such, employ a different process. With these libraries, the entry point is at the discussion level rather than the code level. Directly submitting a code PR to these libraries with a new feature or language keyword is unlikely to be successful.

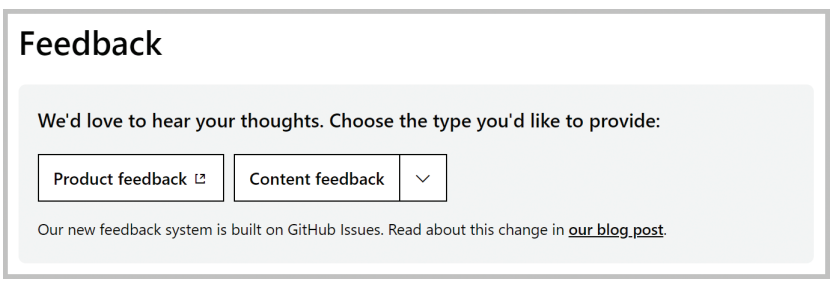


Figure 1 **Submitting Suggestions and Changes to Documentation**

While the design process is generally visible, it's not a free-for-all. In fact, submissions for these repositories don't even start with proposals. (Check out the C# language proposals folder at bit.ly/2BVUbjf for a good look at the main features currently under consideration.) Rather, if you want to suggest a feature, you start by submitting an issue and tagging it with the "Discussion" label. If discussion items reach some level of consensus, such that further evaluation should be considered, they're picked up in the Language Design Meetings, which in turn are informed by further discussion, experiments and offline design work. Proposals themselves—not finalized features—are then championed by members of the language design team.

While the process is intended to be open to feedback, not everyone can just make changes however they choose. The volume of distribution and the impact of change is too great to not be tightly controlled (very similar to the control that Linus Torvalds has with Linux). In the end, the project is still open source. If you want the freedom to change the code in whatever way your heart desires, simply fork the repo and get started.

This approach is an important way to collaborate well before code implementation begins. Even then, changes are likely to be in a separate branch for an extended period while they're programmed (and repeatedly re-programmed) and evaluated. The community is essential in deciding what the shape of something is going to be. Opening a discussion issue, commenting, and providing feedback with existing proposals provides direct access to the team.

You'll observe that contributing can run the gamut from simple to extremely difficult. Adding a method or class to an API, for example, is one thing, but adding a new language keyword is something else entirely.

What Happens After Submitting a PR

In April of 2018, the Roslyn team realized that they had fallen behind on processing all the PRs that had been submitted. With all the changes that had occurred after the PRs, it was likely that some of them would no longer be relevant. To address the problem, Microsoft stepped up and assigned staff to each project. It was their job to respond to all future PRs to ensure that the effort put in was more likely to lead to success. Toward that effort, they put in place the following categorization of PRs:

- **Approved by Project Lead:** Approved PRs are assigned a buddy or coach from the project team to improve the chances of successful adoption and help incorporate the PR into the code base. The coach ensures that community members are engaged, following up with contributors within three business days if the PR is rejected for whatever reason.
- **Pending Discussion:** Sometimes significant concerns crop up—unit tests are missing, the purpose is unclear or the code fails significantly to meet the guidelines. In these instances, the project lead raises the concerns with the community contributor, identifying what needs to happen. The onus is on the contributor to follow up within two weeks. Furthermore, PRs in this grouping need to keep up with the code base during this time.
- **Rejected:** The PR isn't in line with the vision of the project, involves too much risk or doesn't successfully solve a priority. In this case the lead will reject the PR, clearly identifying the issue. While the PR can be resubmitted, it will require significant change or rework.

Wrapping Up

Occasionally you can observe behavior within open source projects that's well below the standards of decency. This can include participants who are rude, intolerant or repeatedly fail to listen to opposing views. It also includes contributors who fail to accept Microsoft's role as the final arbiter of Microsoft OSS projects, repeatedly spam a repo or otherwise disrupt the collaborative process. Anyone that can't follow the rules of general civility will find themselves blocked from a repo (and returning under a pseudonym with the same behavior will not get you any further). Microsoft is committed to making participation a positive experience for all, and enforcing the Code of Conduct is core to that commitment.

I encourage you to review the Microsoft Open Source Code of Conduct at bit.ly/2wmAYlB. Also review its associated FAQ at bit.ly/2NwNNRa.

In my experience, how you approach making changes to Microsoft OSS depends largely on what generates your interest in wanting a change. I expect that for most of you the catalyst is a problem in the form of a defect or a missing feature. Initially, the trigger may be a flaw or issue in the documentation and the desire to fix it for other readers. Alternatively, perhaps you're working in the Xamarin code base and discover that a method you wish to override isn't virtual, so you submit a PR to make it such.

Some of you will want to take on an even bigger challenge. With .NET Core, I've been astounded by the fact that there (still) isn't one command-line parser that can easily accept the command-line arguments and parse them into a strongly typed object from which I can access the values in my program. It was this itch that prompted me to start collaborating with Microsoft's Jon Sequeira (who wrote the command-line parser for Dotnet CLI) to build such a parser. Alas, the code is still too unstable and our participation too casual for us to take the project open source. Hopefully it won't be too long before this project is something we can open to the public, so it too can benefit from the engagement of the OSS community. In the interim, if you have significant time to dedicate and have an interest in our parser project, send an e-mail to Kathleen or me and we can work out a way to get you involved. And, yes, I just introduced yet another way to contribute—volunteering before the code is public. ■

Mark Michaelis *is founder of IntelliTect, where he serves as its chief technical architect and trainer. He's been a Microsoft MVP for nearly two decades, and a Microsoft Regional Director since 2007. Michaelis serves on several Microsoft software design review teams, including C#, Microsoft Azure, SharePoint and Visual Studio ALM. He speaks at developer conferences and has written numerous books including his most recent, "Essential C# 7.0 (6th Edition)" (itl.tc/EssentialCSharp). Contact him on Facebook at facebook.com/Mark.Michaelis, on his blog at IntelliTect.com/Mark, on Twitter: @markmichaelis or via e-mail at mark@IntelliTect.com.*

Thanks to the following Microsoft technical experts for their help collaborating and reviewing this article: *Kevin Bost (IntelliTect), Kathleen Dollard, Neal Gafter, Sam Harwell, Immo Landwerth, Jared Parsons, Jon Sequeira, Bill Wagner, Maira Wenzel*

DocuVieware

Universal HTML5 and Document Management Kit

Easy integration

Full support for custom snap-in

Zero-footprint solution

Fully customizable UI

Mobile devices optimization

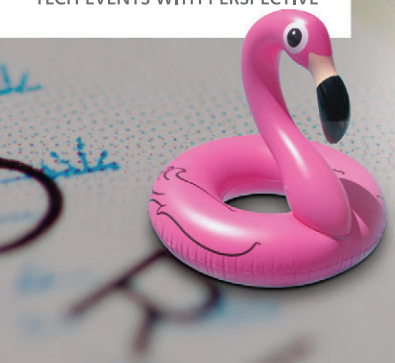
Fast & crystal-clear rendering

Check the New Features and the Online Demos
60-day Free Trial Support Included at **www.docuvieware.com**

Powered by **GdPicture**.NET

DocuVieware is an ORPALIS imaging technologies product

LIVE! 360
TECH EVENTS WITH PERSPECTIVE



Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

ROYAL PACIFIC RESORT AT UNIVERSAL ORLANDO
December 2-7, 2018

Coding in Paradise

Developers, engineers, software architects and designers will code in paradise this December at Visual Studio Live! (VSLive!™). Help us celebrate 25 years of coding innovation by returning to warm, sunny Orlando for six days of unbiased and cutting-edge education on the Microsoft Platform. Soak in the knowledge on everything from Visual Studio and the .NET framework, to ASP.NET, .NET Core, JavaScript, Xamarin, Database Analytics, and so much more. VSLive!™ features 60+ sessions led by industry experts and Microsoft insiders.

Grab your flip flops, and your laptops, and make plans to attend the conference more developers rely on to expand their .NET skills and the ability to build better applications.

REGISTER NOW

SAVE $400 WITH SUPER EARLY BIRD PRICING!
REGISTER BY OCTOBER 5 Use Promo Code MSDN

A Part of Live! 360: The Ultimate Education Destination

6 CONFERENCES, 1 GREAT PRICE

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

VSLIVE.COM/ORLANDO

EVENT PARTNERS
Microsoft
Magenic

SILVER SPONSOR
Progress
GOVERLAN REACH

PREMIER & ALLIANCE MEDIA PARTNERS
netda
UserGroup.TV

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

PRODUCED BY
CONVERGE 360
An 1105 MEDIA company

AGENDA-AT-A-GLANCE

DevOps in the Spotlight	Cloud, Containers and Microservices	App Dev Data	Developing New Experiences	Delivery and Deployment	.NET and More	Full Stack Web Development

START TIME	END TIME				
Visual Studio Live! Full Day Hands-On Labs: Sunday, December 2, 2018					
9:00 AM	6:00 PM	**VSS01** Hands-On Lab: Develop an ASP.NET Core 2 and EF Core 2 App in a Day - *Philip Japikse*	**VSS02** Hands-On Lab: From 0-60 in a Day with Xamarin and Xamarin.Forms - *Roy Cornelissen & Marcel de Vries*	**VSS03** Hands-On Lab: Busy Developer's Workshop on VueJS - *Ted Neward*	
2:00 PM	7:00 PM	Pre-Conference Registration - Royal Pacific Resort Conference Center			
Visual Studio Live! Pre-Conference Workshops: Monday, December 3, 2018					
8:30 AM	5:30 PM	**VSM01** Workshop: Web Development in 2018 - *Chris Klug*	**VSM02** Workshop: Architect and Build a Modern ASP.NET App in the Azure Cloud with a full CI/CD Pipeline with VSTS - *Brian Randell and Miguel Castro*	**VSM03** Workshop: Cross-Platform C# Using .NET Core and WebAssembly - *Rockford Lhotka & Jason Bock*	
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			
Visual Studio Live! Day 1: Tuesday, December 4, 2018					
8:00 AM	9:00 AM	**VISUAL STUDIO LIVE! & MODERN APPS LIVE! KEYNOTE: .NET Everywhere and for Everyone** *James Montemagno, Principal Program Manager - Mobile Developer Tools, Microsoft*			
9:15 AM	10:30 AM	**VST01** ASP.NET Core 2 For Mere Mortals - *Philip Japikse*	**VST02** Busy Developer's Guide to Kotlin - *Ted Neward*	**VST03** Azure 101 - *Laurent Bugnion*	**VST04** Modernizing Your Source Control: Migrating to Git from Team Foundation Version Control (TFVC) - *Colin Dembovsky*
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	1:50 PM	**VST05** Fast Focus: Serverless Computing: Azure Functions and Xamarin in 20 minutes - *Laurent Bugnion*	**VST06** Fast Focus: Getting Git - *Jason Bock*	**VST07** Fast Focus: Xamarin.Essentials - Cross-Platform APIs for Your Mobile Apps - *James Montemagno*	
2:00 PM	2:20 PM	**VST08** Fast Focus: Getting Started with ASP.NET Core 2.0 Razor Pages - *Walt Ritscher*	**VST09** Fast Focus: Cross Platform Device Testing with xUnit - *Oren Novotny*	**VST10** Fast Focus: Get Your Full .NET Code into .NET Standard - *Rockford Lhotka*	
2:20 PM	2:45 PM	Networking Break • Visit the EXPO - *Pacifica 7*			
2:45 PM	4:00 PM	**VST11** Introduction to Webpack - *Chris Klug*	**VST12** Xamarin.Forms Takes You Places! - *Sam Basu*	**VST13** Azure, Windows and Xamarin: Using the Cloud to Power Your Cross-platform Applications - *Laurent Bugnion*	**VST14** Writing Testable Code and Resolving Dependencies – DI Kills Two Birds with One Stone - *Miguel Castro*
4:15 PM	5:30 PM	**VST15** Angular Application Testing Outside the Church of TDD - *Chris Klug*	**VST16** Busy .NET Developer's Guide to Python - *Ted Neward*	**VST17** HoloLens, Mixed Reality & VR Development with the Cloud - *Nick Landry*	**VST18** Testing in Production Using Azure and Visual Studio Team Services (VSTS) - *Colin Dembovsky*
5:30 PM	7:30 PM	Exhibitor Reception - *Pacifica 7*			
Visual Studio Live! Day 2: Wednesday, December 5, 2018					
8:00 AM	9:15 AM	**VSW01** The Whirlwind Tour of Authentication and Authorization with ASP.NET Core - *Chris Klug*	**VSW02** Essential Tools for Xamarin Developers! - *Sam Basu*	**VSW03** Introduction to Windows Containers and Docker - *Marcel de Vries*	**VSW04** OWASP DevSlop: DevSecOps with VSTS & Azure - *Tanya Janca*
9:30 AM	10:45 AM	**VSW05** Assembling the Web - A Tour of WebAssembly - *Jason Bock*	**VSW06** Cross-Platform App Dev with Xamarin and CSLA .NET - *Rockford Lhotka*	**VSW07** Microservices with AKS (Azure Kubernetes Service) - *Vishwas Lele*	**VSW08** To Be Announced
10:45 AM	11:30 AM	Networking Break • Visit the EXPO - *Pacifica 7*			
11:30 AM	12:30 PM	**LIVE! 360 KEYNOTE: Enterprise Transformation (And You Can Too)** *Donovan Brown, Principal DevOps Manager, Cloud Developer Advocacy Team, Microsoft*			
12:30 PM	1:30 PM	Birds-of-a-Feather Lunch			
1:30 PM	2:00 PM	Dessert Break • Visit the EXPO			
2:00 PM	3:15 PM	**VSW09** No Strings Attached: JavaScript without Webpack, Transpilers, or Frameworks - *Ashley Grant*	**VSW10** DevOps for the SQL Server Database - *Brian Randell*	**VSW11** Programming with Microsoft Flow - *Walt Ritscher*	**VSW12** What's New in C# 7 - *Phil Japikse*
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** Electron: Desktop Development For Web - *Chris Woodruff*	**VSW14** Entity Framework for Enterprise Applications - *Benjamin Day*	**VSW15** The Mystical World of I/O Bindings in Azure Functions - *Ashley Grant*	**VSW16** Signing Your Code the Easy Way - *Oren Novotny*
7:30 PM	9:00 PM	Live! 360 Dessert Luau - *Wantilan Pavilion*			
Visual Studio Live! Day 3: Thursday, December 6, 2018					
8:00 AM	9:15 AM	**VSH01** Encrypting the Web - *Robert Boedigheimer*	**VSH02** Learning The Language Of HTTP For A Better Data Experience In Your Mobile Apps - *Chris Woodruff*	**VSH03** How to Interview a Developer - *Billy Hollis*	**VSH04** Creating a Release Pipeline with Team Services - *Esteban Garcia*
9:30 AM	10:45 AM	**VSH05** Advanced Fiddler Techniques - *Robert Boedigheimer*	**VSH06** Making XAML Do Things You Didn't Realize It Could - *Billy Hollis*	**VSH07** What Developers Want - *Rabeb Othmani*	**VSH08** To Be Announced
11:00 AM	12:00 PM	**VISUAL STUDIO LIVE! PANEL DISCUSSION: What Matters Most for the future of Modern Apps: AI, Data, Security, or UX?** *Brian Randell (Moderater), Billy Hollis, Tanya Janca, Oren Novotny, & Rabeb Othmani*			
12:00 PM	1:00 PM	Lunch on the Lanai - *Lanai / Pacifica 7*			
1:00 PM	2:15 PM	**VSH09** Get Func-y: Understanding Delegates in .NET - *Jeremy Clark*	**VSH10** Non-Useless Unit Testing Entity Framework & ASP.NET MVC - *Benjamin Day*	**VSH11** Modernizing the Enterprise Desktop Application - *Oren Novotny*	**VSH12** DevOps: A Catalyst for Enterprise Agility - *Heidi Araya & Esteban Garcia*
2:30 PM	3:45 PM	**VSH13** IEnumerable, ISaveable, IDontGetIt: Understanding .NET Interfaces - *Jeremy Clark*	**VSH14** Finding Your Place in the Cosmos: When and Why You Should Consider Azure Cosmos DB - *Eric Potter*	**VSH15** Pushing Left Like a Boss: Application Security Foundation - *Tanya Janca*	**VSH16** Scrum, Kanban, or Scrumban? - *Heidi Araya*
4:00 PM	5:00 PM	Next? Live! 360 Networking Event *Heidi Araya, Andrew Brust, Jeremy Clark, Ben Curry, Benjamin Day, Peter De Tender, Brent Edwards, Kevin Ford, Grant Fritchey, Esteban Garcia, Seth Juarez, Dave Kawula, Cristal Kawula, Sami Laiho, Thomas LaRock, Vishwas Lele, Rockford Lhotka, Karen Lopez, Matthew McDermott, Brian Randell, Paul Schaeflein, Jen Underwood, Rob Windsor*			
Visual Studio Live! Post-Conference Workshops: Friday, December 7, 2018					
8:00 AM	05:00 PM	**VSF01** Workshop: UX Design for Developers: Basics of Principles and Process - *Billy Hollis*	**VSF02** Workshop: Microservices with AKS (Managed Kubernetes) - *Vishwas Lele*		

Speakers and sessions subject to change

Augmented Reality in Xamarin.Forms

Rajeev K R

Augmented reality (AR) is quickly emerging as an incredibly useful tool for solving everyday problems. AR is an interactive, reality-based display environment that blends virtual objects with real ones to create an immersive UX. Enabled by advanced hardware like Microsoft HoloLens, it employs virtual 2-D/3-D objects, sound, text, effects and tracking.

AR has found a ready home in mobile devices and smartphones, thanks to high-resolution cameras, fast processors and high-bandwidth wireless networks. Support for AR experiences in mobile OSes like iOS and Android lower the barrier for entry. Broad mobile support makes AR an attractive target for Xamarin.Forms developers, who can leverage the code-sharing benefits of the framework and mature tooling with all the compelling benefits of AR.

Enterprises are already using AR to engage customers and improve experiences. Harley-Davidson has created an iPad app that provides a virtual shopping experience that allows customers to view body types, seats, lights and add other options for their custom bike designs. Hyundai's Virtual Guide app uses AR to teach owners how its car works, while IKEA has developed an app that lets shoppers see how furniture might look in their living spaces.

AR has found a ready home in mobile devices and smartphones, thanks to high-resolution cameras, fast processors and high-bandwidth wireless networks.

Currently, AR is a device-specific feature that requires a native platform to execute. iOS exposes AR to developers via ARKit, while Android does so via ARCore. ARKit requires an iOS device with iOS 11 or newer and at least an A9 processor. ARCore requires Android 7.0 or newer and access to the Google Play Store. You can refer to the official documentation from Google (bit.ly/2BY68oS) and Apple (apple.co/2PcSVdm) to learn more about the hardware and software requirements.

This article discusses:

- Augmented reality concepts and interaction design guidelines for AR
- ARKit in Xamarin.Forms application
- Using Dependency Services for accessing device-specific features

Technologies discussed:

iOS, Android, Xamarin Platform, Xamarin.Forms, ARKit, ARCore

Code download available at:

msdn.com/magazine/1018magcode

While learning a new technology, it's important to understand the fundamental concepts behind it. Let's consider these concepts for AR.

Tracking locates your device position in the real world in real time. To create a real-time relationship between physical and virtual spaces, ARKit uses a technique called visual-inertial odometry (VOI), while ARCore uses Concurrent Odometry and Mapping (COM).

> iOS exposes AR to developers via ARKit, while Android does so via ARCore.

Environmental Understanding is the process of detecting feature points and planes in the real world. ARCore and ARKit are capable of determining each plane's boundary. This can be used for placing a virtual object inside a real-world plane boundary.

Anchor refers to position and orientation in physical space. We need to find an actual position in the real world for placing a scene. ARKit and ARCore are both able to maintain this position when the camera is moving around.

Light Estimate determines the amount of light in the physical environment and applies the correct amount of lighting to virtual objects embedded within it to produce a more realistic effect. ARKit and ARCore use the camera sensors to estimate the light.

Interactive Environment enables ARCore and ARKit to map the physical environment to the device screen, employing hit testing to determine X, Y coordinates on screen. This provides users the ability to interact with the environment through the screen interface, using, for example, gestures like tap and swipe.

Enter Xamarin.Forms

Xamarin is one of the most widely used cross-platform tools for enterprises developing mobile applications. With the introduction of Xamarin.Forms, enterprises could build native UIs for iOS and Android using a single, shared C# code base—overcoming a key obstacle to adoption and making it a compelling platform for developing enterprise applications. ARKit and ARCore are fully supported by Xamarin, and the Xamarin Web site provides in-depth documentation for both ARKit and ARCore.

Figure 1 **Running the Application on iOS**

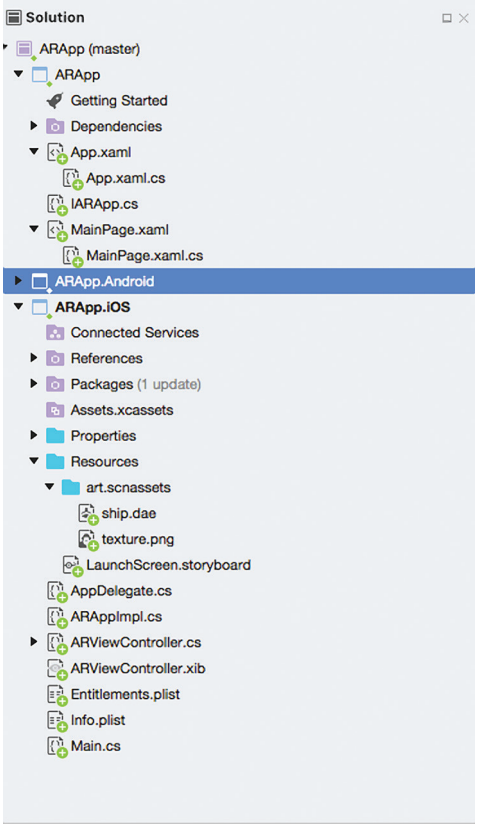


Figure 2 **Project Structure**

From my experience in mobile application development, it's possible to achieve a considerable reduction in both lines of code and number of bugs by paying more attention in the design stage of Xamarin.Forms applications. A proper application architecture design in Xamarin.Forms enables maximum code reusability across platforms. It also enables faster development, ease of automation testing, integration of changes, reduction of bugs and more. In the Agile world, in order for an application to succeed, it should enable easy addition of functionality with minimal code changes, without introducing new bugs. This is why Xamarin.Forms is a preferred approach for mobile development, especially when executed with Agile methodology.

As a Xamarin developer, it's interesting to see how the power of C# and the advantages of Xamarin.Forms can improve development of exciting AR-based applications.

Consider AR as a UI layer in Xamarin that handles all the user interactions. Because AR is a feature that's device-specific and a hardware-sensitive, there may often be situations where the AR-native platform functionalities aren't implemented in the Forms API. This is where the real power of Xamarin.Forms comes in handy.

In order to develop Xamarin applications, Visual Studio must be installed on the machine. I can code Xamarin on a Windows machine with Visual Studio, but I need a Mac for running and debugging iOS applications. Install the latest iOS and Android SDKs in development machines.

While Xamarin Live Player can be used to test Xamarin.Forms applications, it's not a professional-grade tool and won't support

LIVE! 360
TECH EVENTS WITH PERSPECTIVE

TECHMENTOR

IN-DEPTH TRAINING FOR IT PROS

Training Conference for IT Pros

ROYAL PACIFIC RESORT AT UNIVERSAL ORLANDO
December 2-7, 2018

Where IT Training Meets the Sunshine

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

TRACK TOPICS INCLUDE:

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

REGISTER NOW

SAVE $400 WITH SUPER EARLY BIRD PRICING!
REGISTER BY OCTOBER 5

Use Promo Code MSDN

A Part of Live! 360: The Ultimate Education Destination

6 CONFERENCES, 1 GREAT PRICE

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

TECHMENTOREVENTS.COM/ORLANDO

EVENT PARTNERS
Microsoft
Magenic

SILVER SPONSOR
Progress
GOVERLAN REACH

PREMIER & ALLIANCE MEDIA PARTNERS
netda
UserGroup.TV

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

PRODUCED BY
CONVERGE 360
An 1105 MEDIA company

AGENDA-AT-A-GLANCE

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

START TIME	END TIME	
TechMentor Full Day Hands-On Labs: Sunday, December 2, 2018		
7:30 AM	9:00 AM	Registration • Coffee and Morning Pastries
9:00 AM	6:00 PM	**TMS01** Hands-On Lab: Advanced Troubleshooting for IT/Ops Pros - *Bruce Mackenzie-Low* | **TMS02** Hands-On Lab: Windows PowerShell Jump Start HOL - *Jeffery Hicks*
2:00 PM	7:00 PM	Pre-Conference Registration - Royal Pacific Resort Conference Center
TechMentor Pre-Conference Workshops: Monday, December 3, 2018		
7:00 AM	8:30 AM	Registration • Coffee and Morning Pastries
8:30 AM	5:30 PM	**TMM01** Workshop: Mastering the Sysinternals Toolkit - *Sami Laiho* | **TMM02** Workshop: How to Successfully Manage a Fabric Using System Center 2016/2019? - *Mikael Nystrom*
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
TechMentor Day 1: Tuesday, December 4, 2018		
7:00 AM	8:00 AM	Registration • Coffee and Morning Pastries
8:00 AM	9:00 AM	**TECHMENTOR KEYNOTE: OneDrive and M365 - Enabling Secure and Seamless Collaboration for a Untethered Workforce** *Stephen Rose, Sr. Product Manager, OneDrive for Business, Microsoft & Jason Moore, Head of OneDrive, Partner Group Program Manager, Microsoft*
9:15 AM	10:30 AM	**TMT01** Everything You're Doing Wrong in PowerShell - *Don Jones* | **TMT02** Shields Up! Managing the Windows Firewall with Advanced Security - *Richard Hicks* | **TMT03** Image Factory: Automation Grand Deluxe - *Mikael Nystrom*
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	1:50 PM	**TMT04** Fast Focus: Microsoft and Open Source - *Jessica Deen* | **TMT05** Fast Focus: 5 Things Everyone Needs to Know About Azure - *Peter De Tender*
2:00 PM	2:20 PM	**TMT06** Fast Focus: Office 365 and Azure AD Security Quick Wins - *Holly Lockhart & Oana Enache* | **TMT07** Fast Focus: AutoPilot Notes from the Field - *Ami Casto*
2:20 PM	2:45 PM	Networking Break • Visit the EXPO - *Pacifica 7*
2:45 PM	4:00 PM	**TMT08** How I Would Audit Your Windows Security - *Sami Laiho* | **TMT09** DevOps with Azure, Kubernetes, and Helm - *Jessica Deen* | **TMT10** Consolidating DataCenters with Windows Failover Clusters - *Bruce Mackenzie-Low*
4:15 PM	5:30 PM	**TMT11** Securing Access to Office 365, SaaS and On-Premises Applications - *Oana Enache & Holly Lockhart* | **TMT12** Using Desired State Configuration in Azure - *Will Anderson* | **TMT13** Exploring Storage Replica in Windows Server 2019 - *Mikael Nystrom*
5:30 PM	7:30 PM	Exhibitor Reception - *Pacifica 7*
TechMentor Day 2: Wednesday, December 5, 2018		
7:30 AM	8:00 AM	Registration • Coffee and Morning Pastries
8:00 AM	9:15 AM	**TMW01** DevOps Making Snowflakes into Cattle - *Don Jones* | **TMW02** No More Passwords! An Introduction to Windows Hello for Business - *Richard Hicks* | **TMW03** Windows 10 Servicing with Traditional and Modern Options - *Johan Arwidmark & Ami Casto*
9:30 AM	10:45 AM	**TMW04** PowerShell: Under the Hood - *Don Jones* | **TMW05** Techie Got Talent - Run Your Own Podcast / Webcast - *Harjit Dhaliwal & Nick Brattoli* | **TMW06** How to Setup and Maintain Config Manager in your Environment - *Johan Arwidmark & Ami Casto*
10:45 AM	11:30 AM	Networking Break • Visit the EXPO - *Pacifica 7*
11:30 AM	12:30 PM	**LIVE! 360 KEYNOTE: Enterprise Transformation (And You Can Too)** *Donovan Brown, Principal DevOps Manager, Cloud Developer Advocacy Team, Microsoft*
12:30 PM	1:30 PM	Birds-of-a-Feather Lunch
1:30 PM	2:00 PM	Dessert Break • Visit the EXPO
2:00 PM	3:15 PM	**TMW07** Conceptualizing Desired State Configuration Using Your Own Scripts - *Will Anderson* | **TMW08** Securely Scripting PowerShell - *Jeffery Hicks* | **TMW09** How to Use PowerShell to Become a Windows Management SuperHero - *Petri Paavola*
3:15 PM	4:00 PM	Networking Break • Visit the EXPO • Expo Raffle @ 3:30 p.m. - *Pacifica 7*
4:00 PM	5:15 PM	**TMW10** Everything You Need to Know About Microsoft 365 - *Harjit Dhaliwal & Nick Brattoli* | **TMW11** The Zen of PowerShell Scripting - *Jeffery Hicks* | **TMW12** Co-Management Status Quo - *Panu Saukko*
7:30 PM	9:00 PM	Live! 360 Dessert Luau - *Wantilan Pavilion*
TechMentor Day 3: Thursday, December 6, 2018		
7:30 AM	8:00 AM	Registration • Coffee and Morning Pastries
8:00 AM	9:15 AM	**TMH01** The PowerShell Core Tutorial - *Don Jones* | **TMH02** What's New in Windows Server 2019 - *Dave Kawula* | **TMH03** Securing Service Accounts the Modern Way - *John O'Neil Sr.*
9:30 AM	10:45 AM	**TMH04** Start Using JEA Today to Stop Overprivileged User Accounts - *John O'Neil Sr.* | **TMH05** Deploying Application Whitelisting on Windows Pro or Enterprise - *Sami Laiho* | **TMH06** Automated Troubleshooting Techniques in Enterprise Domains - *Petri Paavola*
11:00 AM	12:00 PM	**TECHMENTOR PANEL DISCUSSION: The Future of IT** *Sami Laiho and Dave Kawula (Moderators); Will Anderson, Johan Arwidmark, Ami Casto, and Don Jones*
12:00 PM	1:00 PM	Lunch on the Lanai - *Lanai / Pacifica 7*
1:00 PM	2:15 PM	**TMH07** Azure Security Unchained - *Peter De Tender* | **TMH08** The Case of the Shrinking Data - Deduplication in Windows Server 2019 on ReFS - *Dave Kawula* | **TMH09** Top 10 Configuration Manager Current Branch Mistakes (and How to Avoid Them) - *Panu Saukko*
2:30 PM	3:45 PM	**TMH10** Azure Stack, What You Need to Know - *Peter De Tender* | **TMH11** The Weakest Link of Office 365 Security - *Nestori Syynimaa* | **TMH12** How to Become a Community Rockstar - Learn How to Showcase Your Skills - *Cristal Kawula*
4:00 PM	5:00 PM	Next? Live! 360 Networking Event *Heidi Araya, Andrew Brust, Jeremy Clark, Ben Curry, Benjamin Day, Peter De Tender, Brent Edwards, Kevin Ford, Grant Fritchey, Esteban Garcia, Seth Juarez, Dave Kawula, Cristal Kawula, Sami Laiho, Thomas LaRock, Vishwas Lele, Rockford Lhotka, Karen Lopez, Matthew McDermott, Brian Randell, Paul Schaeflein, Jen Underwood, Rob Windsor*
TechMentor Post-Conference Workshops: Friday, December 7, 2018		
7:30 AM	8:00 AM	Registration • Coffee and Morning Pastries
8:00 AM	05:00 PM	**TMF01** Workshop: Notes from the Field on Storage Spaces Direct & Managing Windows Server with Project Honolulu (Windows Admin Center) - *Dave Kawula* | **TMF02** Workshop: Hacking and Hardening Office 365 and Azure AD - *Nestori Syynimaa*

Speakers and sessions subject to change

custom renderers, effects or third-party ModelView-View-Model (MVVM) products. Microsoft recommends using emulators built into Visual Studio for testing Xamarin applications. Many developers also use Genymotion. For testing, it's recommended to install iOS and Android AR-based applications on physical, mobile devices.

Create Your Xamarin.Forms AR Project

In this sample, I'll show you how to place a 3-D aircraft image in a physical space using Xamarin.Forms and ARKit. You'll also learn how to animate this 3-D object. **Figure 1** shows the final AR app running on iOS.

I'll start by creating a Xamarin.Forms application with the name ARApp. As shown in **Figure 2**, a default Xamarin.Forms project named ARApp is generated for .NET Standard, along with the platform-specific projects ARApp.iOS and ARApp.Android. Next, I create a folder named art.scnassets and add 3-D files and texture files to the ARApp.iOS project. These assets will be for loading the AR scene.

Now I'll create the UI for initiating the AR functionality. For demo purposes, I'll define a button in the MainPage.xaml file, like so:

```
<Button Text="Click Me" VerticalOptions="Center"
HorizontalOptions="Center"
  Clicked="OnButtonClick"></Button>
```

Figure 3 **Implementing a UI on iOS**

```
public override void ViewDidLoad ()
{
  base.ViewDidLoad();
  startAR();
}
// Initialize AR scene view
public void startAR()
{
  // Create the scene view for displaying the 3-D scene
  sceneView = new ARSCNView();
  sceneView.Frame = View.Frame;
  View = sceneView;
  CreateARScene(sceneView);
  PositionScene(sceneView);
}
// Configure AR scene with 3-D object
public void CreateARScene(ARSCNView sceneView)
{
  // lLoading the 3-D asset from file
  var scene = SCNScene.FromFile("art.scnassets/ship");
  // Attaching the 3-D object to the scene
  sceneView.Scene = scene;
  // This is for debugging purposes
  sceneView.DebugOptions = ARSCNDebugOptions.ShowWorldOrigin  |
ARSCNDebugOptions.ShowFeaturePoints;
}
// Position AR scene
public void PositionScene(ARSCNView sceneView)
{
  // ARWorldTrackingConfiguration uses the back-facing camera,
  // tracks a device's orientation and position, and detects
  // real-world surfaces, and known images or objects
  var arConfiguration = new ARWorldTrackingConfiguration
  {
    PlaneDetection = ARPlaneDetection.Horizontal,
    LightEstimationEnabled = true
  } ;
  // Run the AR session
  sceneView.Session.Run(arConfiguration, ARSessionRunOptions.ResetTracking);

  var sceneNode = sceneView.Scene.RootNode.FindChildNode("ship", true);
  sceneNode.Position = new SCNVector3(0.0f, 0.0f, -30f);
    sceneView.Scene.RootNode.AddChildNode(sceneNode);
  // Add some animation
  sceneNode.RunAction(SCNAction.RepeatActionForever(
    SCNAction.RotateBy(0f, 6f, 0, 5)));
}
```

Next, I'll implement the button click in codebehind, with this code:

```
public partial class MainPage : ContentPage
{
  public MainPage()
  {
    InitializeComponent();
  }
  private void OnButtonClick(object sender, EventArgs e)
  {
    // Handle the button click from XAML UI here
  }
}
```

I now have the Xamarin.Forms UI ready. Now I'll move to the native AR layer implementation.

Platform-Specific AR Implementation

It's time to implement AR using platform-specific APIs for either ARKit or ARCore, depending on the platform. I'll start by creating a platform-specific UI with the AR UI implementation, which can be invoked using DependencyService, the Xamarin implementation of dependency injection (DI) that allows apps to call into platform-specific functionality from shared code.

I'll get started with an implementation for UI on iOS. First, I create a viewController in the iOS project and add the code shown in **Figure 3**.

I'm now set to invoke AR from Xamarin.Forms. But how? This is where the real power of Xamarin.Forms comes in. DI is used to call native platform code in .NET Standard or, optionally, a Portable Class Library (PCL). Let's see how this is implemented.

Dependency Services

The DependencyService functionality allows apps to call into platform-specific functionality from shared code. This functionality enables Xamarin.Forms apps to do anything that a native app can do. There are four components needed to use DependencyService:

- **Interface:** Define the functionality as an interface in shared code.
- **Implementation Per Platform:** Add classes implementing the interface to each platform project.
- **Registration:** Register each implementing class with DependencyService via a metadata attribute. Registration enables DependencyService to find the implementing class and supply it in place of the interface at run time.
- **Call to DependencyService:** Request implementations of the interface by explicitly calling DependencyService from code.

I'll use Dependency Services in a Xamarin.Form application to implement AR.

Interface An interface to define the interaction with ARKit/ARCore platform-specific APIs must be designed. It's important to give more attention to this layer in the design, as it's the API layer exposed to external classes. For example, you can create an interface named IARApp and define a method named LaunchAR:

```
public interface IARApp{
  void LaunchAR (  ); // Note that interface members are public by default
}
```

Platform-Specific Implementation for AR Interface The interface must be implemented in the project for each platform that you target. As the implementation happens in a shared environment, the interface may be called from all of the platforms. Any platform lacking the implementation will generate a NullReferenceException.

Figure 4 **Implementation for IARApp Interface on iOS**

```
[assembly: Xamarin.Forms.Dependency(typeof(ARDemo.iOS.ARAppImpl))]
namespace ARDemo.iOS
{
  public class ARAppImpl:IARApp
  {
    public void LaunchAR()
    {
      // This is in native code; invoke the native UI
        ARViewController viewController = new ARViewController();
        UIApplication.SharedApplication.KeyWindow.RootViewController.
          PresentViewController(viewController, true, null);
    }
  }
}
```

You can see the code for implementation for IARApp interface on iOS in **Figure 4**. The code for implementing the same interface on Android is here:

```
[assembly: Xamarin.Forms.Dependency(typeof(ARDemo.Droid.ARAppImpl))]
namespace ARDemo.Droid
{
  public class ARAppImpl:IARApp
  {
    public void LaunchAR()
    {
      // Launch AR in Android
    }
  }
}
```

Note that the [assembly:] attribute must be declared above the namespace. Otherwise, the dependency service won't get invoked. With all this done, I can now invoke AR from a Xamarin.Forms button click, calling the Dependency Service with this code:

```
public partial class MainPage : ContentPage
{
  public MainPage()
  {
    InitializeComponent();
  }
  private void OnButtonClick(object sender, EventArgs e)
  {
    DependencyService.Get<IARApp>().LaunchAR(); // Launch AR
  }
}
```

Wrapping Up

Figure 5 offers a look at the application architecture for the sample app. The AR implementation for iOS is complete, and you can complete the Android implementation using the same approach.

> For designing an engaging immersive experience with AR in mobile, it's important to prioritize the user interaction design.

Here's a tip for those who intend to implement this for Android: Use ARCore through dependency services for implementing AR in Android. HelloAR is a Google AR project implemented for Android platforms that's been ported to Xamarin. This project will help you start the AR implementation in Android. Please refer to the following links for more information: bit.ly/2ojBZq4 and bit.ly/2PKcomX.

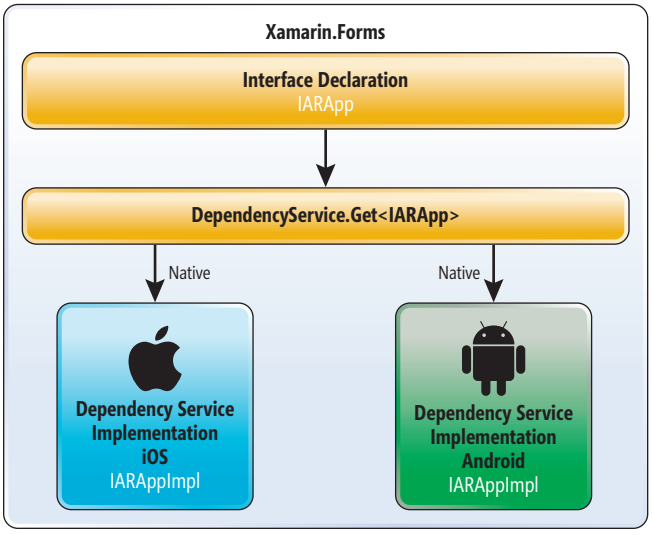


Figure 5 **The Application Architecture**

One last thing: For designing an engaging immersive experience with AR in mobile, it's important to prioritize the user interaction design. ARKit and ARCore documentation provides useful tips and guidelines for creating optimal immersive experiences for end users. Let's go through some important interaction design guidelines for AR:

- Interactions should be intuitive and simple. Avoid needless complexity.
- Use audio and haptic feedback to enhance the immersive experience.
- Use the entire screen space (display) to engage the user.
- Scenes, objects and animations should be convincing to the user.
- Consider how people must hold their device when using your app and make sure that it provides an optimal immersive experience for them.
- Provide intuitive context-based guidance for the user whenever needed.

AR is a game changer. It's the main ingredient of new technologies around digital twinning, which leverages digital replicas of physical assets, processes and systems to benefit a wide range of activities. The growth of this technology in the enterprise will happen very quickly, and Xamarin.Forms stands to be an important enabler of digital twinning going forward. Be sure to explore the magic of AR in different platforms and get yourself equipped.

Happy coding! ■

Rajeev K R *is a solution architect at TCS interactive in Tata Consultancy Services. He is mainly concentrating on deriving end-to-end enterprise digital channel strategy and building enterprise architecture for mobile solutions. He's passionate about the latest digital technologies and interested in experiments by mixing immersive technologies in mobile with AI/ML and conversational UI platforms.*

Thanks to the following Microsoft technical expert for reviewing this article: *Dan Hermes*

DINO ESPOSITO

Blazor at Work: Events, Binding and Composition

I confess I've never been a huge fan of Angular and React, two extremely popular frameworks for writing modern Web front ends. I don't want to spark any heated debates, but I think that it's safe to state that it's all about how much work the application does on the server and how much it does on the client. Clearly, any solution centered on the client side of the Web has the potential to be more responsive and, maybe more importantly, much less impactful on the server. On the other hand, there's the learning curve and extra programming work that has to be done with JavaScript or, at best, TypeScript.

Blazor is an interesting and warmly welcomed attempt to enable developers to build client-centric applications using, for the most part, C# and the .NET runtime. As Blazor evolves, more and more features will be added to make it provide nearly the same set of capabilities as Angular, React or Vue. In the end, writing a Blazor application will be a matter of stitching together HTML and CSS using C# code, the Razor template language, and JSON data downloaded from some HTTP endpoints.

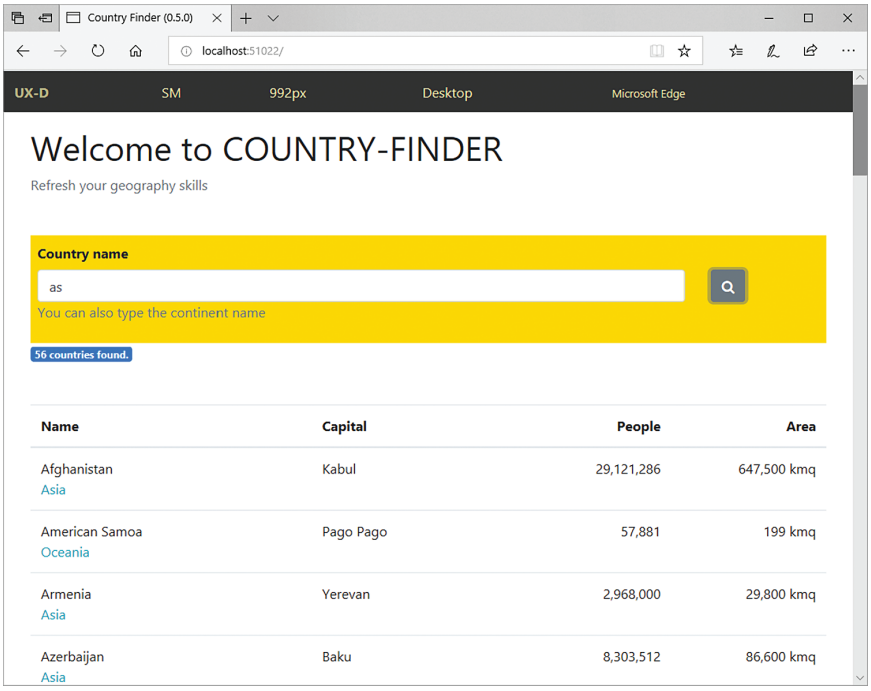


Figure 1 **The Sample Application in Action**

The killer feature of Blazor is that an object-oriented language like C# is much easier to work with than JavaScript when expressing even moderately sophisticated business logic. In addition, Blazor offers a client-centric programming environment that's much more familiar than, say, Angular is to the community of ASP.NET developers.

It will take time for Blazor to mature and compete with established frameworks, but the dream of C# code running in the browser via an open technology (WebAssembly) rather than a proprietary plug-in is alive.

In a previous column (msdn.com/magazine/mt829756), I presented the highlights of the Blazor architecture and presented a basic example based on a digital clock and a timer to update it. In this column, I move toward the type of core tasks you'd expect to perform in a client-centric application. This demo will use UI components, parameters, events, JavaScript interoperability and data binding. **Figure 1** offers a glimpse of the final result: a form that uses auto-completion to accept a string, runs a query on the back end, gets JSON data and refreshes the UI.

Code download available at bit.ly/2v0IYYZ.

Creating the Project

The sample project is built as an ASP.NET Core-hosted application. The solution is made of three projects—the client Blazor project, the Web API server for remote operations and an optional shared library linked to both projects. This shared library is the ideal location for all data-transfer objects and view models used on both the client and the server. The default configuration of the solution in Visual Studio sets the ASP.NET Core project as the runner.

Let's take a look at the content of the startup class. In addition to the canonical stuff to set up the MVC router, error handling, logging, compression and whatever else you may need, it has ad hoc middleware to use Blazor:

```
public void Configure(IApplicationBuilder app)
{
  app.UseResponseCompression();
    ...
  app.UseBlazor<Client.BlazorProgram>();
}
```

Manipulating Documents?

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

Try GroupDocs APIs for FREE

Download a Free Trial at

https://downloads.groupdocs.com

GROUPDOCS

GroupDocs.Viewer

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

GroupDocs.Annotation

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

GroupDocs.Conversion

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

GroupDocs.Comparison

Compare two documents and get a difference summary report.

GroupDocs.Signature

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

GroupDocs.Assembly

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

GroupDocs.Metadata

Organize documents with metadata within any cross platform application.

GroupDocs.Search

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

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

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

The type passed as an argument to the middleware is the type of the program entry point in the Blazor application. Needless to say, the ASP.NET Core server application (acting as a plain Web API) must have a reference to the Blazor project. Though it's not strictly a requirement, it is recommended that the application uses compression to reduce the amount of traffic over the network. The compression service is configured in the ConfigureServices method of the startup class as:

```
services.AddResponseCompression(options =>
{
  options.MimeTypes =
    ResponseCompressionDefaults.MimeTypes.
Concat(new[]
    {
      MediaTypeNames.Application.Octet,
      WasmMediaTypeNames.Application.Wasm,
    });
});
```

You don't need to have anything else in the Web API project other than the layers of code necessary to retrieve and serve JSON data to the client Blazor front end. The Web API used in this article returns information about countries of the world. The list of countries is embedded in the project as an internal JSON resource, and a repository class takes care of caching and filtering its content (see **Figure 2**). The Shared library defines the Country class and its ancillary types. As mentioned, these types will be shared between the Web API and the Blazor application.

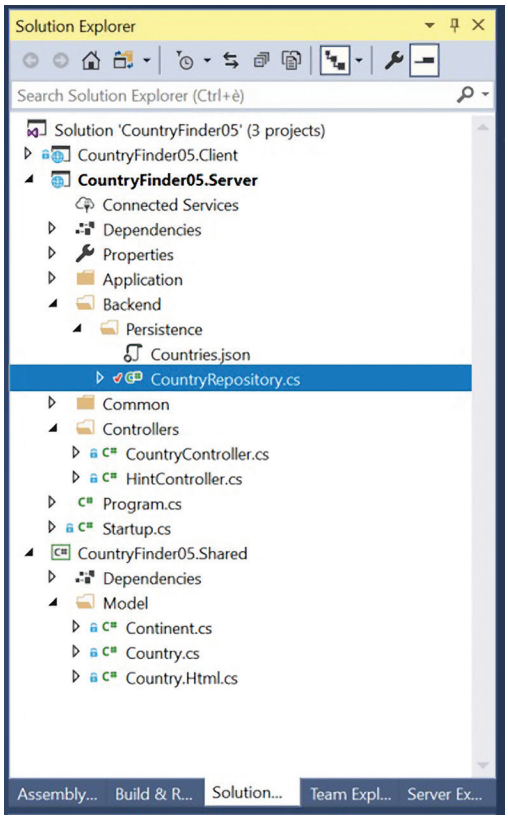


Figure 2 **The Sample Project in Visual Studio**

UI Components of a Blazor Application

Realistically, any Blazor application consists of a bunch of client-side navigable Web pages, each of which is written in Razor. Ultimately, a sample .cshtml page consists of a bit of HTML, some Razor extensions, and custom components similar to the custom tag helpers of ASP.NET Core. UI components are distinct segments of Razor code that at first may resemble the partial views of ASP.NET Core applications. At a deeper look, though, it turns out that components are pretty different from partial views. Components take parameters, are stateful, have a specific lifecycle, can define events and have a different compilation model.

Let's say you have a CountrySelector.cshtml file somewhere in your project. A common location is the Views/Shared folder of the Blazor application, but the folder isn't really important. Aside from some possible CSS style and HTML layout information, the component will contain an input field and a button, as shown here:

```
<input type="text"
  bind="@SelectedCountry"
  placeholder="@Placeholder" />
<button type="button" onclick="@GetCountryInfo">
  <i class="fa fa-search"></i>
</button>
```

When the button is clicked, the GetCountryInfo method runs. The value displayed in the textbox is associated with a component-level field named SelectedCountry. Here's the minimal code you need to have in the component to handle the click and query the Web API:

```
@functions
{
  private string SelectedCountry;
  async void GetCountryInfo()
  {
    var url = WebUtility.UrlEncode($"/
country/search/{SelectedCountry}");
    var countries = await
      Http.GetJsonAsync<IList<Country>>(url);
  }
}
```

The Bind attribute used on the INPUT element ensures two-way data binding between the value attribute of the DOM element and the SelectedCountry field. The binding is automatic on the onchange event.

There are a few interesting things to remark on about UI components. For example, a UI component can expose parameters that the parent sets. Here's how the sample page of **Figure 1** references the CountrySelector UI component:

```
<CountrySelector
  uniqueId="cs"
  placeholder="Start typing the country name"
  typeAheadUrl="/hint/countries"
  onSelectionMade="@PopulateGrid" />
```

Note that the case of attributes is a matter of preference. You can use the client-side common camelCase style (as shown in this snippet) or go with PascalCase style to make it clear that those attributes fall in the .NET land.

All attributes listed are defined as parameters in the @functions{…} block of the UI component, like so:

```
[Parameter]
string UniqueId { get; set; } = "country-selector";
[Parameter]
string TypeAheadUrl { get; set; } = "";
[Parameter]
string Placeholder { get; set; } = "Country";
[Parameter]
Action<IList<Country>> OnSelectionMade { get; set; }
```

Most of the parameters set in the sample CountrySelector are used to customize the HTML being output. For example, the Placeholder attribute sets the helper text in an empty textbox, and the TypeAheadUrl attribute sets the URL that ultimately provides autocompletion. (More on this in a moment.)

Firing and Handling Events

The OnSelectionMade parameter is ultimately an event handler, set by the parent component to specify the code to execute in reaction to an event fired by the CountrySelector component. In its simplest form, an event is defined as an Action<T> delegate. In the code snippet I just showed you, the OnSelectionMade event provides the handler with a list of countries to process:

```
[Parameter]
Action<IList<Country>> OnSelectionMade { get; set; }
```

Let's find out how the event is fired from within a component. The sample CountrySelector component has two main purposes:

Spreadsheets Everywhere.

SpreadsheetGear 2017 Released

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

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

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

Scalable Reporting

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

WIN Windows Forms | Silverlight | WPF

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

capturing a string of text to be used as the argument of a query, and running the query against a remote HTTP endpoint. The list of countries received from the query is exposed to whom it may concern through the OnSelectionMade custom event, like so:

```
async void GetCountryInfo()
  {
    var url = $"/country/search/{SelectedCountry}";
    var countries = await
      Http.GetJsonAsync<IList<Country>>(url);
    OnSelectionMade?.Invoke(countries);
  }
```

If not null, the Action<T> delegate is invoked with the list of countries. In general, a good practice is wrapping the event invocation code in a general-purpose and parameter-less method like NotifyStateChanged. In this case, it might be necessary that the state to be notified is captured in a centralized data structure.

Let's focus now on how an event can be handled from a parent component, such as the host page. The main page rendered in **Figure 1** contains the following @functions block:

```
@functions
{
  public string Message = "";
  IList<Country> Countries = new List<Country>();

  private void PopulateGrid(IList<Country> countries)
  {
    Message = $"{countries.Count} countries found.";
    Countries = countries;
    StateHasChanged();
  }
}
```

The method PopulateGrid is invoked when the event OnSelectionMade is fired. The method receives the list of selected countries and saves it internally to the Countries field. In addition, the total number of countries is saved to the Message field. Both fields, Countries and Message, are bound to a Blazor component (CountryGrid) and an HTML SPAN element, like so:

```
<CountrySelector onSelectionMade="@PopulateGrid" />
<span class="badge badge-primary">@Message</span>
<CountryGrid dataSet="@Countries" />
```

The content of the CountryGrid Blazor component is a plain HTML table-based template dynamically populated around the content of its DataSet parameter:

```
@functions
{
  [Parameter]
  IList<Country> DataSet { get; set; } = new List<Country>();
}
```

Having the event handler set the Countries field to a new value doesn't automatically refresh any components in the current view. If state updates occur because of an explicit user action, the UI is refreshed automatically. Otherwise, to refresh the view, an explicit, programmatic call to the StateHasChanged method is required:

```
void PopulateGrid(IList<Country> countries)
{
  Countries = countries;
  StateHasChanged();
}
```

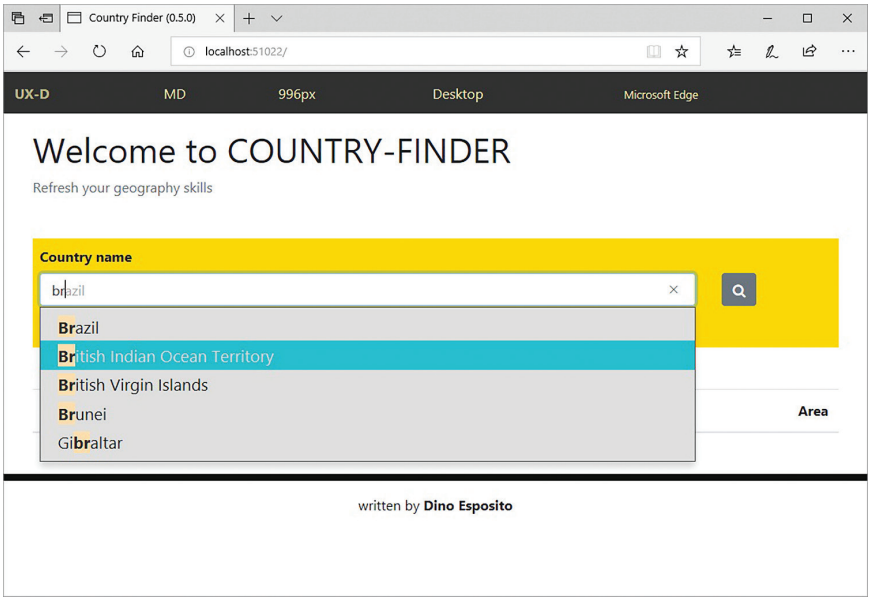


Figure 3 **Typeahead in Action**

Foundation of Data-Bound Components

Any client-centric Web framework spends a large share of its uptime doing data binding, and most of the time bindable data comes as JSON strings from some remote HTTP endpoint. Fetching data, whether customer details, flights or weather forecasts, is a fairly common scenario. To download data, you need an HTTP client referenced in the component. The easiest way to do that is using dependency injection, like this:

```
@inject HttpClient Http
```

The symbol Http references a singleton instance of HttpClient that can be used to get a JSON feed and serialize it to C# collections and classes. Here's the code you use to get bindable JSON data from a remote URL in Blazor:

```
var countries = await Http.GetJsonAsync<IList<Country>>(url);
```

Once you have the data, you need to find a way to bind it to visual elements. At present, in Blazor you don't reference UI components (or HTML elements) by name and code against their programming interface. Instead, you expose a public event, receive data through it and use component parameters to pass data to child components. In the previous example, the CountryGrid component has no ID property assigned and no instance of it is programmatically referenced to receive the data to bind. Instead, a parent field is assigned (Countries) that was data-bound to the DataSet property of the CountryGrid component.

JavaScript Interoperability

Blazor allows you to use C# code to implement client-side operations. However, the UI (both layout and style) is still expressed in HTML and CSS with some JavaScript that may be required here and there. Blazor doesn't mandate JavaScript interoperability, but realistically, some form of interoperation may be necessary at times. One scenario is to integrate in a Blazor UI some existing, and fairly sophisticated, pieces of JavaScript code.

As an example, let's consider the popular typeahead.js library

for auto-completion. The library comes as a jQuery plug-in that you programmatically connect to an input field. Upon loading, the plug-in must be invoked to react to the input event fired by the DOM as the user types in the field. How would you call JavaScript from within a Blazor view? As of Blazor 0.5.0, you need to wrap up any JavaScript code you wish to invoke from .NET in a top-level function registered on the browser's window object, like this:

```
window.MyFunctions = {
  typeAhead = function() {
    $(document).ready(function() {      // Typeahead initialization code
    });
  },
  uxDebug = function() {
    $(document).ready(function() { ... });
  }
}
```

> Blazor allows you to use C# code to implement client-side operations.

You place similar JavaScript code in a file referenced in the main index.html file, right after the Blazor bootstrap SCRIPT tag. The browser's window object is only decorated with custom methods, but still none of them are called. To enable typeahead.js, you invoke the JavaScript function from within the OnInit method of the Blazor component. Hence, the CountrySelector component will contain the following:

```
protected override void OnInit()
{
  JSRuntime.Current.InvokeAsync<Task>(
    "MyFunctions.typeAhead", "#input-field",
    TypeAheadUrl);
}
```

As a result, the JavaScript wrapper for typeAhead is invoked as soon as the Blazor view is rendered and, because of the implementation, its effect will be bound to the DOM's ready event. **Figure 3** shows what the typeAhead library looks like implemented in an app UI.

The black ribbon visible at the top of the Web page in **Figure 3** is a relatively simple piece of JavaScript that uses the services of the WURFL.JS endpoint (wurfl.io) to collect reliable browser information. Here, it shows the current Bootstrap size, actual width and some browser information.

Wrapping Up

Blazor promotes a component-based UI with data binding being the primary tool to glue together data and layout elements. HTML and CSS are the underlying languages to express visuals. JavaScript is fully supported, but in general, complex UI elements are better implemented using framework-specific native components. ■

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

Thanks to the following Microsoft technical expert for reviewing this article: *Daniel Roth*

LIVE! 360

TECH EVENTS WITH PERSPECTIVE

SQL Server LIVE!

TRAINING FOR DBAs AND IT PROS

ROYAL PACIFIC RESORT AT UNIVERSAL ORLANDO
December 2-7, 2018

Data. Driven.

With timely, relevant content delivered by recognized experts, who, like you, are driven by data, SQL Server Live! is back to help you do more with your SQL Server investment. With 6 days of workshops, deep dives and breakout sessions, SQL Server Live! provides a broad range of sessions on everything from performance tuning to security, to reporting and data integration. ***This training event is for administrators, DBAs, developers and BI pros to improve old approaches, adopt new techniques, and to modernize the SQL Server infrastructure.***

TRACK TOPICS INCLUDE:

- BI, Big Data, Data Analytics, and Data Visualization
- SQL Server Administration & Maintenance
- SQL Server in the Cloud
- SQL Server for Developers
- SQL Server Performance Tuning and Optimization

REGISTER NOW

SAVE $400 WITH SUPER EARLY BIRD PRICING!
REGISTER BY OCTOBER 5

Use Promo Code MSDN

A Part of Live! 360: The Ultimate Education Destination

6 CONFERENCES, 1 GREAT PRICE

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

SQLLIVE360.COM

EVENT PARTNERS: Microsoft, Magenic

SILVER SPONSOR: Progress, GOVERLAN REACH

PREMIER & ALLIANCE MEDIA PARTNERS: netda, UserGroup.TV

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

PRODUCED BY: CONVERGE 360, An 1105 MEDIA company

AGENDA-AT-A-GLANCE

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

START TIME	END TIME			
SQL Server Live! Full Day Hands-On Lab: Sunday, December 2, 2018				
7:30 AM	9:00 AM	Registration • Coffee and Morning Pastries		
9:00 AM	6:00 PM	**SQS01** Hands-On Lab: Developer Dive into SQL Server - *Leonard Lobel*		
2:00 PM	7:00 PM	Pre-Conference Registration - Royal Pacific Resort Conference Center		
START TIME	**END TIME**	**SQL Server Live! Pre-Conference Workshops: Monday, December 3, 2018**		
7:00 AM	8:30 AM	Registration • Coffee and Morning Pastries		
8:30 AM	5:30 PM	**SQM01** Workshop: Planning SQL Server Solutions in a Physical and Cloudy World - *Allan Hirt*	**SQM02** Workshop: How Data Science Makes Your Data Warehouse Relevant - *Bradley Ball, Josh Luedeman, & Jorge Segarra*	
6:30 PM	8:00 PM	Dine-A-Round Dinner @ Universal CityWalk - 6:30pm - Meet at Conference Registration Desk to walk over with the group		
START TIME	**END TIME**	**SQL Server Live! Day 1: Tuesday, December 4, 2018**		
7:00 AM	8:00 AM	Registration • Coffee and Morning Pastries		
8:00 AM	9:00 AM	**SQL SERVER LIVE! & ARTIFICIAL INTELLIGENCE LIVE! KEYNOTE: To Be Announced**		
9:15 AM	10:30 AM	**SQT01** Availability Fundamentals for SQL Server - *Allan Hirt*	**SQT02** What's New with Azure Data Factory - *Josh Luedeman*	**SQT03** Azure Cosmos DB Part I - Introduction to Cosmos DB - *Leonard Lobel*
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	1:50 PM	**SQT04** Fast Focus: SQL Server Data Security and Privacy Features - *Thomas LaRock*	**SQT05** Fast Focus: Common T-SQL Coding Mistakes and How to Fix Them - *Amy Herold*	
2:00 PM	2:20 PM	**SQT06** Fast Focus: Graph DB Support in SQL Server - *Karen Lopez*	**SQT07** Fast Focus: SQL Server Execution Plans - *Grant Fritchey*	
2:20 PM	2:45 PM	Networking Break • Visit the EXPO - *Pacifica 7*		
2:45 PM	4:00 PM	**SQT08** Data Protection and Privacy in the Database World - *Grant Fritchey*	**SQT09** Introduction to SQL Server Essential Concepts - *Bradley Ball*	**SQT10** Parameterization and Performance in SQL Server - *Mindy Curnutt*
4:15 PM	5:30 PM	**SQT11** Exploring Execution Plans - *Grant Fritchey*	**SQT12** Everything You Need to Know About SQL Server Indexes - *Janis Griffin*	**SQT13** Reading Between the Lines - Using XEvents to Diagnose Application Issues - *Mindy Curnutt*
5:30 PM	7:30 PM	Exhibitor Reception - *Pacifica 7*		
START TIME	**END TIME**	**SQL Server Live! Day 2: Wednesday, December 5, 2018**		
7:30 AM	8:00 AM	Registration • Coffee and Morning Pastries		
8:00 AM	9:15 AM	**SQW01** Top Tips for Deploying AGs and FCIs On Premises or In the Cloud - *Allan Hirt*	**SQW02** Level Up Your SQL Server Cloud Skills - *David Klee*	**SQW03** Azure Cosmos DB Part II – Building Cosmos DB Applications - *Leonard Lobel*
9:30 AM	10:45 AM	**SQW04** What's New in the 2017 Query Store - *Janis Griffin*	**SQW05** Redundant Devs & DBAs - Adaptive Query Processing and Automatic Tuning - *Pinal Dave*	**SQW06** Power BI - What Have You Done for Me Lately - *Andrew Brust*
10:45 AM	11:30 AM	Networking Break • Visit the EXPO - *Pacifica 7*		
11:30 AM	12:30 PM	**LIVE! 360 KEYNOTE: Enterprise Transformation (And You Can Too)** *Donovan Brown, Principal DevOps Manager, Cloud Developer Advocacy Team, Microsoft*		
12:30 PM	1:30 PM	Birds-of-a-Feather Lunch		
1:30 PM	2:00 PM	Dessert Break • Visit the EXPO		
2:00 PM	3:15 PM	**SQW07** Leveraging Data Value with Azure Data Catalog - *Karen Lopez*	**SQW08** Upgrading to SQL Server 2017 - *Thomas LaRock*	**SQW09** Design Techniques for Improving Power BI Visualizations - *Ginger Grant*
3:15 PM	4:00 PM	Networking Break • Visit the EXPO • Expo Raffle @ 3:30 p.m. - *Pacifica 7*		
4:00 PM	5:15 PM	**SQW10** The New Rules of SQL Server Monitoring - *Grant Fritchey*	**SQW11** Table Indexing - *Denny Cherry*	**SQW12** SQL Server Internals and Optimal Configuration for Machine Learning Services - *Ginger Grant*
7:30 PM	9:00 PM	Live! 360 Dessert Luau - *Wantilan Pavilion*		
START TIME	**END TIME**	**SQL Server Live! Day 3: Thursday, December 6, 2018**		
7:30 AM	8:00 AM	Registration • Coffee and Morning Pastries		
8:00 AM	9:15 AM	**SQH01** Cosmos DB for SQL Server Pros - *David Klee*	**SQH02** Performance in 60 Seconds – SQL Tricks Everybody MUST Know - *Pinal Dave*	**SQH03** How Modern is Your Data Warehouse? - *Stacia Varga*
9:30 AM	10:45 AM	**SQH04** HiHo! HiHo! SQL Server on Linux, We Go! - *Janis Griffin*	**SQH05** SQL Server 2016 Database Administration for the non-DBA - *Denny Cherry*	**SQH06** Where Does R Fit Into Your SQL Server Stack? - *Stacia Varga*
11:00 AM	12:00 PM	**SQL SERVER LIVE! & ARTIFICIAL INTELLIGENCE LIVE! PANEL DISCUSSION: Keeping Pace with AI and Machine Learning While Maintaining Your Day Job** *Andrew Brust (moderator); Thomas LaRock, Jen Stirrup, Jen Underwood, & Stacia Varga*		
12:00 PM	1:00 PM	Lunch on the Lanai - *Lanai / Pacifica 7*		
1:00 PM	2:15 PM	**SQH07** SQL Server Design Features Contentious Issues - *Karen Lopez*	**SQH08** Virtual SQL Servers, Actual Performance - *David Klee*	**SQH09** Let's Pretend This Never Happened: Common T-SQL Coding Mistakes and How to Fix Them - *Amy Herold*
2:30 PM	3:45 PM	**SQH10** SQL Server Audit - *Thomas LaRock*	**SQH11** Secrets of SQL Server - Database Worst Practices - *Pinal Dave*	**SQH12** Real-World PowerShell for the DBA - *Amy Herold*
4:00 PM	5:00 PM	Next? Live! 360 Networking Event *Heidi Araya, Andrew Brust, Jeremy Clark, Ben Curry, Benjamin Day, Peter De Tender, Brent Edwards, Kevin Ford, Grant Fritchey, Esteban Garcia, Seth Juarez, Dave Kawula, Cristal Kawula, Sami Laiho, Thomas LaRock, Vishwas Lele, Rockford Lhotka, Karen Lopez, Matthew McDermott, Brian Randell, Paul Schaeflein, Jen Underwood, Rob Windsor*		
START TIME	**END TIME**	**SQL Server Live! Post-Conference Workshops: Friday, December 7, 2018**		
7:30 AM	8:00 AM	Registration • Coffee and Morning Pastries		
8:00 AM	5:00 PM	**SQF01** Workshop: Migrating Data and Databases to Microsoft Azure - *Karen Lopez & Thomas LaRock*	**SQF02** Workshop: Data Due Diligence – Developing a Strategy for BI, Analytics, and Beyond - *Stacia Varga*	

Speakers and sessions subject to change

JAMES MCCAFFREY

Sentiment Analysis Using CNTK

Imagine you have text data, such as a collection of e-mail messages or online product reviews, and you want to determine if the overall feeling is positive or negative (or possibly neutral). That's the goal of sentiment analysis. In this article I show you how to create a sentiment analysis system using the Microsoft CNTK code library.

Take a look at the screenshot in **Figure 1** to see where this article is headed. The demo program uses the IMDB movie review dataset, which has a total of 50,000 reviews. There are 25,000 reviews in the training set and 25,000 reviews in the test set. Each set has 12,500 positive reviews and 12,500 negative reviews.

The demo program uses a small subset of the IMDB dataset—only reviews that have 50 words or less. Behind the scenes, the demo uses the CNTK library to create a long, short-term memory (LSTM) neural network and trains it using 400 iterations. After training, the model's accuracy on the held-out test reviews is 60.12 percent—not very good because of the small dataset size. The demo concludes by making a prediction for a previously unseen review of "I like this movie." The numeric prediction is (0.2740, 0.7259) and because the second value is larger than the first, the prediction is that this is a positive review.

This article assumes you have intermediate or better programming skill with a C-family language and a basic familiarity with neural networks, but doesn't assume you know much about LSTM networks. The entire demo code is presented in the article, and the associated data files are available in the accompanying download.

Understanding the Data

The key to understanding the sentiment analysis demo is understanding the structure of the data files. Suppose you have just two reviews: "I like this movie" and "Movie is terrible." A CNTK-format data file would look like:

```
0 |x 12:1 |y 0 1
0 |x 407:1
0 |x 13:1
0 |x 20:1
1 |x 20:1 |y 1 0
1 |x 9:1
1 |x 387:1
```

The first value on each line is a sequence number. The "|x" and "|y" tags indicate the start of input and output values, respectively. For output, a negative review is encoded as (1, 0) and a positive review is encoded as (0, 1). Each word has an integer index value: "i" = 12, "like" = 407, "this" = 13, "movie" = 20, "is" = 9 and "terrible" = 387.

Code download available at msdn.com/magazine/1018magcode.

The LSTM model expects input to be one-hot encoded. This is a vector consisting of all zeros except for a single one value at the index of the associated word. For example, if an entire vocabulary consists of just the four words "good," "bad," "ugly," "neutral," then "good" = (1, 0, 0, 0), "bad" = (0, 1, 0, 0), "ugly" = (0, 0, 1, 0) and "neutral" = (0, 0, 0, 1).

The IMDB movie review dataset can be found in several locations on the Internet. The dataset has 129,888 distinct words in the training set. Therefore, if you used a direct one-hot encoding approach, each word would be encoded as a huge vector with 129,887 0s and a single 1, which is extremely inefficient. CNTK

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

C:\CNTK\IMDB\EmbeddingLayer>python imdb_lstm.py

Begin IMDB sentiment analysis with CNTK LSTM

Selected CPU as the process wide default device.
Training model . . . .

i = 0
curr mini-batch has 9 sequences
accuracy curr mb = 44.44%
loss curr mb = 0.6932
----------
i = 80
curr mini-batch has 9 sequences
accuracy curr mb = 66.67%
loss curr mb = 0.6358
----------
i = 160
curr mini-batch has 9 sequences
accuracy curr mb = 44.44%
loss curr mb = 0.7171
----------
i = 240
curr mini-batch has 10 sequences
accuracy curr mb = 30.00%
loss curr mb = 0.8575
----------
i = 320
curr mini-batch has 8 sequences
accuracy curr mb = 62.50%
loss curr mb = 0.6197
----------

Training complete

Evaluating model on test items
Classification accuracy on all test items = 60.12%
Predicting sentiment for 'i like this movie'
where [1 0] is negative and [0 1] is positive
[[0.27407956 0.72592044]]

C:\CNTK\IMDB\EmbeddingLayer>
```

Figure 1 **Sentiment Analysis Using CNTK**

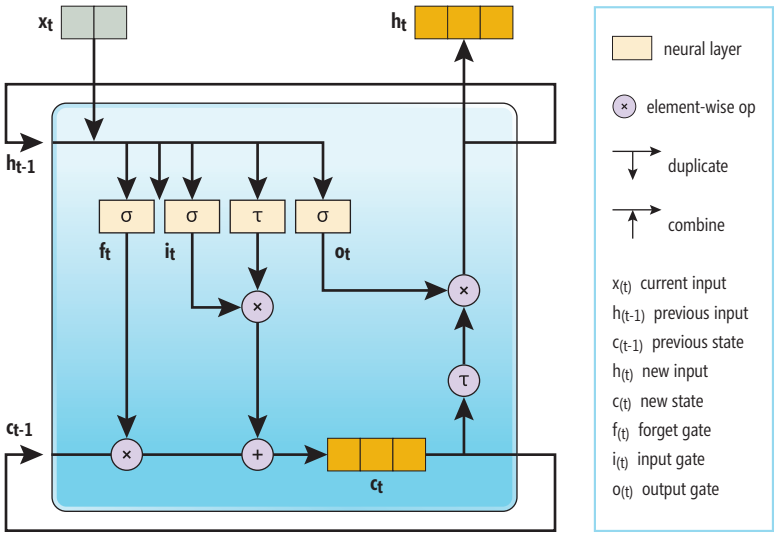


Figure 2 **A Simplified LSTM Cell**

supports a sparse format. For example, the 12:1 means, "place a 1 at index 12 and make all the other values 0." The total number of digits must be inferred or supplied elsewhere.

In principle, the index values for each word can be anything because they act only as unique IDs. However, the demo program encodes each word using a scheme that's fairly common for natural language systems. Words are encoded according to frequency in the source corpus, where 4 is assigned to the most frequent word, 5 is assigned to the second most frequent word and so on.

A value of 0 is reserved for padding in situations where you want all input sequences to have the same length. A value of 1 is used for "start of sequence" in situations where data isn't organized with an explicit delimiter (typically a newline). A value of 2 is used for "out of vocabulary" for words that are unknown because they didn't occur in a training dataset. A value of 3 is reserved for custom usage.

There are a lot of details to consider when preparing data for a sentiment analysis model.

Before starting to work on the LSTM model, I wrote a custom program to generate a file of training data and a file of test data. In very high-level pseudo-code, here's what it does:

```
read all 50,000 training and test reviews into memory,
  removing punctuation and converting to lowercase.
create a vocabulary collection of all unique words
  in training data.
sort the collection by word frequency,
  1 = most frequent.
loop through each review
  if review has more than 50 words, skip it
  write output to train, test file in CNTK format.
end-loop.
```

There are a lot of details to consider when preparing data for a sentiment analysis model. I removed all punctuation characters except for the single-quote character to retain words such as don't and wouldn't. All words in the reviews were converted to lowercase. I didn't remove any stop words such as "and" and "the." The final training file contains 620 reviews and the test file has 667 reviews. I also created a tiny file containing two reviews of "I like this movie" and "Movie is terrible."

The IMDB movie review dataset is binary because reviews are either positive or negative. For binary classification, it's common to encode negative as 0 and positive as 1. However, in order to make the demo program easily adaptable to a multi-class problem (for example, where a review can be positive, negative or neutral) I encoded negative as (1, 0) and positive as (0, 1).

Understanding LSTM Networks

A sentence is a sequence of words. In most cases the meaning of a word is highly dependent on the preceding words. For example, suppose you have a review of, "I wish I could say this was a great movie." A naive approach that just looks at single words would see "great" and probably conclude the review is positive. LSTM networks have a memory, so they can handle sequence data, specifically sentences.

There are many variations of LSTM networks and closely related networks, such as gated recurrent units (GRUs). You can get an idea of how LSTMs work by examining the diagram in **Figure 2**. The diagram shows a simplified LSTM cell. As you'll see shortly, an LSTM network consists of one or more LSTM cells plus additional plumbing.

The x(t) is the input at time t, which for sentiment analysis is a word in a sentence. The h(t) is the output at time t. The c(t) is the cell state, or memory, at time t. In the diagram, the cell state is shown as having the same size as the output vector, but in most cases the cell state will be larger (so the LSTM would require some additional components to scale the vector sizes correctly). Notice that the output at time t depends on the input and the cell state, and that the output at time t-1 and cell state at time t-1 contribute to both cell state and cell output.

Overall Program Structure

Installing CNTK involves two main steps. First, you install a Python distribution, then you install CNTK as an add-on package. A Python distribution consists of a specific version of the core Python interpreter plus several hundred useful packages that have been verified to work with the Python version and with each other. In the open source world, version incompatibilities can be a nightmare.

I used the Anaconda3 4.1.1 distribution that has Python 3.5.2. You can find this by doing an Internet search for "anaconda archive." After installing Anaconda, I searched for "cntk python" and found a .whl installer file for CNTK 2.4 and downloaded it to my local machine. You can think of a .whl file as somewhat similar to a Windows .msi file. I installed CNTK 2.4 by launching a command shell and then using the pip utility:

```
> pip install C:\MyWheels\cntk-2.4-cp35-cp35m-win_amd64.whl
```

The structure of the demo program, with a few minor edits to save space, is shown in **Figure 3**. I use Notepad as my CNTK editor of choice, but most of my colleagues prefer something more sophisticated.

Program-defined function create_reader looks for tag "|x" and "|y" in a CNTK format data file. The definition is shown in **Figure 4**.

Notice that the x-input values are indicated as sparse, but the y-output values are not. The main function sets the global NumPy random number generator seed to 1 so that results are reproducible. Then the demo sets up the key variables:

```
train_file = ".\\Data\\imdb_sparse_train_50w.txt"
test_file = ".\\Data\\imdb_sparse_test_50w.txt"
input_dim = 129888 + 4
output_dim = 2
X = C.sequence.input_variable(shape=input_dim,
  is_sparse=True)
Y = C.ops.input_variable(output_dim)
```

Recall that the IMDB training dataset has 129,888 distinct words and that the datafiles use an offset of 4. If you're performing sentiment analysis on your own data, you'll need to remember to record the number of distinct words when you create your training and test data files. Next, the demo sets up a data reader for the training data:

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

It's important to process training data in a random order. The sweeps parameter is set so that the training data can be traversed multiple times.

Defining the LSTM Model

The demo program creates the LSTM model with these statements:

```
my_init = C.initializer.glorot_uniform(seed=1)
lstm = C.layers.Sequential([
  C.layers.Embedding(50, init=my_init),
  C.layers.Recurrence(C.layers.LSTM(25)),
  C.sequence.last,
  C.layers.Dense(output_dim, init=my_init)])
model = lstm(X)
```

LSTM networks are often highly sensitive to how the model weights are initialized. The glorot_uniform algorithm is often used, but there are several alternatives, including random uniform and random normal.

> If you're performing sentiment analysis on your own data, you'll need to remember to record the number of distinct words when you create your training and test data files.

The Embedding layer is critical. In theory, you can pass the raw index values that represent words directly to an LSTM network, but in practice you should convert each word into a vector of multiple values. In this case, the Embedding layer will convert each input word into a vector of 50 values.

The code sets up a single LSTM cell with a cell state/memory size of 25. The embedding vector size and the LSTM cell size are free parameters and good values must be determined by trial and error. The model's last layer uses the Dense function so that the output is constrained to two values because the target output is either (1, 0) or (0, 1).

Training and Evaluating the LSTM Model

The demo prepares training with these statements:

```
max_iter = 400
batch_size = 10 * 40  # ~9 sequences
lr = C.learning_parameter_schedule_per_sample(0.1)
sgd = C.sgd(model.parameters, lr)
tr_loss = C.cross_entropy_with_softmax(model, Y)
tr_accu = C.classification_error(model, Y)
trainer = C.Trainer(model, (tr_loss,tr_accu), [sgd])
```

Because an average review length is about 40 words, setting the batch_size to 400 will fetch approximately nine or 10 reviews per training batch. For simplicity, the demo uses the basic stochastic gradient descent algorithm, which is rarely the best choice for LSTM networks. The Adam, AdaGrad and RMSprop algorithms often work better. **Figure 5** shows how training is performed.

A progress message is displayed every 400 iterations / 5 = 80 iterations to monitor loss and accuracy on the current batch of training

Figure 3 **Overall Program Structure**

```
# imdb_lstm.py
import numpy as np
import cntk as C

def create_reader(path, input_dim, output_dim,
  is_random, sweeps):
def main():
  # get started
  print("Begin IMDB demo")
  np.random.seed(1)

  # define LSM model
  # train model
  # evaluate model
  # make a prediction
if __name__ == "__main__":
  main()
```

Figure 4 **The create_reader Definition**

```
def create_reader(path, input_dim, output_dim,
  is_random, sweeps):
  x_strm = C.io.StreamDef(field='x', shape=input_dim,
    is_sparse=True)
  y_strm = C.io.StreamDef(field='y',
   shape=output_dim, is_sparse=False)
  streams = C.io.StreamDefs(x_src=x_strm,
    y_src=y_strm)
  deserial = C.io.CTFDeserializer(path, streams)
  mb_source = C.io.MinibatchSource(deserial,
    randomize=is_random, max_sweeps=sweeps)
  return mb_source
```

Figure 5 **Training the LSTM Model**

```
for i in range(max_iter):
  mb = rdr.next_minibatch(batch_size, input_map=imdb_map)
  trainer.train_minibatch(mb)
  if i % int(max_iter/5) == 0:
    print("i = %d" % i)
    num_seqs = mb[Y].num_sequences
    print("curr mini-batch has %d sequences" % num_seqs)
    curr_class_err = (1.0 - \
trainer.previous_minibatch_evaluation_average) * 100
    print("accuracy curr mb = %0.2f%%" % curr_class_err)
    curr_loss = trainer.previous_minibatch_loss_average
    print("loss curr mb = %0.4f" % curr_loss)
```

items. After training completes, the demo evaluates the model by applying it to the test data. First, a new reader object is created:

```
rdr = create_reader(test_file, input_dim,
  output_dim, is_random=False, sweeps=1)
imdb_map = {
  X : rdr.streams.x_src,
  Y : rdr.streams.y_src
}
```

Notice that there's no need to traverse the test data in random order, and you only need to traverse one time. The classification accuracy is computed and displayed:

```
num_to_test = 500 * 25000
all_test = rdr.next_minibatch(num_to_test,
  input_map=imdb_map)
class_acc = (1.0 - trainer.test_minibatch(all_test)) * 100
print("Classification accuracy on all test items = \
 %0.2f%%" % class_acc)
```

The num_to_test variable is set to capture far more reviews than there are in the test data. Because the reader is configured to traverse the test data only once, the excess count will be ignored. One of the minor quirks of CNTK is that there's a classification error function (percentage incorrect) rather than a classification accuracy function, so the demo subtracts from 1 to get an accuracy.

Making a Prediction

To make a sentiment prediction on a new, previously unseen review, the easiest approach is to set up a file that has the same structure as the training or test data. The demo sets up a reader for the file:

```
review_file = ".\\Data\\my_reviews_50w.txt"
rdr = create_reader(review_file, input_dim,
  output_dim, is_random=False, sweeps=1)
imdb_map = {
  X : rdr.streams.x_src,
  Y : rdr.streams.y_src
}
```

Next, a single review is read into memory as a CNTK mini-batch object and output is computed using the eval method:

```
review_mb = rdr.next_minibatch(1, input_map=imdb_map)
model = C.ops.softmax(model)
predicted = model.eval(review_mb[X])
print(predicted)
```

Before calling eval, the model is modified using the softmax function. This coerces the two output values to sum to 1.0, which makes interpreting the result a bit easier.

Wrapping Up

As recently as about 24 months ago, creating a custom sentiment analysis model would have been considered an extreme leading-edge application of deep learning, and you likely would have had to use a canned solution such as Azure Cognitive Services. But the emergence of the CNTK library, as well as similar libraries such as Keras/TensorFlow, make custom sentiment analysis feasible. To be sure, sentiment analysis is not an easy problem, but even so, the demo program presented in this article should give you all the information you need to get started on a production-quality system. ■

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: *Joey Carson, Si-Qing Chen, Eunice Kim, Lucas Meyer*

LIVE! 360
TECH EVENTS WITH PERSPECTIVE

Office & SharePoint LIVE!

ON-PREMISE, CLOUD & CROSS-PLATFORM TRAINING

ROYAL PACIFIC RESORT AT UNIVERSAL ORLANDO
December 2-7, 2018

Training to Collaborate Anywhere

People are expected to work from anywhere at any time. Office & SharePoint Live! provides leading-edge knowledge and training to administrators and developers who must customize, deploy and maintain SharePoint Server on-premises and in Office 365 to maximize the business value. In addition, this event will also cover Office 365 and Office clients for developers & administrators. ***Office & SharePoint Live! brings together Managers, IT Pros, DBAs and Developers for 6 days of workshops, keynotes and sessions to help you work through your most pressing projects.***

TRACK TOPICS INCLUDE:

- Managing Office 365
- SharePoint, Office 365 and the Cloud
- High-Value SharePoint Workloads: Search, Flow and PowerApps
- Developing for Office, Office 365 and SharePoint
- Office 365 Identity and Security

REGISTER NOW

SAVE $400 WITH SUPER EARLY BIRD PRICING!
REGISTER BY OCTOBER 5

Use Promo Code MSDN

A Part of Live! 360: The Ultimate Education Destination

6 CONFERENCES, 1 GREAT PRICE

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

SQLLIVE360.COM

EVENT PARTNERS
Microsoft
Magenic

SILVER SPONSOR
Progress
GOVERLAN REACH

PREMIER & ALLIANCE MEDIA PARTNERS
netda
UserGroup.TV

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

PRODUCED BY
CONVERGE 360
An 1105 MEDIA company

AGENDA-AT-A-GLANCE

Managing Office 365	SharePoint, Office 365 and the Cloud	High-Value SharePoint Workloads: Search, Flow, and PowerApps	Developing for Office, Office 365 and SharePoint	Office 365 Identity and Security

START TIME	END TIME	
		Office & SharePoint Live! Full Day Hands-On Lab: Sunday, December 2, 2018
7:30 AM	9:00 AM	Registration • Coffee and Morning Pastries
9:00 AM	6:00 PM	**OSS01** Hands-On Lab: Your Search is Broken - Let's Fix It! - *Agnes Molnar and Matthew McDermott*
2:00 PM	7:00 PM	Pre-Conference Registration - Royal Pacific Resort Conference Center
START TIME	**END TIME**	**Office & SharePoint Live! Pre-Conference Workshop: Monday, December 3, 2018**
7:00 AM	8:30 AM	Registration • Coffee and Morning Pastries
8:30 AM	5:30 PM	**OSM01** Workshop: Modern Application Development with Office, SharePoint and Azure - *Eric Shupps*
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**	**Office & SharePoint Live! Day 1: Tuesday, December 4, 2018**
7:00 AM	8:00 AM	Registration • Coffee and Morning Pastries
8:00 AM	9:00 AM	**OFFICE & SHAREPOINT LIVE! KEYNOTE: What is Keeping You from the Cloud? It's You!** *Neil Hodgkinson, Senior Program Manager, O365 CXP Team, Microsoft and Aimme Triplet, Engineer, Fast Track Center, Microsoft*
9:15 AM	10:30 AM	**OST01** Getting Started with Office 365 Groups, Team Sites and OneDrive for Business - *Dan Usher* **OST02** An Overview of the Office Development Platform - *Bill Ayers*
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	1:50 PM	**OST03** Fast Focus: The 5 Most Important Aspects to Deploying Office 365 - *Ben Stegink*
2:00 PM	2:20 PM	**OST04** Fast Focus: 5 Query Rules Every SharePoint Pro Should Know - *Matthew McDermott*
2:20 PM	2:45 PM	Networking Break • Visit the EXPO - *Pacifica 7*
2:45 PM	4:00 PM	**OST05** What's New and Improved in SharePoint 2019 - *Neil Hodgkinson and Aimme Triplet* **OST06** Developing SharePoint Framework Solutions for the Enterprise - *Eric Shupps*
4:15 PM	5:30 PM	**OST07** Why You Need to Invest Into Search and How to Start this Journey? - *Agnes Molnar* **OST08** An Introduction to Development with the SharePoint Framework (SPFx) - *Rob Windsor*
5:30 PM	7:30 PM	Exhibitor Reception - *Pacifica 7*
START TIME	**END TIME**	**Office & SharePoint Live! Day 2: Wednesday, December 5, 2018**
7:30 AM	8:00 AM	Registration • Coffee and Morning Pastries
8:00 AM	9:15 AM	**OSW01** Managing and Configuring Office 365 Identity - *Scott Hoag* **OSW02** Introduction to Azure Web Applications for Office and SharePoint Developers - *Eric Shupps*
9:30 AM	10:45 AM	**OSW03** Collaborating, Inside and Out - Building Secure Sharing Solutions - *Ben Curry* **OSW04** Enterprise JavaScript Development Patterns - *Rob Windsor*
10:45 AM	11:30 AM	Networking Break • Visit the EXPO - *Pacifica 7*
11:30 AM	12:30 PM	**LIVE! 360 KEYNOTE: Enterprise Transformation (And You Can Too)** *Donovan Brown, Principal DevOps Manager, Cloud Developer Advocacy Team, Microsoft*
12:30 PM	1:30 PM	Birds-of-a-Feather Lunch
1:30 PM	2:00 PM	Dessert Break • Visit the EXPO
2:00 PM	3:15 PM	**OSW05** Your Office 365 Environment by the Numbers - *Ben Stegink* **OSW06** Microsoft Graph: The Toolkit for Building Modern Business Solutions - *Bill Ayers*
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	**OSW07** Setting Up Directory Synchronization for Office 365 - *Dan Usher* **OSW08** Smart UI with Adaptive Cards - Reducing Friction and Improve Productivity - *Paul Schaeflein*
7:30 PM	9:00 PM	Live! 360 Dessert Luau - *Wantilan Pavilion*
START TIME	**END TIME**	**Office & SharePoint Live! Day 3: Thursday, December 6, 2018**
7:30 AM	8:00 AM	Registration • Coffee and Morning Pastries
8:00 AM	9:15 AM	**OSH01** Managing Office 365 Data on Mobile Devices - *Ben Curry* **OSH02** Introduction to the SharePoint Developer PnP Core Libraries - *Rob Windsor*
9:30 AM	10:45 AM	**OSH03** Securing Office 365 and SharePoint with Azure Active Directory - *Scott Hoag* **OSH04** PowerApps - Beyond the Basics - *Paul Schaeflein*
11:00 AM	12:00 PM	**OFFICE & SHAREPOINT LIVE! PANEL DISCUSSION: Is It Time for IT Pros to Start Trusting Developers?** *Matthew McDermott (Moderator); Ben Curry, Paul Schaeflein, & Rob Windsor*
12:00 PM	1:00 PM	Lunch on the Lanai - *Lanai / Pacifica 7*
1:00 PM	2:15 PM	**OSH05** SharePoint Search and Taxonomy Better Together - *Matthew McDermott* **OSH06** TypeScript for SharePoint Developers - *Rob Windsor*
2:30 PM	3:45 PM	**OSH07** Automating SharePoint Online with Flow, PowerShell and Azure - *Ben Stegink* **OSH08** Reach for the Cloud: Extend your SharePoint Framework Solutions with the Power of Microsoft Graph - *Bill Ayers*
4:00 PM	5:00 PM	Next? Live! 360 Networking Event *Heidi Araya, Andrew Brust, Jeremy Clark, Ben Curry, Benjamin Day, Peter De Tender, Brent Edwards, Kevin Ford, Grant Fritchey, Esteban Garcia, Seth Juarez, Dave Kawula, Cristal Kawula, Sami Laiho, Thomas LaRock, Vishwas Lele, Rockford Lhotka, Karen Lopez, Matthew McDermott, Brian Randell, Paul Schaeflein, Jen Underwood, Rob Windsor*
START TIME	**END TIME**	**Office & SharePoint Live! Post-Conference Workshop: Friday, December 7, 2018**
7:30 AM	8:00 AM	Registration • Coffee and Morning Pastries
8:00 AM	5:00 PM	**OSF01** Workshop: Developing Components for SharePoint and Teams - *Paul Schaeflein*

Speakers and sessions subject to change

Reading the T Leaves

I just had a blast teaching Xamarin.Forms at Harvard's summer session. The keystone of any programmer's education is building a working app. I can lecture and show code samples until my head falls off, and the students can regurgitate them on exams. But students don't own those skills until they've fired them in anger against a target they need to kill if they want to eat (or, in this case, pass the class).

As always, I wanted to provide my students with a real-life project, not some artificial concoction. Last year's class built a mobile weight tracker app for Harvard Medical School (see my December 2017 column at msdn.com/magazine/mt814422) that can improve care and outcomes for patients with congestive heart failure. I found another great project this year.

The Massachusetts Bay Transportation Authority (MBTA), universally called "the T," runs most public transportation (bus, subway, rail, ferry) in eastern Massachusetts. As with most government agencies, the T is not universally respected for its competence. As columnist Howie Carr wrote in February of 2016: "The only way to stop the Trump train now might be to turn it over to the MBTA."

But the T did something extraordinary in 2016. They hired as their CTO an upstart named David Block-Schachter, co-founder and former technical lead for the startup Bridj. I read of his hiring just as I was finishing my book "The Joy of UX." Always looking to pour oil on troubled fires, I asked him to review the chapter containing my critique of the T's mobile app for commuter rail. (See my video at bit.ly/2Pmg790, and sample chapter download at bit.ly/2Pqwulj.) His response: "Completely agree with the criticism of the current app (and it's before my time, so I'm free to do so without being defensive)."

Block-Schachter realized that an organization like the T will never be able to develop great software, nor should it be asked to. What it can and should do is enable third parties to write great software. Under his guidance, the T has put all of its data (schedules, routes, stops—everything) online, accessible to any app via RESTful services. The documentation and discussion forum is online at mbta.com/developers.

With these APIs, anyone can develop any kind of app their customers want. An app can display the schedules for bus and commuter rail, updated with real-time vehicle position. Crusaders for improvement could build an app for reporting unsafe conditions or broken equipment—snap a picture, attach GPS coordinates, e-mail it to the repair staff. A local bicycling app could include the current availability of bike lockers at the nearest rail station, or the real-time schedules of busses capable of carrying bikes. The ticketing portion isn't open to developers yet, but it will be in two years when the T's plan for cashless operation is implemented.

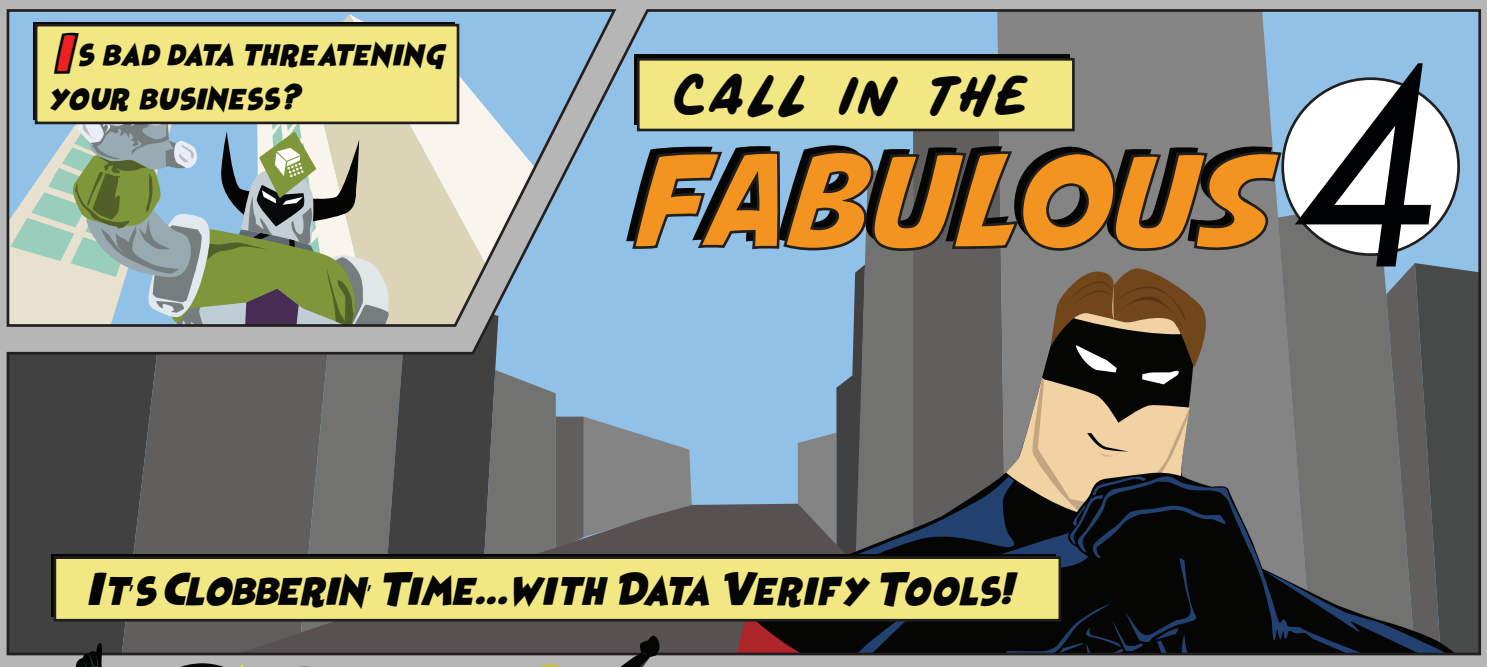
Block-Schachter and his merry men came to an early class to explain the system to my students, and then to the last class to see the results. The best ones were fabulous, and may well be deployed by the working students.

> Block-Schachter realized that an organization like the T will never be able to develop great software, nor should it be asked to.

This project was an excellent match for Xamarin.Forms, which has improved a lot since I last taught it a year ago. For example, it now seamlessly uses .NET Standard 2.0, instead of requiring custom fitting. It crashes less, works better in the debugger, and its display of IntelliSense is better. Your code can also access device features, such as accelerometers, GPS and even cameras in a platform-independent way. It reminds me somewhat of Visual Basic 6. The framework geeks understand the underlying devices and build the components, and the application programmers understand the users and their problems, and assemble these components into useful tools. If you haven't tried Xamarin.Forms yet, now is the time.

The most difficult part of a killer app, as always, is not coding this piece or that piece, it's figuring out what ought to be coded to solve the users' problems. I taught the students as much of this as I could, leading them daily in the chant, "Know Thy User, for He Is Not Thee." But I could only fit one UX lecture into the crammed schedule. You, my friend, are luckier. You can learn Xamarin UX in a hands-on workshop, working on your own project with my assistance. I'm teaching at the Microsoft NERD center in Cambridge, from Oct. 22-24. You can find information and register at joyfulxamarinapps.com. Be there! Aloha. ■

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

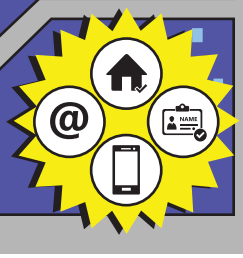


Visit Melissa Developer Portal to quickly combine our APIs (address, phone, email and name verification) and enhance ecommerce and mobile apps to prevent bad data from entering your systems.

With our toolsets, you'll be a data hero – preventing fraud, reducing costs, improving data for analytics, and increasing business efficiency.

- **Single Record & Batch Processing**
- **Scalable Pricing**
- **Flexible, Easy to Integrate Web APIs: REST, JSON & XML**
- **Other APIs available: Identity, IP, Property & Business**

Let's Team Up to Fight Bad Data Today!



melissadeveloper.com
1-800-MELISSA

melissa