THE MICROSOFT JOURNAL FOR DEVELOPERS

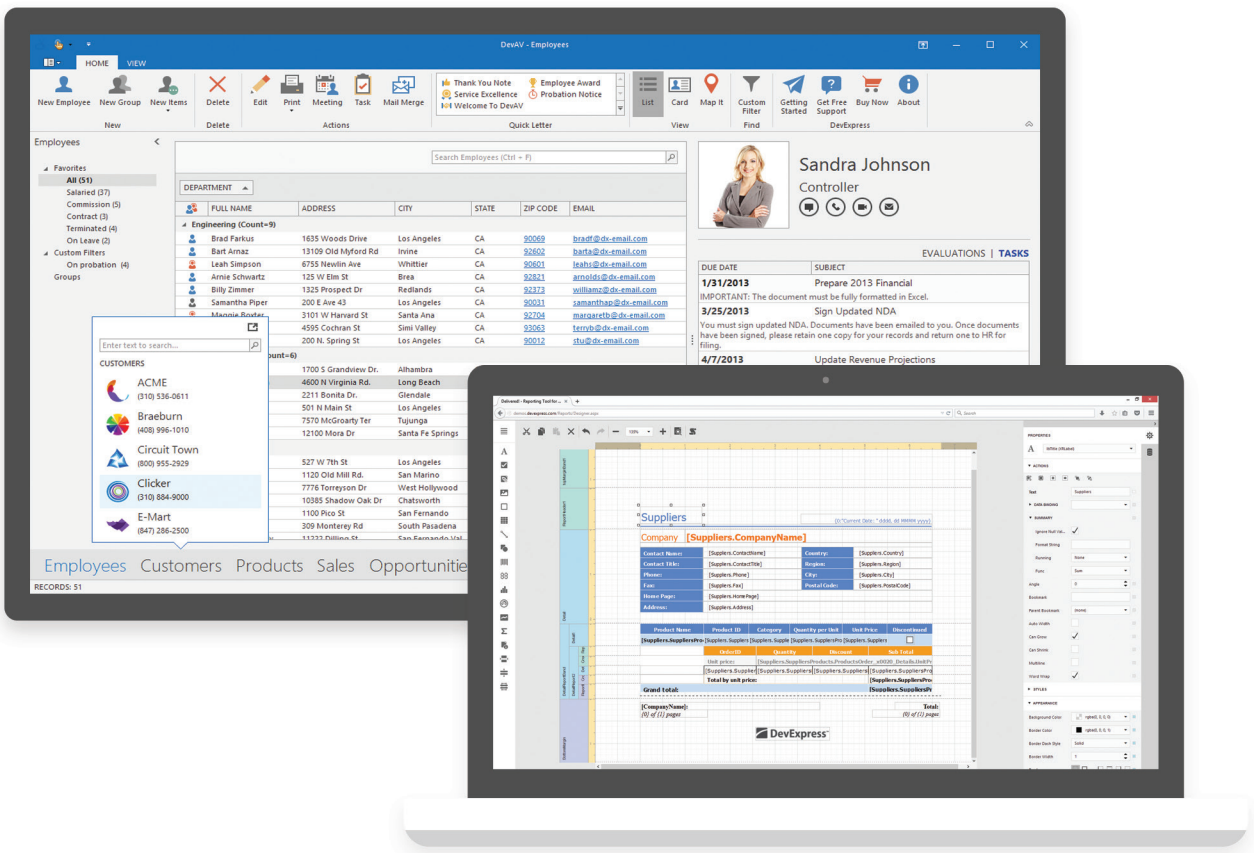
APRIL 2019 VOL 34 NO 4

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

APRIL 2019 VOL 34 NO 4

msdn magazine

Xamarin.Forms 4.0.....18

COLUMNS

Microsoft

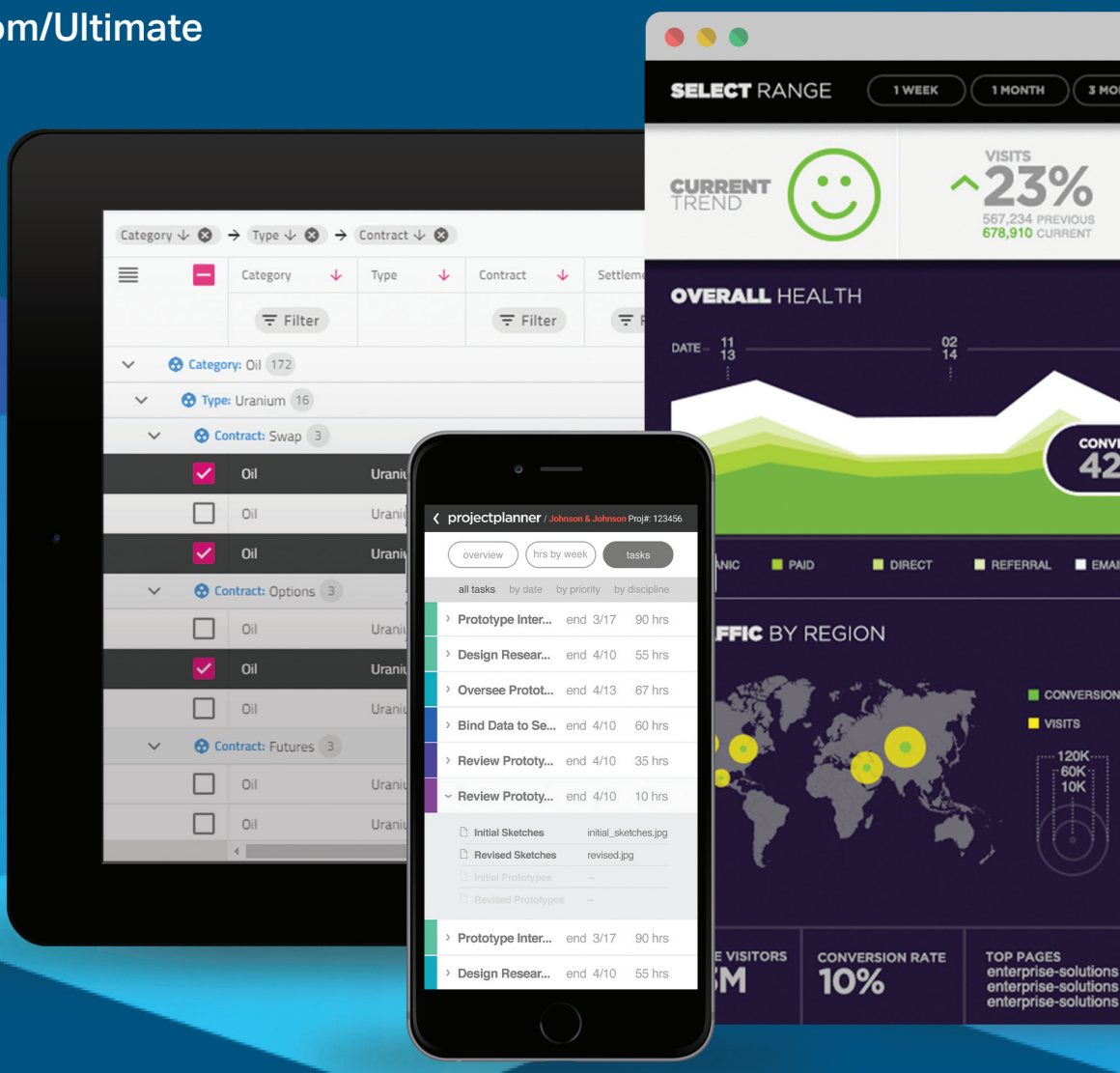
INFRAGISTICS

Faster Paths to Amazing Experiences

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

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

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



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

Infragistics Ultimate

- Fastest **grids** & **charts** on the market for the Angular developer
- The most complete Microsoft Excel and Spreadsheet solution for creating dashboards and reports without ever installing Excel
- An end-to-end design-to-code platform with Indigo.Design
- Best-of-breed charts for financial services

msdn magazine

APRIL 2019 VOLUME 34 NUMBER 4

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, File 2272, 1801 W.Olympic Blvd., Pasadena, CA 91199-2272, email MSDNmag@1105service.com or call 866-293-3194 or 847-513-6011 for U.S. & Canada; 00-1-847-513-6011 for International, fax 847-763-9564. POSTMASTER: Send address changes to MSDN Magazine, P.O. Box 2166, Skokie, IL 60076. Canada Publications Mail Agreement No: 40612608. Return Undeliverable Canadian Addresses to Circulation Dept. or XPO Returns: P.O. Box 201, Richmond Hill, ON L4B 4R5, Canada.

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

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

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

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

Reaching the Staff

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

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

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

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

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

Microsoft

BPA WORLDWIDE
1906 2006 100 AMERICAN BUSINESS MEDIA
WPA Western Publications Association

LEADTOOLS®

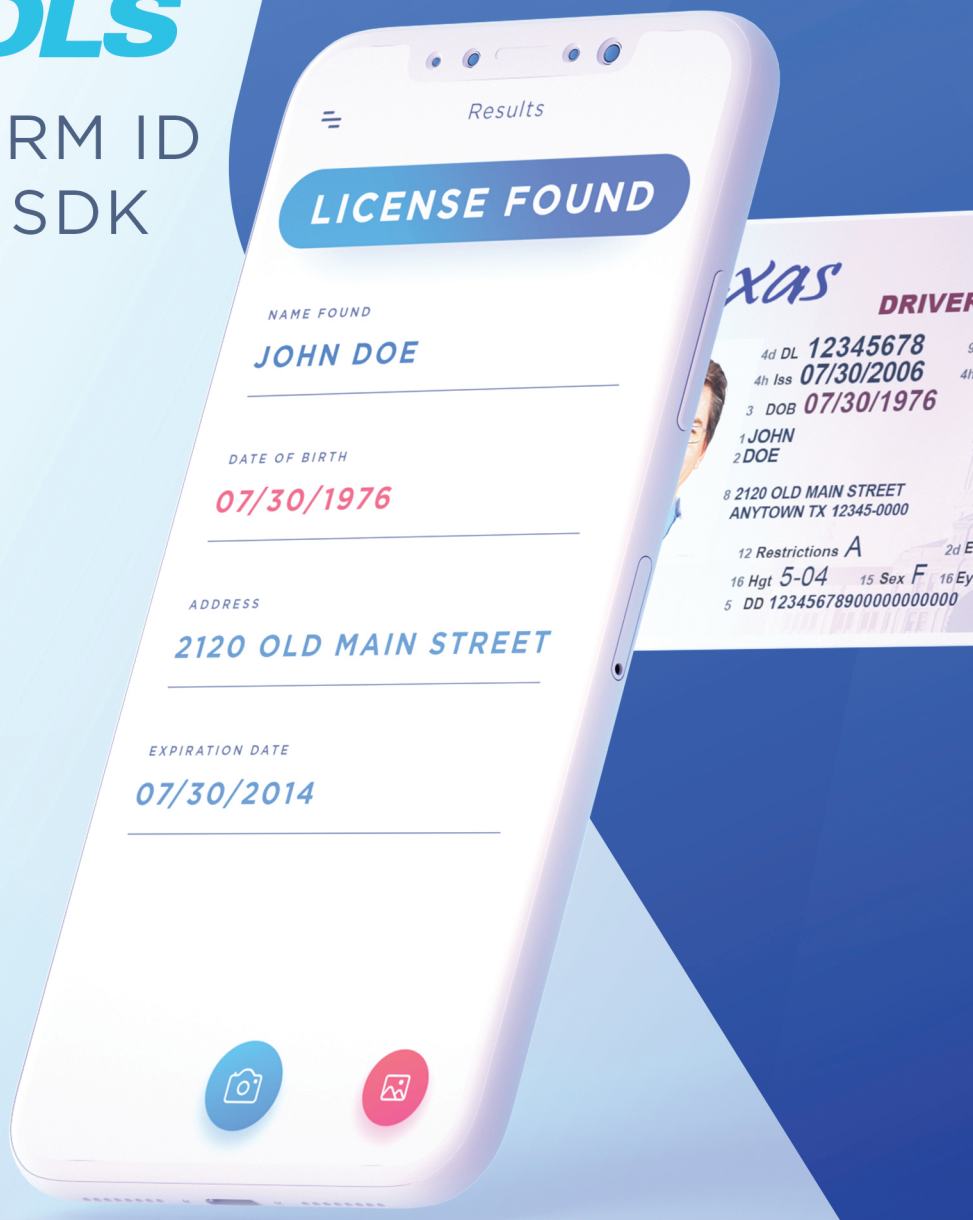
MULTI-PLATFORM ID RECOGNITION SDK

WEB

MOBILE

DESKTOP

SERVER



LEADTOOLS ID recognition and processing SDK technology provides a high-level framework to classify and extract data from driver's licenses and similar forms of identification. The state-of-the-art algorithms recognize structured and unstructured passports, driver's licenses, and ID cards from any country, state, or agency.

Microsoft .NET · Xamarin · Windows · Java · Linux · iOS · macOS · Android · HTML5

Get Started Today

DOWNLOAD OUR FREE EVALUATION

LEADTOOLS.COM

MICHAEL DESMOND

Adapting to Machine Learning

When Geoff Hulten, until recently a principal machine learning scientist at Microsoft, approached me about writing an article about machine learning principles and patterns, I was intrigued. *MSDN Magazine* is committed to providing code-level, how-to articles that help working developers improve and extend their skills. And while we've published dozens of articles that look at various aspects of machine learning over the years, we haven't drilled into the higher-level, operational concepts that should guide decision making around machine learning.

Hulten's feature, "Closed-Loop Intelligence: A Design Pattern for Machine Learning," changes that. Instead of a tactical exploration of a thorny technical problem or new tool, this article digs into patterns and practices that can be critical to a successful machine learning project. The article delves into four key aspects of machine learning management: Connecting with users to enable closed-loop feedback, applying machine learning to the right objectives, building systems to support machine learning-based solutions, and understanding what goes into running a machine learning-based system.

> Hulten says developers tend to treat machine learning errors as code defects, like bugs "to be burned down to zero."

I asked Hulten how important it is for developers to take the time to understand key principles and patterns as they embark on machine learning projects. He offered a comparison familiar to many developers.

"I remember going through my own personal transition from procedural to object-oriented programming. It was easy to fall into old habits and create an essentially procedural program with a few objects floating around. It was also easy to go into full-on, object-bizarre-o-land and create something where objects took over and turned the program into useless gibberish," Hulten says. "I think machine learning is just as powerful and transformational as object-oriented programming: To get the most out of it you need to think about problems a bit differently; you also need to develop the wisdom and judgment to use the power of machine learning appropriately."

Not surprisingly, with a significant paradigm shift, mistakes abound. I asked Hulten about common machine learning missteps, and he singled out the tendency of developers to treat machine learning as a "magical block box" that can be integrated into an application like a function call.

"This misses all the simple ways the application can support the machine learning and position it for the best impact," he explains. "Presenting the predictions for success, reducing the costs of mistakes, shaping user interactions so they create the best training data, and so much more."

Another common misconception: How to handle errors and mistakes in machine learning systems. Hulten says developers tend to treat machine learning errors as code defects, like bugs "to be burned down to zero." But he emphasizes that machine learning, by design, makes mistakes. To leverage the power of machine learning, applications must "embrace those mistakes and help reduce the damage they cause," he says.

Hulten also calls for developers to take a longer view and give thought to how their machine learning applications can evolve over time to yield improved impact and efficiency. It's advice that calls back to the closed-loop pattern described in his article.

Says Hulten: "When you put machine learning in the right position, your application should improve every single time a user interacts with it."

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

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

JULIE LERMAN

EF Core in a Docker Containerized App

I've spent a lot of time with Entity Framework and EF Core, and a lot of time working with Docker. You've seen evidence of this in many past columns. But, until now, I haven't put them together. Because the Docker Tools for Visual Studio 2017 have been around for a while, I imagined it would be an easy leap. But it wasn't. Perhaps it's because I prefer to know what's going on under the covers and I also like to know and comprehend my options. In any case, I ended up sifting through information from blog posts, articles, GitHub issues and Microsoft documentation before I was able to achieve my original goal. What I hope is to make it easier for readers of this column to find the path (and avoid some of the hiccups I encountered) by consolidating it all in one place.

I'm going to focus on Visual Studio 2017, and that means Windows, so you'll need to ensure you have Docker Desktop for Windows installed (dockr.ly/2tEQgR4) and that you've set it up to use Linux containers (the default). This also requires that Hyper-V be enabled on your machine, but the installer will alert you if necessary. If you're working in Visual Studio Code (regardless of OS), there are quite a few extensions for working with Docker directly from the IDE.

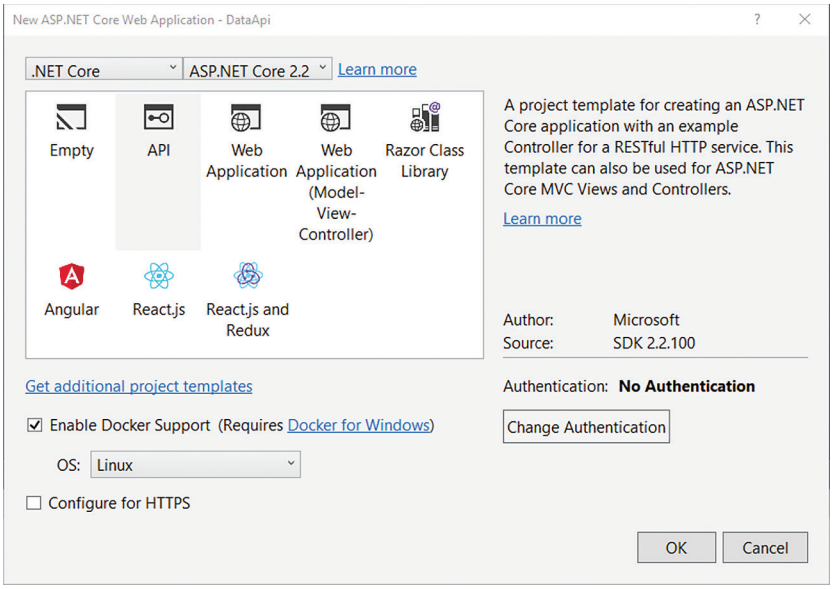


Figure 1 **Configuring the New ASP.NET Core API Project**

Creating the Project

I began my journey with a simple ASP.NET Core API. The steps to set up a new project to match mine are: New Project | .NET Core | ASP.NET Core Web Application. On the page where you choose the application type, choose API. Be sure that Enable Docker Support is checked (**Figure 1**). Leave the OS setting at Linux. Windows containers are larger and more complicated, and hosting options for them are still quite limited. I learned this the hard way.

Because you enabled Docker support, you'll see a Dockerfile in the new project. Dockerfile provides instructions to the Docker engine for creating images and for running a container based on the final image. Running a container is akin to instantiating an object from a class. **Figure 2** shows the Dockerfile created by the template. (Note that I'm using Visual Studio 2017 version 15.9.7 with .NET Core 2.2 installed on my computer. As the Docker Tools evolve, so may the Dockerfile.)

Code download available at msdn.com/magazine/0419magcode.

The first instruction identifies the base image used to create the subsequent images and container for your app is specified as:

```
microsoft/dotnet:2.2-aspnetcore-runtime
```

Then a build image will be created based on the base image. The build image is solely for building the application, so it needs the SDK, as well. A number of commands are executed on the build image to get your project code into the image, and to restore needed packages before building the image.

Figure 2 **The Default Dockerfile Created by the Project Template**

```
FROM microsoft/dotnet:2.2-aspnetcore-runtime AS base
WORKDIR /app
EXPOSE 80

FROM microsoft/dotnet:2.2-sdk AS build
WORKDIR /src
COPY ["DataApi/DataApi.csproj", "DataApi/"]
RUN dotnet restore "DataApi/DataApi.csproj"
COPY . .
WORKDIR "/src/DataApi"
RUN dotnet build "DataApi.csproj" -c Release -o /app

FROM build AS publish
RUN dotnet publish "DataApi.csproj" -c Release -o /app

FROM base AS final
WORKDIR /app
COPY --from=publish /app .
ENTRYPOINT ["dotnet", "DataApi.dll"]
```

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

The next image created will be used for publishing; it's based on the build image. For this image, Docker will run *dotnet publish* to create a folder with the minimal assets needed to run the application.

```
C:\Users\Julie>docker images -a
REPOSITORY                  TAG                        IMAGE ID        CREATED             SIZE
<none>                      <none>                     3139566cfb28    22 seconds ago      271MB
<none>                      <none>                     78ed08d1c2d0    22 seconds ago      271MB
dataapi                     dev                        2b9743c3cded    22 seconds ago      271MB
microsoft/dotnet            2.2-aspnetcore-runtime     78bd7dc97547    2 days ago          271MB
docker4w/nsenter-dockerd    latest                     2f1c802f322f    4 months ago        187kB
```

Figure 3 **Exposed and Hidden Docker Images After Running the API**

The final image has no need for the SDK and is created from the base image. All of the publish assets will get copied into this image and an Entrypoint is identified—that is, what should happen when this image is run.

By default, the Visual Studio tools only perform the first stage for the Debug configuration, skipping the publish and final images, but for a Release configuration the entire Dockerfile is used.

The various sets of builds are referred to as multi-stage builds, with each step focused on a different task. Interestingly, each command performed on an image, such as the six commands on the build image, causes Docker to create a new layer of the image. The Microsoft documentation on architecture for containerized .NET applications at bit.ly/2TeCblu does a wonderful job of explaining the Dockerfile line by line and describing how it's been made more efficient through the multi-stage builds.

For now, I'll leave the Dockerfile at its default.

Debugging the Default Controller in Docker

Before heading to Docker debug, I'll first verify the app by debugging in the ASP.NET Core self-hosted profile (which uses Kestrel, a cross-platform Web server for ASP.NET Core). Be sure that the Start Debugging button (green arrow on the toolbar) is set to run using the profile matching the name of your project—in my case that's DataAPI.

Then run the app. The browser should open pointing to the URL http://localhost:5000/api/values and displaying the default controller method results ("value1," "value2"). So now you know the app works and it's time to try it in Docker. Stop the app and change the Debug profile to Docker. If Docker for Windows is running (with the appropriate settings noted at the start of this article), Docker will run the Dockerfile. If you've never pulled the referenced images before, the Docker engine will start by pulling down those images from the Docker hub. Pulling the images may take a few minutes. You can watch its progress in the Build Output window. Then Docker will build any images following the steps in Dockerfile, though it won't rebuild any that haven't changed since the last run. The final step is performed by the Visual Studio Tools for Docker, which will call *docker build* and then *docker run* to start the container. As a result, a new browser window (or tab) will open with the same output as before, but the URL is different because it's coming from inside the Docker image that's exposing that port. In my case, it's http://172.26.137.194/api/values. Alternate setups will cause the browser to launch using http://localhost:hostPort.

If Visual Studio Can't Run Docker

I encountered two issues that initially prevented Docker from building the images. The very first time I tried to debug from Visual Studio targeting Docker, I got a message that, "An error occurred while attempting to run Docker container." The error referenced container.targets line 256. This was neither informative nor helpful until I later realized that I could have seen the details of the error in the Build Output. After trying a variety of things (including a lot of reading on the Internet, yet still not checking the Build Output window, I ended up trying to pull an image from the Docker CLI, which prompted me to log in to Docker, even though I was already logged in to the Docker Desktop app. Once I did this, I was able to debug from Visual Studio 2017. Subsequently, logging out via the Docker CLI didn't affect this and I could still debug. I'm not sure of the relationship between the two actions. However, when I completely uninstalled and reinstalled Docker Desktop for Windows, I was, again, forced to log in via the Docker CLI before I could run my app. According to the issue on GitHub at bit.ly/2Vxhsx4, it seems it's because I logged into Docker for Windows using my e-mail address, not my login name.

I also got this same error once when I had disabled Hyper-V. Re-enabling Hyper-V and restarting the machine solved the problem. (For the curious, I needed to run a VirtualBox virtual machine for a completely unrelated task and VirtualBox requires Hyper-V to be disabled.)

What Is the Docker Engine Managing?

As a result of running this app for the first time, the Docker engine pulled down the two noted images from the Docker Hub (hub.docker.com) and was keeping track of them. But the Dockerfile also created other images that it had cached. Running *docker images* at the command line revealed the docker4w image used by Docker for Windows itself, an aspnetcore-runtime image pulled from Docker Hub, and the dataapi:dev image that was created by building the Dockerfile—that is, the image from which your application is running. If you run *docker images -a* to show hidden images, you'll see two more images (those with no tags) that are the build and publish intermediate images created by Dockerfile, as shown in **Figure 3**. You won't see anything about the SDK image, according to Microsoft's Glenn Condron, "due to a quirk in the way Docker multi-stage build works."

You can look at even more details about an image using the command:

```
docker image inspect [imageid]
```

What about the containers? The command *docker ps* reveals the container created by Docker Tools for Visual Studio 2017 calling *docker run* (with parameters) on the dev image. I've stacked the result in **Figure 4** so you can see all the columns. There are no hidden containers.

Setting Up the Data API

Now let's turn this into a data API using EF Core as the data persistence mechanism. The model is simplistic in order to focus on the containerization and its impact on your EF Core data source.

DevExpress DXperience 18.2 | from $1,439.99

DevExpress

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

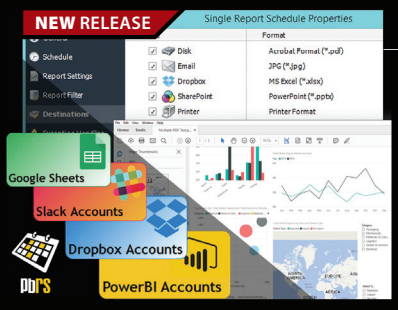
- WinForms – New Sunburst Chart, Office Navigation UX, SVG Office 2019 skins
- WPF – New Gantt control, improved Data Filtering UX and App Theme Designer
- ASP.NET & MVC – New Adaptive Layouts, improved Rich Text Editor and Spreadsheet
- Reporting – Vertical Band support, Free-hand drawing and improved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

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

LEADTOOLS
THE WORLD LEADER IN IMAGING SDKs

Powerful DICOM, PACS, and HL7 functionality.

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



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

Christiansteven

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

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

Intellifront BI | from $23,246.35

Christiansteven

Cost-Effective Data Analytics & Business Intelligence.

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

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

Begin by adding a class called Magazine.cs:

```
public class Magazine
{
  public int MagazineId { get; set; }
  public string Name { get; set; }
}
```

Next, you need to install three different NuGet packages. Because I'll be showing you the difference between using a self-contained SQLite database and a SQL Server database, add both the Microsoft.EntityFrameworkCore.Sqlite and Microsoft.EntityFrameworkCore.SqlServer packages to the project. You'll also be running EF Core migrations, so the third package to install is Microsoft.EntityFrameworkCore.Design.

Now I'll let the tooling create a controller and DbContext for the API. In case this is new for you, here are the steps:

- Right-click on the Controllers folder in Solution Explorer.
- Choose Add | Controller | API Controller with actions, using Entity Framework.
- Select the Magazine class as the Model class.
- Click the plus sign next to Data context class and change the highlighted portion of the name to Mag, so it becomes [YourApp].Models.MagContext, and then click Add.
- Leave the default controller name as MagazinesController.
- Click Add.

When you're done, you'll have a new Data folder with the MagContext class and the Controllers folder will have a new MagazineController.cs file.

Now I'll have EF Core seed the database with three magazines using the DbContext-based seeding I wrote about in my August 2018 column (msdn.com/magazine/mt829703). Add this method to MagContext.cs:

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
  modelBuilder.Entity<Magazine>().HasData(
    new Magazine { MagazineId = 1, Name = "MSDN Magazine" },
    new Magazine { MagazineId = 2, Name = "Docker Magazine" },
    new Magazine { MagazineId = 3, Name = "EFCore Magazine" }
  );
 }
```

Setting Up the Database

To create the database, I need to specify the provider and connection string, and create and then run a migration. I want to start by going down a familiar path, so I'll begin by targeting SQL Server LocalDB and specifying the connection string in the appsettings.json file.

When you open appsettings.json, you'll find it already contains a connection string, which was created by the controller tooling when I let it define the MagContext file. Even though both SQL Server and SQLite providers were installed, it seems to have defaulted to the SQL Server provider. This proved to be true in subsequent tests. I prefer my own connection string name and my own database name, so, I replaced the MagContext connection string with MagsConnectionMssql, and added my preferred database name, DP0419Mags:

```
"ConnectionStrings": {
    "MagsConnectionMssql":
      "Server=(localdb)\\mssqllocaldb;Database=DP0419Mags;Trusted_Connection=True;"
  }
```

In the app's startup.cs file, which includes a ConfigureServices method, the tooling also inserted code to configure the DbContext. Change its connection string name from MagContext to match the new name:

```
CONTAINER ID   IMAGE          COMMAND                  CREATED
a21274b515fd   dataapi:dev    "C:\\remote_debugger\\…"   16 minutes ago

   STATUS          PORTS                     NAMES
   Up 16 minutes   0.0.0.0:18183->80/tcp     eloquent_haibt
```

Figure 4 **The Docker Container Created by Running the App from Visual Studio 2017 Debug**

```
services.AddDbContext<MagContext>(options =>
  options.UseSqlServer(Configuration.GetConnectionString(
  "MagsConnectionMssql")));
```

Now I can use EF Core migrations to create the first migration and, as I'm in Visual Studio, I can do that using PowerShell commands in the Package Manager Console:

```
add-migration initMssql
```

Migrating the Database

That command created a migration file, but I'm not going to create my database using the migration commands—when I deploy my app, I don't want to have to execute migrations commands to create or update the database. Instead, I'll use the EF Core Database.Migrate method. Where this logic method goes in your app is an important decision. You need it to run when the application starts up. A lot of people interpret this to mean when the startup.cs file is run, but the ASP.NET Core team recommends placing application startup code in the program.cs file, which is the true starting point of an ASP.NET Core app. But as with any decision, there may well be factors that affect this guidance.

The program's default Main method calls the ASP.NET Core method, CreateWebHostBuilder, which performs a lot of tasks on your behalf, then calls two more methods—Build and Run:

```
public static void Main(string[] args)
{
  CreateWebHostBuilder(args).Build().Run();
}
```

I need to migrate the database after Build but before Run. To do this, I've created an extension method to read the service provider information defined in startup.cs, which will discover the DbContext configuration. Then the method calls Database.Migrate on the context. I adapted code (and guidance from EF Core team member, Brice Lambson) from the GitHub issue at bit.ly/2T19cbY to create the extension method for IWebHost shown in **Figure 5**. The method is designed to take a generic DbContext type.

Figure 5 **Extension Method for IWebHost**

```
public static IWebHost MigrateDatabase<T>(this IWebHost webHost) where T : DbContext
{
  using (var scope = webHost.Services.CreateScope())
  {
    var services = scope.ServiceProvider;
    try
    {
      var db = services.GetRequiredService<T>();
      db.Database.Migrate();
    }
    catch (Exception ex)
    {
      var logger = services.GetRequiredService<ILogger<Program>>();
      logger.LogError(ex, "An error occurred while migrating the database.");
    }
  }
  return webHost;
}
```

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

Then I modified the Main method to call MigrateDatabase for MagContext between Build and Run:

```
CreateWebHostBuilder(args).Build().MigrateDatabase<MagContext>().Run();
```

As you're adding all of this new code, Visual Studio should prompt you to add using statements for Microsoft.EntityFrameworkCore, Microsoft.Extensions.DependencyInjection and the namespace for your MagContext class.

Now the database will get migrated (or even created) as needed at runtime.

One last step before debugging is to tell ASP.NET Core to point to the Magazines controller when starting, not the values controller. You can do that in the launchsettings.json file, changing the instances of launchUrl from api/values to api/Magazines.

Running the Data API in Kestrel, Then in Docker

As I did for the values controller, I'm going to start by testing this out in the self-hosted server via the project profile (for example, DataAPI), not the Docker profile. Because the database doesn't yet exist, Migrate will create the new database, which means there will be a short delay because SQL Server, even LocalDB, has a lot of work to do. But the database does get created and seeded and then the default controller method reads and displays the three magazines in the browser at localhost:5000/api/Magazines.

Now let's try it out with Docker. Change the Debug profile to Docker and run it again. Oh no! When the browser opens, it displays a SQLException, with details explaining that TryGetConnection failed.

What's going on here? The app is looking for the SQL Server (defined as "(localdb)\\mssqllocaldb" in the connection string) inside the running Docker container. But LocalDB is installed on my computer and not inside the container. Even though it's a common choice for preparing for a SQL Server database when you're in a development environment, it doesn't work so easily when you're targeting Docker containers.

This means I have more work to do—and you possibly have more questions. I certainly did.

A Detour to Easy Street

There are some great options, such as using SQL Server for Linux in another Docker container or targeting a SQL Azure database. I'll be digging into those solutions in the next couple of articles, but first I want you to see a quick solution where the database server will indeed exist inside of the container and your API will run successfully. You can achieve this easily with SQLite, which is a self-contained database.

You should already have the Microsoft.EntityFramework.SQLite package installed. This NuGet package's dependencies will force the SQLite runtime components to install in the image where the app is built.

Add a new connection string called MagsConnectionSqlite to the appsettings.json file. I've specified the file name as DP0419Mags.db:

```
"ConnectionStrings": {
    "MagsConnectionMssql":
      "Server=(localdb)\\mssqllocaldb;Database=DP0419Mags;Trusted_Connection=True;",
    "MagsConnectionSqlite": "Filename=DP0419Mags.db;"
  }
```

In Startup, change the DbContext provider to SQLite with the new connection string name:

```
services.AddDbContext<MagContext>(options =>
  options.UseSqlite(Configuration.GetConnectionString(
  "MagsConnectionSqlite")));
```

The migration file you created is specific to the SQL Server provider, so you'll need to replace that. Delete the entire Migrations folder, then run *Add-Migration initSqlite* in the Package Manager Console to recreate the folder along with the migration and snapshot files.

You can run this against the built-in server if you want to see the file that gets created, or you can just start debugging this in Docker. The new SQLite database gets created very quickly when the Migrate command is called and the browser then displays the three magazines again. Note that the IP address of the URL will match the one you saw earlier when running the values controller in Docker. In my case, that's http://172.26.137.194/api/Magazines. So now the API and SQLite are both running inside the Docker container.

A More Production-Friendly Solution, Coming Soon

While using the SQLite database certainly simplifies the task of letting EF Core create a database inside the same container that's running the app, this is most likely not how you'd want to deploy your API into production. One of the beauties of containers is that you can express separation of concerns by employing and coordinating multiple containers.

In the case of this tiny solution, perhaps SQLite would do the job. But for your real-world solutions, you should leverage other options. Focusing on SQL Server, one of those options would be to target an Azure SQL Database. With this option, regardless of where you're running the app (on your development machine, in IIS, in a Docker container, in a Docker container in the cloud), you can be sure to always be pointing to a consistent database server or database depending on your requirements. Another path is to leverage containerized database servers, such as SQL Server for Linux, as I've written about in an earlier column (msdn.com/magazine/mt784660). Microservices introduces another layer of possible solutions given that the guidance is to have one database per microservice. You could easily manage those in containers, as well. There's a great (and free) book from Microsoft on architecting .NET apps for containerized microservices at bit.ly/2NsfYBt.

In the next few columns, I'll explore some of these solutions as I show you how to target SQL Azure or a containerized SQL Server; manage connection strings and protect credentials using Docker environment variables; and enable EF Core to discover connection strings at design time, using migrations commands and at run time from within the Docker container. Even with my pre-existing experience with Docker and EF Core, I went through a lot of learning curves working out the details of these solutions and am looking forward to sharing them all with you. ■

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 experts for reviewing this article: *Glenn Condron, Steven Green, Mike Morton*

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

FRANK LA VIGNE

How Do Neural Networks Learn?

In my previous column ("A Closer Look at Neural Networks," msdn.com/magazine/mt833269), I explored the basic structure of neural networks and created one from scratch with Python. After reviewing the basic structures common to all neural networks, I created a sample framework for computing the weighted sums and output values. Neurons themselves are simple and perform basic mathematical functions to normalize their outputs between 1 and 0 or -1 and 1. They become powerful, however, when they're connected to each other. Neurons are arranged in layers in a neural network and each neuron passes on values to the next layer. Input values cascade forward through the network and affect the output in a process called forward propagation.

However, exactly how do neural networks learn? What is the process and what happens inside a neural network when it learns? In the previous column, the focus was on the forward propagation of values. For supervised learning scenarios, neural networks can leverage a process called backpropagation.

$$f(m,b) = \frac{1}{N}\sum_{i=1}^{n}(y_i - (mx_i + b))^2$$

Figure 1 **The Cost, or Loss, Function**

Backpropagation, Loss and Epochs

Recall that each neuron in a neural network takes in input values multiplied by a weight to represent the strength of that connection. Backpropagation discovers the correct weights that should be applied to nodes in a neural network by comparing the network's current outputs with the desired, or correct, outputs. The difference between the desired output and the current output is computed by the Loss, or Cost, function. In other words, the Loss function tells us how accurate our neural network is at making predictions for a given input.

The formula to calculate the Loss is in **Figure 1**. Don't be intimidated by the math, all it does is add up the squares of all the differences. Typically, weights and biases are initially set to random values, which often yield a high Loss value when starting to train a neural network.

The algorithm then adjusts each weight to minimize the difference between the computed value and the correct value. The term "backpropagation" comes from the fact that the algorithm goes back and adjusts the weights and biases after computing an answer. The smaller the Loss for a network, the more accurate it becomes. The learning process, then, can be quantified as minimizing the loss function's output. Each cycle of forward propagation and backpropagation correction to lower the Loss is called an epoch.

Simply put, backpropagation is about finding the best input weights and biases to get a more accurate output or "minimize the Loss." If you're thinking this sounds computationally expensive, it is. In fact, compute power was insufficient until relatively recently to make this process practical for wide use.

Gradient Descent, Learning Rate and Stochastic Gradient Descent

How are the weights adjusted in each epoch? Are they randomly adjusted or is there a process? This is where a lot of beginners start to get confused, as there are a lot of unfamiliar terms thrown around, like gradient descent and learning rate. However, it's really not that complicated when explained properly. The Loss function reduces all the complexity of a neural network down to a single number that indicates how far off the neural network's, answer is from the desired answer. Thinking of the neural network's output as a single number allows us to think about its performance in simple terms. The goal is to find the series of weights that results in the lowest loss value, or the minimum.

Plotting this on a graph, as in **Figure 2**, shows that the Loss function has its own curve and gradients that can be used as a guide to adjust the weights. The slope of the Loss function's curve serves

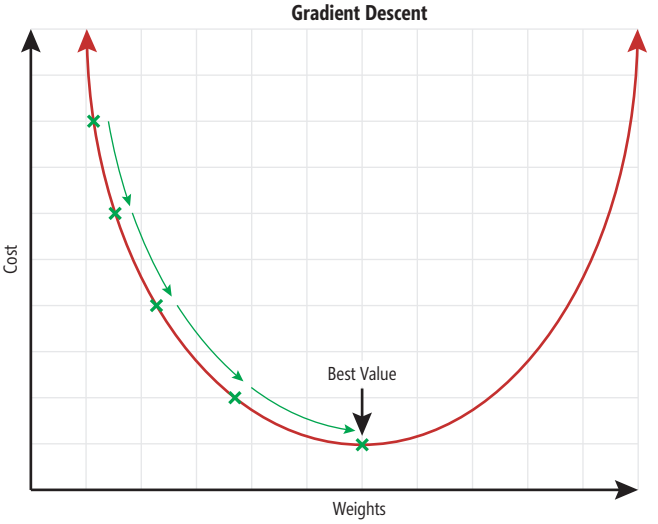


Figure 2 **Graph of the Loss Function with a Simple Curve**

as a guide and points to the minimum value. The goal is to locate the minimum across the entire curve, which represents the inputs where the neural network is most accurate.

In **Figure 2**, adding more to the weights reaches a low point and then starts to climb again. The slope of the line reveals the direction to that lowest point on the curve, which represents the lowest loss. When the slope is negative, add to the weights. When the slope is positive, subtract from the weights. The specific amount added or subtracted to the weights is known as the Learning Rate. Determining an ideal learning rate is as much an art as it is a science. Too large and the algorithm could overshoot the minimum. Too low and the training will take too long. This process is called Gradient Descent. Readers who are more familiar with the intricacies of calculus will see this process for what it is: determining the derivative of the Loss function.

As in any software engineering endeavor, knowing the fundamentals always helps when faced with challenges in the field.

Rarely, however, is the graph of a Loss function as simple as the one in **Figure 2**. In practice, there are many peaks and valleys. The challenge then becomes how to find the lowest of the low points (the global minimum) and not get fooled by low points nearby (local minima). The best approach in this situation is to pick a point along the curve at random and then proceed with the gradient descent process previously described, hence the term "Stochastic Gradient Descent." For a great explanation of the mathematical concepts on this process, watch the YouTube video, "Gradient Descent, How Neural Networks Learn | Deep Learning, Chapter 2," at youtu.be/IHZwWFHWa-w.

For the most part, this level of neural network architecture has been largely abstracted away by libraries such as Keras and TensorFlow. As in any software engineering endeavor, knowing the fundamentals always helps when faced with challenges in the field.

Putting Theory to Practice

In the previous column, I had created a neural network from scratch to process the MNIST digits. The resulting code base to bootstrap the problem was great at illustrating the inner workings of neural network architectures, but was impractical to bring forward. There exist so many frameworks and libraries now that perform the same task with less code.

To get started, open a new Jupyter notebook and enter the following into a blank cell and execute it to import all the required libraries:

```
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import to_categorical
import matplotlib.pyplot as plt
```

Note that the output from this cell states that Keras is using a TensorFlow back end. Because the MNIST neural network example is so common, Keras includes it as part of its API, and even splits the data into a training set and a test set. Write the following code into a new cell and execute it to download the data and read it into the appropriate variables:

```
# import the data
from keras.datasets import mnist
# read the data
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

Once the output indicates that the files are downloaded, use the following code to briefly examine the training and test dataset:

```
print(X_train.shape)
print(X_test.shape)
```

The output should read that the x_train dataset has 60,000 items and the x_test dataset has 10,000 items. Both consist of a 28x28 matrix of pixels. To see a particular image from the MNIST data, use MatPlotLib to render an image with the following code:

```
plt.imshow(X_train[10])
```

The output should look like a handwritten "3." To see what's inside the testing dataset, enter the following code:

```
plt.imshow(X_test[10])
```

The output shows a zero. Feel free to experiment by changing the index number and the dataset to explore the image datasets.

Shaping the Data

As with any AI or data science project, the input data must be reshaped to fit the needs of the algorithms. The image data needs to be flattened into a one-dimensional vector. As each image is 28x28 pixels, the one-dimensional vector will be 1 by (28x28), or 1 by 784. Enter the following code into a new cell and execute (note that this will not produce output text):

```
num_pixels = X_train.shape[1] * X_train.shape[2]

X_train = X_train.reshape(X_train.shape[0], num_pixels).astype('float32')
X_test = X_test.reshape(X_test.shape[0], num_pixels).astype('float32')
```

Determining an ideal learning rate is as much an art as it is a science. Too large and the algorithm could overshoot the minimum. Too low and the training will take too long.

Pixel values range from zero to 255. In order to use them, you'll need to normalize them to values between zero and one. Use the following code to do that:

```
X_train = X_train / 255
X_test = X_test / 255
```

Then enter the following code to take a look at what the data looks like now:

```
X_train[0]
```

The output reveals an array of 784 values between zero and one.

The task of taking in various images of handwritten digits and determining what number they represent is classification. Before

building the model, you'll need to split the target variables into categories. In this case, you know that there are 10, but you can use the to_categorical function in Keras to determine that automatically. Enter the following code and execute it (the output should display 10):

```
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

num_classes = y_test.shape[1]
print(num_classes)
```

Build, Train and Test the Neural Network

Now that the data has been shaped and prepared, it's time to build out the neural networks using Keras. Enter the following code to create a function that creates a sequential neural network with three layers with an input layer of num_pixels (or 784) neurons:

```
def classification_model():
  model = Sequential()
  model.add(Dense(num_pixels, activation='relu', input_shape=(num_pixels,)))
  model.add(Dense(100, activation='relu'))
  model.add(Dense(num_classes, activation='softmax'))
  model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
  return model
```

Compare this code to the code from my last column's "from scratch" methods. You may notice new terms like "relu" or "softmax" referenced in the activation functions. Up until now, I've only explored the Sigmoid activation function, but there are several kinds of activation functions. For now, keep in mind that all activation functions compress an input value by outputting a value between 0 and 1 or -1 and 1.

> As with any AI or data science project, the input data must be reshaped to fit the needs of the algorithms.

With all of the infrastructure in place, it's time to build, train and score the model. Enter the following code into a blank cell and execute it:

```
model = classification_model()
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, verbose=2)
scores = model.evaluate(X_test, y_test, verbose=0)
```

As the neural network runs, note that the loss value drops with each iteration. Accordingly, the accuracy also improves. Also, take note of how long each epoch takes to execute. Once completed, enter the following code to see the accuracy and error percentages:

```
print('Model Accuracy: {} \n Error: {}'.format(scores[1], 1 - scores[1]))
```

The output reveals an accuracy of greater than 98 percent and an error of 1.97 percent.

Persisting the Model

Now that the model has been trained to a high degree of accuracy, you can save the model for future use to avoid having to train it again. Fortunately, Keras makes this easy. Enter the following code into a new cell and execute it:

```
model.save('MNIST_classification_model.h5')
```

This creates a binary file that's about 8KB in size and contains the optimum values for weights and biases. Loading the model is also easy with Keras, like so:

```
from keras.models import load_model
pretrained_model = load_model('MNIST_classification_model.h5')
```

> Referencing the predefined model doesn't require the computationally expensive process of training and, in the final production system, the neural network can be implemented rapidly.

This h5 file contains the model and can be deployed along with code to reshape and prepare the input image data. In other words, the lengthy process of training a model needs only to be done once. Referencing the predefined model doen't require the computationally expensive process of training and, in the final production system, the neural network can be implemented rapidly.

Wrapping Up

Neural networks can solve problems that have confounded traditional algorithms for decades. As we've seen, their simple structure hides their true complexity. Neural networks work by propagating forward inputs, weights and biases. However, it's the reverse process of backpropagation where the network actually learns by determining the exact changes to make to weights and biases to produce an accurate result.

Learning, in the machine sense, is about minimizing the difference between the actual result and the correct result. This process is tedious and compute-expensive, as evidenced by the time it takes to run through one epoch. Fortunately, this training needs only to be done once and not each time the model is needed. Additionally, I explored using Keras to build out this neural network. While it is possible to write the code needed to build out neural networks from scratch, it's far simpler to use existing libraries like Keras, which take care of the minute details for you. ■

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, "Franks World TV" (FranksWorld.TV).*

Thanks to the following technical experts for reviewing this article:
Andy Leonard

What's New in Xamarin.Forms 4.0

Alessandro Del Sole

During the Connect(); conference in November 2018, Microsoft unveiled the first public preview of Xamarin.Forms 4.0, the latest version of the popular development technology that makes it simple to create shared UIs across different OSes, including Android, iOS and Windows 10. Updated previews have followed, giving developers a closer look at the progress of this major release. The most important new features that make Xamarin.Forms 4.0 amazing can be summarized as follows.

Shell is a container for applications that provides most of the common features the UI needs. It represents a single place to describe the visual structure of an application and includes a common UI for page navigation; a navigation service with deep linking; and an integrated search handler. For now, the Shell will be available only for iOS and Android.

CollectionView is a completely new control created to work with lists of data in a more efficient and performant way. It's also the foundation for the updated version of the **CarouselView** control.

Visual is the name of a new property that some views can use to render controls based on a given design, making it easy to create consistent UIs across platforms.

You'll find a thorough discussion of the Shell feature in the Connect(); Special Issue (msdn.com/ magazine/mt848639), so I won't cover it here. Instead, I'll focus on CollectionView, CarouselView, Visual and other miscellaneous improvements to the existing code base. I'll also take a quick look at some new features included in Visual Studio 2019 Preview 2 for Xamarin.Forms. Keep in mind that the bits described in this article are still in preview and may be subject to change.

The technologies presented in this article are in preview; all information is subject to change.

This article discusses:

- The new CollectionView and updated CarouselView controls
- The new Visual property
- Miscellaneous improvements to the existing code base
- New features in pre-releases of Visual Studio 2019 for Xamarin.Forms

Technologies discussed:

Xamarin.Forms 4.0, XAML, Visual Studio 2019 Preview 2

Code download available at:

msdn.com/magazine/0419magcode

Creating and Setting Up a Project

To work with the current preview of Xamarin.Forms 4.0, you first need to create a Mobile App (Xamarin.Forms) project in Visual Studio 2017, as you normally would. Select the iOS and Android platforms and the .NET Standard code sharing strategy. When the new project is ready, with the NuGet package manager make sure you upgrade the Xamarin.Forms library to the latest pre-release available (at the time of this writing, it's called 4.0.0.94569-pre3). When you install this preview, NuGet will also add the Xamarin.iOS.MaterialComponents package to the iOS platform project, as I'll discussed later when talking about the Visual property.

The last step consists of enabling all the experimental features in the Xamarin.Forms 4.0 preview inside the MainActivity.cs file for Android and the AppDelegate.cs file for iOS. In both files, add the following line of code before the invocation of the Xamarin.Forms.Forms.Init method:

```
global::Xamarin.Forms.Forms.SetFlags("Shell_Experimental", "Visual_Experimental",
  "CollectionView_Experimental");
```

You should add the Shell_Experimental flag if you want to experiment with the Shell feature; otherwise, it's optional for the next topics.

Working with Data: The CollectionView Control

Xamarin.Forms 4.0 introduces a brand-new control called CollectionView, which you can use to display and edit lists of data. From an architectural point of view, CollectionView has a couple of major benefits. First, unlike with ListView, the data templates no longer rely on Cell views, which simplifies the visual tree. As a consequence, and as the second benefit, you code CollectionView objects using the same familiar approach based on XAML, but with much simpler code. Moreover, the CollectionView control makes it simpler to manage the layout of your lists by supporting horizontal layouts and grid views. Let's start by creating some sample data that will be displayed inside a CollectionView control. In the .NET Standard project, add the class shown in **Figure 1**, which represents the simplified definition of a product.

Notice how the Product class implements the INotifyPropertyChanged interface to support change notification if you decide to make a product instance editable. The next step is defining a view model that exposes the product information, which will be data-bound to the UI. This is shown in **Figure 2**.

Figure 1 **A Simplified Implementation of a Product Class**

```
public class Product : INotifyPropertyChanged
{
  public event PropertyChangedEventHandler PropertyChanged;

  private void OnPropertyChanged([CallerMemberName] string propertyName = null)
  {
    PropertyChanged?.Invoke(this,
      new PropertyChangedEventArgs(propertyName));
  }

  private int _productQuantity;
  private string _productName;
  private string _productImage;

  public int ProductQuantity
  {
    get
    {
      return _productQuantity;
    }
    set
    {
      _productQuantity = value;
      OnPropertyChanged();
    }
  }

  public string ProductName
  {
    get
    {
      return _productName;
    }
    set
    {
      _productName = value;
      OnPropertyChanged();
    }
  }

  public string ProductImage
  {
    get
    {
      return _productImage;
    }
    set
    {
      _productImage = value;
      OnPropertyChanged();
    }
  }
}
```

> Declaring CollectionView and assigning its ItemTemplate property with a DataTemplate object in XAML works the same as with the ListView control, with the important difference being that you don't have to deal with Cell views anymore.

In a real-world scenario, data could come from a database or from a Web service, but in this case I create a few products directly in the code for demonstration purposes. The image files used in **Figure 2** can be found in the companion code for the article. In the MainPage.xaml file of your project, you can then declare a CollectionView control to display the data. The key points are assigning to the ItemsSource property the collection you want to display, and defining a data template for each item in the list. **Figure 3** provides an example.

Declaring CollectionView and assigning its ItemTemplate property with a DataTemplate object in XAML works the same as with

Figure 2 **The ProductViewModel Class Exposes Data Information to the UI**

```
public class ProductViewModel
{
  public ObservableCollection<Product> Products { get; set; }

  public ProductViewModel()
  {
    // Sample products
    this.Products = new ObservableCollection<Product>();
    this.Products.Add(new Product { ProductQuantity = 50, ProductName = "Cheese",
                                    ProductImage = "Cheese.png" });
    this.Products.Add(new Product { ProductQuantity = 10, ProductName = "Water",
                                    ProductImage = "Water.png" });
    this.Products.Add(new Product { ProductQuantity = 6, ProductName = "Bread",
                                    ProductImage = "Bread.png" });
    this.Products.Add(new Product { ProductQuantity = 40, ProductName = "Tomatoes",
                                    ProductImage = "Tomato.png" });
  }
}
```

Figure 3 **Declaring a CollectionView to Display Data**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:XF4_WhatsNew"
             x:Class="XF4_WhatsNew.MainPage">

  <StackLayout>
    <CollectionView x:Name="ProductList" ItemsSource="{Binding Products}">
      <CollectionView.ItemTemplate>
        <DataTemplate>
          <Grid>
            <Grid.ColumnDefinitions>
              <ColumnDefinition Width="24"/>
              <ColumnDefinition />
            </Grid.ColumnDefinitions>

            <Image Source="{Binding ProductImage}" HeightRequest="22" />
            <StackLayout Grid.Column="1" Orientation="Vertical" Spacing="5">
              <Label Text="{Binding ProductName}" FontSize="Medium"
                TextColor="Blue"/>
              <Label Text="{Binding ProductQuantity}" FontSize="Small"
                TextColor="Black"/>
            </StackLayout>
          </Grid>
        </DataTemplate>
      </CollectionView.ItemTemplate>
    </CollectionView>
  </StackLayout>
</ContentPage>
```

the ListView control, with the important difference being that you don't have to deal with Cell views anymore. You then create an instance of the ProductViewModel class and assign it to the page's BindingContext property so that data binding can work, like in the following code to be written in the MainPage.xaml.cs file:

```
public partial class MainPage : ContentPage
{

  private ProductViewModel ViewModel { get; set; }

  public MainPage()
  {
    InitializeComponent();

    this.ViewModel = new ProductViewModel();
    this.BindingContext = this.ViewModel;
  }
}
```

If you run this code, you'll see a result like that in **Figure 4**.

As you'll see, CollectionView provides other interesting properties for implementing different layouts and handling item selection.

> If you have experience with the ListView control in Xamarin.Forms, you know that there's no built-in option to set the orientation as horizontal.

Orientation and Layout If you have experience with the ListView control in Xamarin.Forms, you know that there's no built-in option to set the orientation as horizontal. This can certainly be accomplished using custom renderers, but it increases your code complexity. Similarly, there's no built-in option to change the layout from a list view to a grid view. The CollectionView control, however, makes setting different layout options extremely simple via the ItemsLayout property, which can be assigned with properties from the ListItemsLayout class, such as VerticalList and HorizontalList. By default, the ItemsLayout property is assigned with ListItemsLayout.VerticalList, so you don't need to assign it explicitly for a vertical layout. But to implement a horizontal orientation, add the following code snippet to your XAML:

```
<CollectionView ItemsSource="{Binding
  Products}"
  ItemsLayout="{x:Static
    ListItemsLayout.HorizontalList}">
  ...
</CollectionView>
```

Actually, the ItemsLayout property also allows you to define a grid view. In this case, you use the GridItemsLayout class instead of

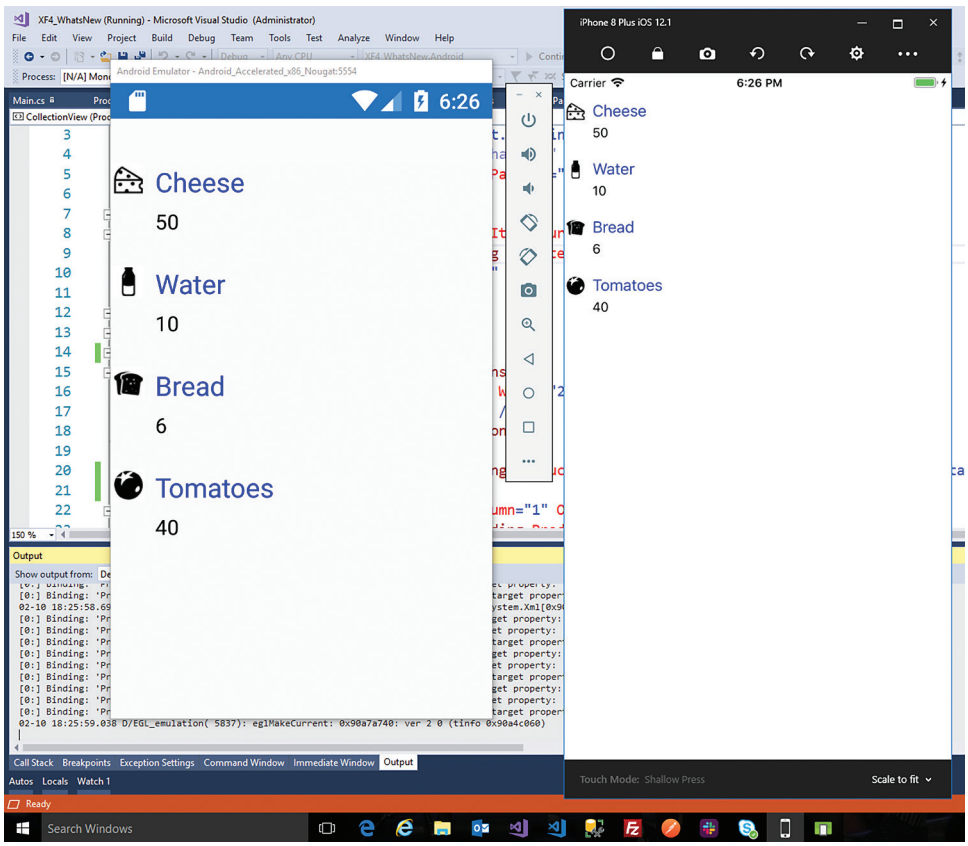


Figure 4 **Displaying Lists of Data with CollectionView**

ListItemsLayout, as in the following example that demonstrates how to create a horizontal grid view:

```
<CollectionView x:Name="ProductList" ItemsSource="{Binding Products}" >
  <CollectionView.ItemsLayout>
    <GridItemsLayout Orientation="Horizontal" Span="3" />
  </CollectionView.ItemsLayout>
    ...
</CollectionView>
```

The CollectionView control allows for selecting single or multiple items, and for disabling selection.

By default, the GridItemsLayout shows items in a single row with the horizontal layout, so you can set the Span property to decide how many rows the grid will contain. As items are added to the data-bound list, the grid view will grow horizontally. You can get a vertical grid view with the following code:

```
<CollectionView x:Name="ProductList" ItemsSource="{Binding Products}" >
  <CollectionView.ItemsLayout>
    <GridItemsLayout Orientation="Vertical" Span="2" />
  </CollectionView.ItemsLayout>
    ...
</CollectionView>
```

Similarly, a vertical grid will display items in a single column by default, so you can change the Span properties and specify the number of columns for the grid.

Managing Item Selection The CollectionView control allows for selecting single or multiple items, and for disabling selection. The SelectionMode property can be assigned values from the Xamarin.Forms.SelectionMode enumeration: None, Single and Multiple. With None, you can disable item selection very easily. The SelectedItem property, of type object, represents one or more selected items in the list. The SelectionChanged event is then raised when the user selects an item in the CollectionView. Such an event will only work if SelectionMode has been assigned with either Single (which is also the default if SelectionMode isn't specified) or Multiple. So, in your XAML you might have the following assignments:

```
<CollectionView x:Name="ProductList" ItemsSource="{Binding Products}"
  SelectionMode="Single" SelectionChanged="ProductList_SelectionChanged">
...
</CollectionView>
```

Then the SelectionChanged event handler allows you to manage item selection via a parameter of type Xamarin.Forms.SelectionChangedEventArgs. This object exposes two useful properties, CurrentSelection and PreviousSelection. As the names imply, the first property returns the currently selected objects while the second property returns objects that were selected previously. This is very useful, because in some situations you might need to know what object instances were actually selected before the new selection. Both properties are of type IReadOnlyList<object> and expose the list of selected items. If SelectionMode is set as Single, you can retrieve the instance of the selected item as follows:

```
private void ProductList_SelectionChanged(object sender,
SelectionChangedEventArgs e)
{
  var selectedProduct = this.ProductList.SelectedItem as Product;

  var singleProduct = e.CurrentSelection.FirstOrDefault() as Product;
}
```

Whether you use the CollectionView.SelectedItem property or the first item in the collection, you need to cast the resulting object into the expected type, Product in this case. If SelectionMode is set

Figure 5 **Extending the View Model with MVVM Support**

```
public class ProductViewModel: INotifyPropertyChanged
{
  public ObservableCollection<Product> Products { get; set; }

  private Product _selectedProduct;
  public Product SelectedProduct
  {
    get
    {
      return _selectedProduct;
    }
    set
    {
      _selectedProduct = value;
      OnPropertyChanged();
    }
  }

  private Command _productSelectedCommand;

  public Command ProductSelectedCommand
  {
    get
    {
      return _productSelectedCommand;
    }
      set
      {
        _productSelectedCommand = value;
        OnPropertyChanged();
      }
    }

  public ProductViewModel()
  {
    // Sample products
    this.Products = new ObservableCollection<Product>();
    this.Products.Add(new Product { ProductQuantity = 50,
      ProductName = "Cheese", ProductImage = "Cheese.png" });
    this.Products.Add(new Product { ProductQuantity = 10,
      ProductName = "Water", ProductImage = "Water.png" });
    this.Products.Add(new Product { ProductQuantity = 6,
      ProductName = "Bread", ProductImage = "Bread.png" });
    this.Products.Add(new Product { ProductQuantity = 40,
      ProductName = "Tomatoes", ProductImage = "Tomato.png" });

    this.ProductSelectedCommand =
      new Command(ExecuteProductSelectedCommand,
        CanExecuteProductSelectedCommand);
  }

  private bool CanExecuteProductSelectedCommand(object arg)
  {
    return this.SelectedProduct != null;
  }

  private void ExecuteProductSelectedCommand(object obj)
  {
    // Handle your object here....
    var currentProduct = this.SelectedProduct;
  }

  public event PropertyChangedEventHandler PropertyChanged;
  private void OnPropertyChanged([CallerMemberName] string propertyName = null)
  {
    PropertyChanged?.Invoke(this,
      new PropertyChangedEventArgs(propertyName));
  }
}
```

as Multiple, you first need to cast the read-only collection of selected items into a list of objects of the expected type, as in the following:

```
private void ProductList_SelectionChanged(object sender,
  SelectionChangedEventArgs e)
{
  var selectedItems = e.CurrentSelection.Cast<Product>();

  foreach(var product in selectedItems)
  {
    // Handle your object properties here...
  }
}
```

The same approach would work for the SelectionChangedEventArgs.PreviousSelection property for both single- and multiple-item selection. As a modern view, CollectionView also provides support for the Model-View-ViewModel (MVVM) pattern when handling item selection, so you can use the SelectionChangedCommand property, which can be bound to an instance of the Command class in your view model; and the SelectionChangedCommandParameter, which allows for passing a parameter to the command. For example, let's extend the ProductViewModel class in order to add a SelectedProduct property that will be data-bound to the CollectionView's SelectedItem property, and with a new command for selection handling. For the sake of simplicity, the property of type Command will use neither the generic implementation nor the command parameter. **Figure 5** shows what the view model looks like.

Then, in your XAML, assign the selection properties as follows:

```
<CollectionView x:Name="ProductList" ItemsSource="{Binding Products}"
                SelectedItem="{Binding SelectedProduct, Mode=TwoWay}"
                SelectionChangedCommand="{Binding ProductSelectionCommand}"
                SelectionMode="Single">
</CollectionView>
```

With this approach, you can work against your selected objects in the view model, which is the place where you implement the business logic, keeping the work on the data decoupled from the views.

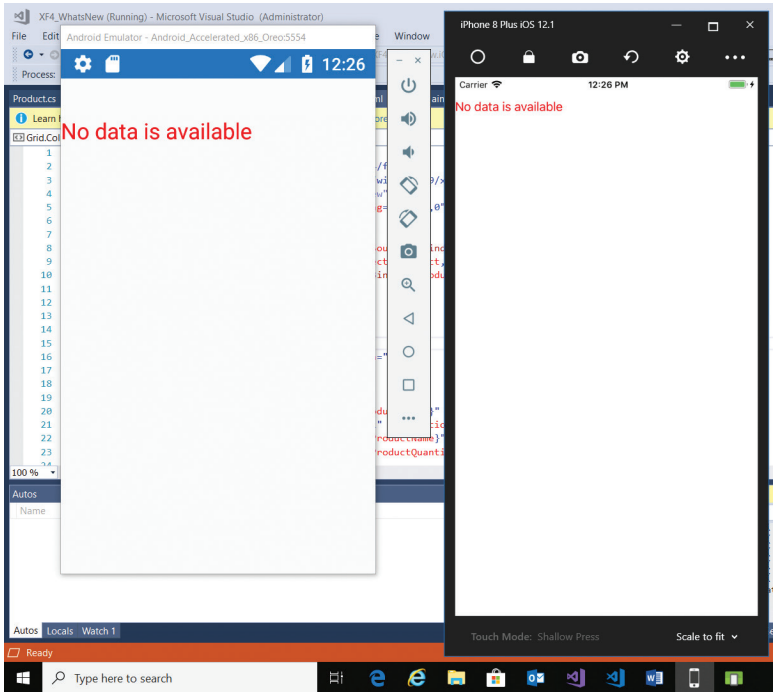


Figure 6 **Providing a View for Empty States**

Managing Empty Lists One of the most important features offered by CollectionView, and definitely one of my favorites, is the built-in option to display a different view if the data-bound list is empty, at no cost. This is possible via the EmptyView property, which allows for specifying the view that will be displayed in the case of empty data. It works like this:

```
<CollectionView>
...
  <CollectionView.EmptyView>
    <Label Text="No data is available" TextColor="Red"
      FontSize="Medium"/>
  </CollectionView.EmptyView>
</CollectionView>
```

This code produces the result shown in **Figure 6**. With this approach, you don't need to create a data template for empty states or to implement data template selectors.

As an alternative, you can use the EmptyViewTemplate property if you want to implement a custom view. For instance, the following code demonstrates how to display both an image and a text message with empty lists:

```
<CollectionView.EmptyViewTemplate>
  <DataTemplate>
    <StackLayout Orientation="Vertical" Spacing="20">
      <Image Source="EmptyList.png" Aspect="Fill"/>
      <Label Text="No data is available" TextColor="Red"/>
    </StackLayout>
  </DataTemplate>
</CollectionView.EmptyViewTemplate>
```

Scrolling CollectionView also exposes the ScrollTo method, which you can use to programmatically scroll the list to a specified item. The following scrolls to the third item in the list:

```
ProductList.ScrollTo(2);
```

Simpler API Surface The CollectionView control exposes a simpler API surface than ListView. For instance, CollectionView has no properties to work with separators (such as SeparatorVisibility and SeparatorColor), so you can implement your own separators with views like the BoxView in the data template. CollectionView doesn't implement a property like ListView.RowHeight, because now an item's height is determined by the first item in the list. The HasUnevenRows property in ListView has a counterpart in CollectionView called ItemSizingStrategy. With regard to pull-to-refresh techniques, properties such as IsPullToRefreshEnabled, IsRefreshing and RefreshCommand, which are exposed by ListView, aren't available in the CollectionView control. However, Microsoft is working on implementing properties that support views for refresh states. The Xamarin.Forms.CollectionView Spec document on GitHub (bit.ly/2N6iw8a) provides a detailed explanation about refresh options and also about other API implementations, including architectural decisions.

Introducing CarouselView

The CarouselView control isn't new in Xamarin.Forms 4.0. However, it has been completely rebuilt based on the CollectionView structure and performance.

With CarouselView, you can show one object in the list at a time, in a card format. **Figure 7**, taken from the official Xamarin blog (bit.ly/2BwWMNV), shows how CarouselView

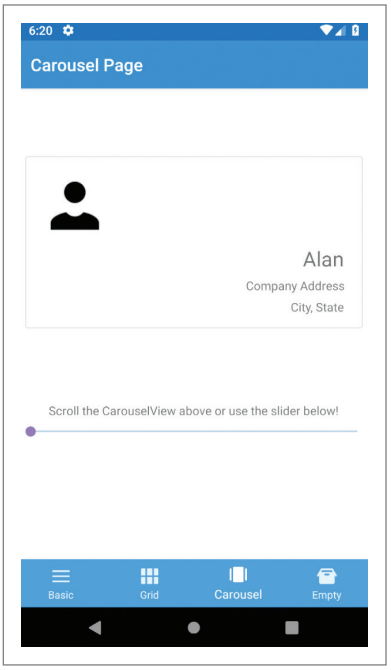


Figure 7 **The CarouselView Shows Items in the Form of Cards, Enabling the Swipe Gesture**

provides a way to swipe between items.

With regard to the sample code described previously, you could use the CarouselView to display the list of products as cards, as shown in **Figure 8**.

Let's focus on the key points in the code.

Declaring CarouselView Declaring CarouselView in XAML is very similar to how you declared CollectionView. You still need to bind the ItemsSource property with the collection of data you want to display. Notice how you can also provide views for empty states, exactly as you did with CollectionView.

Orientation CarouselView supports both horizontal and vertical orientation. You assign the ItemsLayout property with an instance of GridItemsLayout, passing either Horizontal or Vertical to the Orientation property.

Adding Snap Points With the SnapPointsAlignment and SnapPointsType properties on the ItemsLayout, you can configure the behavior of the card. For example, in **Figure 8** the SnapPointsType and SnapPointsAlignment properties make sure the card appears centered. Without those assignments, the card would stop part way between the other cards. Allowed values for SnapPointsAlignment are Center, End and Start. Allowed values for SnapPointsType are Mandatory, MandatorySingle and None.

As with CollectionView, keep in mind that the final bits might provide a different API implementation. With the current preview, you might also experience an InvalidCastException when the app is rendering the view. This is an unstable build, so such problems are to be expected.

Creating Consistent UIs with the Visual Property

Xamarin.Forms 4.0 introduces a new way for developers to create UIs that look largely identical on both Android and iOS with almost no effort. The key is a new property called Visual, exposed by the VisualElement class. The Visual property makes sure that new, additional renderers are used to draw visual elements instead of the default renderer. In its current state of preview, Xamarin.Forms 4.0 includes an experimental rendering mechanism based on material design, currently available for the following views: Button, Entry, Frame and ProgressBar. You can enable the new visual rendering like this:

```
<Entry Visual="Material" />
```

Figure 8 **Displaying Items as Cards with CarouselView**

```
<CarouselView x:Name="ProductList" ItemsSource="{Binding Products}">
  <CarouselView.ItemsLayout>
    <GridItemsLayout Orientation="Horizontal"
                     SnapPointsAlignment="Center"
                     SnapPointsType="Mandatory"/>
  </CarouselView.ItemsLayout>

  <CarouselView.ItemTemplate>
    <DataTemplate>
      <Grid>
        <Grid.ColumnDefinitions>
          <ColumnDefinition Width="24"/>
          <ColumnDefinition />
        </Grid.ColumnDefinitions>

        <Image Source="{Binding ProductImage}" HeightRequest="22" />
        <StackLayout Grid.Column="1" Orientation="Vertical"
                     Spacing="5">
          <Label Text="{Binding ProductName}" FontSize="Medium"
                 TextColor="Blue"/>
          <Label Text="{Binding ProductQuantity}" FontSize="Small"
                 TextColor="Black"/>
        </StackLayout>
      </Grid>
    </DataTemplate>
  </CarouselView.ItemTemplate>
  <CarouselView.EmptyView>
    <Label Text="No data is available" TextColor="Red"
           FontSize="Medium"/>
  </CarouselView.EmptyView>
</CarouselView>
```

Figure 9, taken from the Microsoft release notes for Xamarin.Forms 4.0 (bit.ly/2BB4MxA), shows an example of a consistent UI across Android and iOS, created using Visual.

You could also assign the Visual property at the ContentPage level to make sure that any of the supported views in your code will use the specified design. In order to use Visual, your project must satisfy a few requirements:

- In the iOS project, the Xamarin.iOS.MaterialComponents NuGet package must be installed. However, this is automatically installed when you upgrade your Xamarin.Forms package to 4.0.

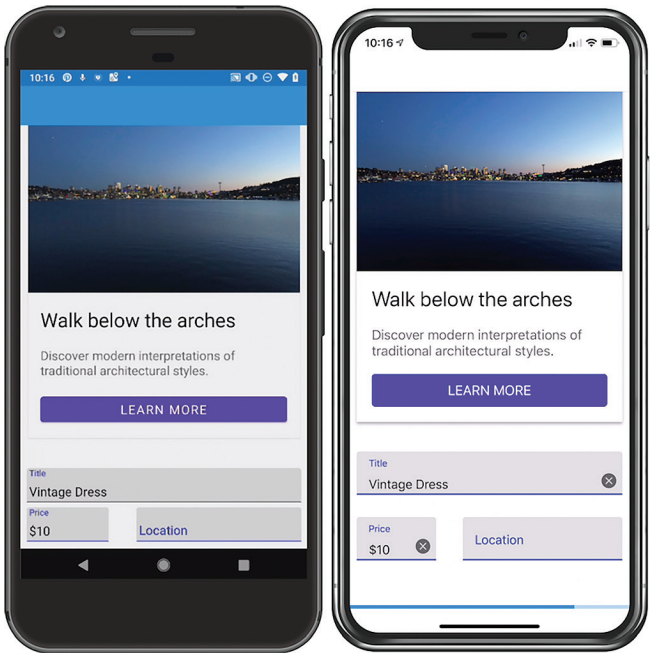


Figure 9 **Consistent UI with the Visual Property**

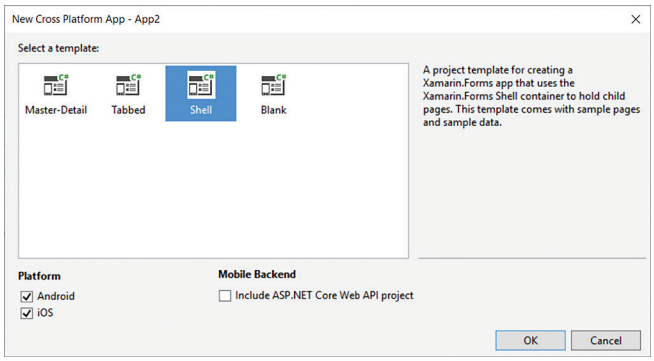


Figure 10 **The New Project Template Based on the Shell**

- For Android, your project must be based on API 29 and the support libraries must use version 28. Additionally, the app theme must inherit from one of the Material Components themes.

It's worth noting that the development of Visual has really just begun, so developers can reasonably expect many updates. In general, it's an extremely useful addition that will save you lots of time when you need to create UIs that are identical across platforms.

Miscellaneous Improvements

In Xamarin.Forms 4.0, you can now specify a color for the spinner when pull-to-refresh is enabled. This is accomplished by assigning the RefreshControlColor property as follows:

```
<ListView RefreshControlColor="Green"/>
```

Additionally, in ListView it's possible to hide the scrollbars without writing custom renderers, by simply assigning the HorizontalScrollBarVisibility and VerticalScrollBarVisibility properties:

```
<ListView HorizontalScrollBarVisibility="Always"
  VerticalScrollBarVisibility="Never"/>
```

Possible values are Always (always visible), Never (never visible) and Default (based on the target platform's default). Also, the Editor control now includes support for text prediction, as in the Entry control. You can simply assign the IsTextPredictionEnabled property with True or False, like so:

```
<Editor IsTextPredictionEnabled="True" />
```

With Xamarin.Forms 4.0, the SwitchCell view receives the OnColor property, just as Switch view did in the previous major releases. So, you can now set a different color for the active state when using SwitchCell in your data templates:

```
<SwitchCell OnColor="Red"/>
```

These are the most interesting improvements from a practical point of view, but the full list of updates and issues fixed is available on the release notes page and will be eventually updated once Xamarin.Forms 4.0 is finally released to production.

A Quick Look at Xamarin.Forms in Visual Studio 2019

Microsoft recently released Visual Studio 2019 Preview 3 (bit.ly/2QcOK6z), which provides an early look at what's coming with the next major release of Visual Studio, including support for Xamarin.Forms 4.0. For example, a new project template called Shell is now available in the New Cross Platform App dialog (see **Figure 10**). This template generates a new project based on the Shell feature, with several views referenced in the Shell object in the main page's XAML code.

Visual Studio 2019 also brings to Xamarin.Forms a new Properties panel, shown in **Figure 11**, which was already available to other development platforms. It displays property values as soon as you click on a view in your XAML code or in the Xamarin.Forms Previewer.

If you change property values through the Properties panel, the XAML code is automatically updated to reflect your changes. Notice how specific editors are available for some types, such as the color editor for properties of type Xamarin.Forms.Color.

Wrapping Up

As with the previous major releases, Xamarin.Forms 4.0 demonstrates the huge investment Microsoft is making to improve the development experience not only with new features, but also by making the existing tools and code base more and more reliable, which is what developers need in their real-world projects. It will be interesting to see what further news is presented at the upcoming Microsoft Build conference, which will take place in Seattle May 6-8. ■

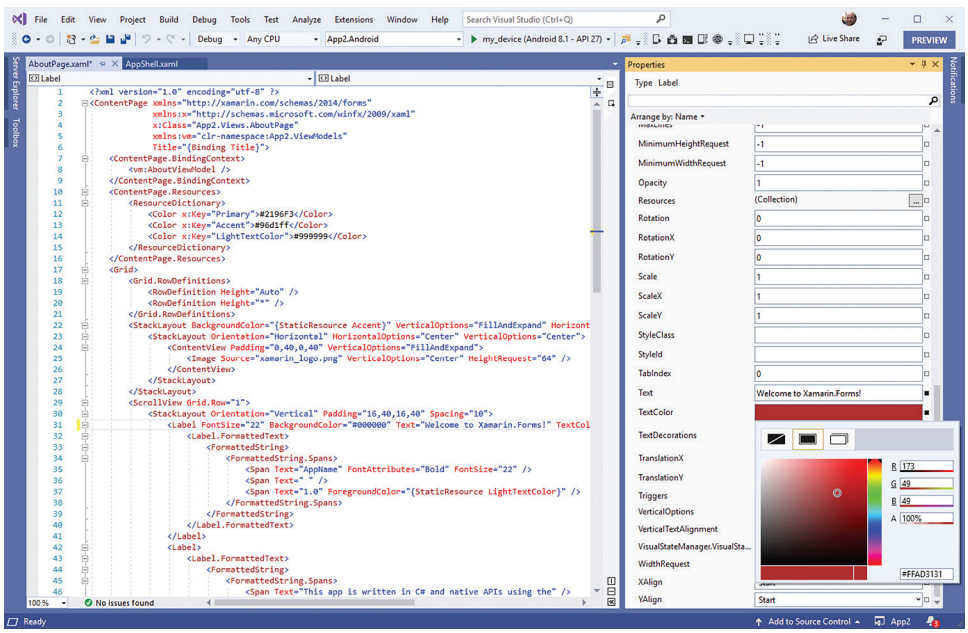


Figure 11 **The New Properties Panel**

Alessandro Del Sole *has been a Microsoft MVP since 2008 and a Xamarin Certified Developer. He has authored many books, eBooks, instructional videos and articles about .NET development with Visual Studio. Del Sole works for Fresenius Medical Care as a senior software engineer, focusing on building .NET and mobile apps with Xamarin in the health care market. You can follow him on Twitter: @progalex.*

Thanks to the following Microsoft technical expert for reviewing this article: *Paul DiPietro*

#VSLIVE

Visual Studio LIVE!
EXPERT SOLUTIONS FOR ENTERPRISE DEVELOPERS

MICROSOFT HQ

August 12-16, 2019
Microsoft Campus in Redmond, WA

INTENSE DEVELOPER TRAINING CONFERENCE

In-depth Technical Content On:

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

New This Year!

VSLive! Conference On-Demand

Get on-demand access for one full year to all keynotes and sessions from Visual Studio Live! Microsoft HQ, including everything Tuesday – Thursday at the conference. Learn more at VSLive.com/OnDemand

Save $400
When You Register By June 21!

vslive.com/microsofthq

SUPPORTED BY Progress

SUPPORTED BY Microsoft Visual Studio msdn magazine Visual Studio MAGAZINE

PRODUCED BY CONVERGE 360 A 1105 MEDIA company

Closed-Loop Intelligence: A Design Pattern for Machine Learning

Geoff Hulten

There are many great articles on using machine learning to build models and deploy them. These articles are similar to ones that teach programming techniques—they give valuable core skills in great detail. But to go beyond building toy examples you need another set of skills. In traditional systems these skills are called things like software engineering, software architecture or design patterns—approaches to organizing large software systems and the teams of people building them to achieve your desired impact.

This article introduces some of the things you'll need to think about when adding machine learning to your traditional software engineering process, including:

Connecting machine learning to users: What it means to close the loop between users and machine learning.

Picking the right objective: Knowing what part of your system to address with machine learning, and how to evolve this over time.

This article discusses:

- Practical machine learning
- Machine learning engineering
- Operating machine learning-based systems

Technologies discussed:

Intelligence Runtimes, Intelligence Management and Telemetry Systems, Intelligence Creation Environments

Implementing with machine learning: The systems you'll need to build to support a long-lived machine learning-based solution that you wouldn't need to build for a traditional system.

Operating machine learning systems: What to expect when running a machine learning-based system over time.

Of course, the first question is determining when you need to use machine learning. One key factor in the decision is how often you think you'll need to update an application before you have it right. If the number is small—for example, five or 10 times—then machine learning is probably not right. But if that number is large—say, every hour for as long as the system exists—then you might need machine learning.

There are four situations that clearly require a number of updates to get right:

- **Big Problems:** Some problems are big. They have so many variables and conditions that need to be addressed that they can't be completed in a single shot.
- **Open-Ended Problems:** Many problems lack a single, fixed solution, and require services that live and grow over long periods of time.
- **Time-Changing Problems:** If your domain changes in ways that are unpredictable, drastic or frequent, machine learning might be worth considering.
- **Intrinsically Hard Problems:** Tough problems like speech recognition and weather simulation and prediction can benefit from machine learning, but often only after years of effort

spent gathering training data, understanding the problems and developing intelligence.

If your problem has one of these properties, machine learning might be right. If not, you might be better off starting with a more traditional approach. If you can achieve your goal with a traditional approach, it will often be cheaper and simpler.

Connecting Machine Learning to Users

Closing the loop is about creating a virtuous cycle between the intelligence of a system and the usage of the system. As the intelligence gets better, users get more benefit from the system (and presumably use it more) and as more users use the system, they generate more data to make the intelligence better.

Consider a search engine. A user types a query and gets some answers. If she finds one useful, she clicks it and is happy. But the search engine is getting value from this interaction, too. When users click answers, the search engine gets to see which pages get clicked in response to which queries, and can use this information to adapt and improve. The more users who use the system, the more opportunities there are to improve.

But a successful closed loop doesn't happen by accident. In order to make one work you need to design a UX that shapes the interactions between your users and your intelligence, so they produce useful training data. Good interactions have the following properties:

The components of the interaction are clear and easy to connect. Good interactions make it possible to capture the context the user and application were in at the time of the interaction, the action the user took and the outcome of the interaction. For example, a book recommender must know what books the user owns and how much they like them (the context); what books were recommended to the user and if they bought any of them (the action); and if the user ended up happy with the purchase or not (the outcome).

The outcome should be implicit and direct. A good experience lets you interpret the outcome of interactions implicitly, by watching the user use your system naturally (instead of requiring them to provide ratings or feedback). Also, there won't be too much time or too many extraneous interactions between the user taking the action and getting the outcome.

Have no (or few) biases. A good experience will be conscious of how users experience the various possible outcomes and won't systematically or unconsciously drive users to under-report or over-report categories of outcomes. For example, every user will look at their inbox in an e-mail program, but many will never look at their junk folder. So the bad outcome of having a spam message in the inbox will be reported at a much higher rate than the bad outcome of having a legitimate message in the junk folder.

Does not contain feedback loops. A closed loop can suffer from feedback that compounds mistakes. For example, if the model makes a mistake that suppresses a popular action, users will stop selecting the action (because it's suppressed) and the model may learn that it was right to suppress the action (because people stopped using it). To address feedback loops, an experience should provide alternate ways to get to suppressed actions and consider a bit of randomization to model output.

These are some of the basics of connecting machine learning to users. Machine learning will almost always be more effective when the UX and the machine learning are designed to support one another. Doing this well can enable all sorts of systems that would be prohibitively expensive to build any other way.

Picking the Right Objective

One interesting property of systems built with machine learning is this: They perform worst on the day you ship them. Once you close the loop between users and models, your system will get better and better over time. That means you might want to start with an easy objective, and rebalance toward more difficult objectives as your system improves.

Imagine designing an autonomous car. You could work on this until you get it totally perfect, and then ship it. Or you could start with an easier sub-problem—say, forward collision avoidance. You could actually build the exact same car for forward collision avoidance that you would build for fully autonomous driving—all the controls, all the sensors, everything. But instead of setting an objective of full automation, which is extremely difficult, you set an objective of reducing collisions, which is more manageable.

Because avoiding collisions is valuable in its own right, some people will buy your car and use it—yielding data that you can leverage with machine learning to build better and better models. When you're ready, you can set a slightly harder objective, say lane following, which provides even more value to users and establishes a virtuous cycle as you ultimately work toward an autonomous vehicle.

This process might take months. It might take years. But it will almost certainly be cheaper than trying to build an autonomous car without a closed loop between users and your machine learning.

Closing the loop is about creating a virtuous cycle between the intelligence of a system and the usage of the system.

You can usually find ways to scale your objectives as your models get better. For instance, a spam filter that initially moves spam messages to a junk folder could later improve to delete spam messages outright. And a manufacturing defect detection system might flag objects for further inspection as a first objective, and later discard defective objects automatically as models improve.

It's important to set an objective that you can achieve with the models you can build today—and it's great when you can grow your machine learning process to achieve more and more interesting objectives over time.

Implementing with Machine Learning

Systems built to address big, open-ended, time-changing or intrinsically hard problems require many updates during their lifetimes.

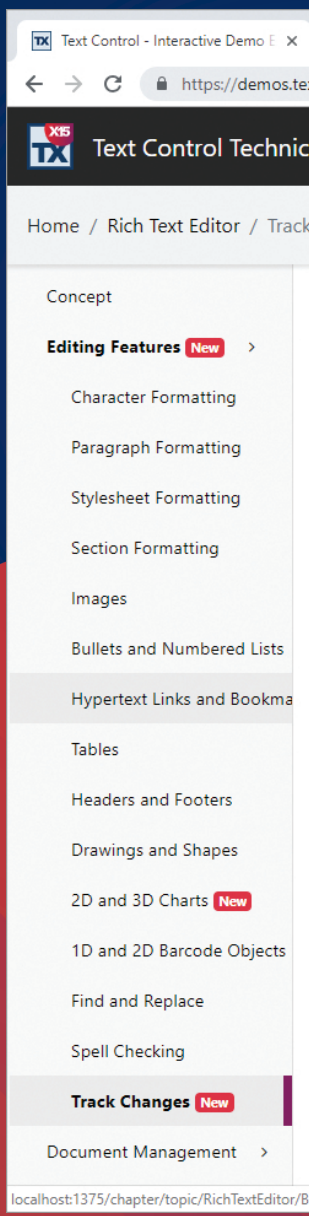
TEXT CONTROL

INTEGRATE DOCUMENT COLLABORATION

Integrate MS Word compatible track changes into cross-platform web applications. Share and review documents with a true WYSIWYG document editor.

See our technology live:

demos.textcontrol.com

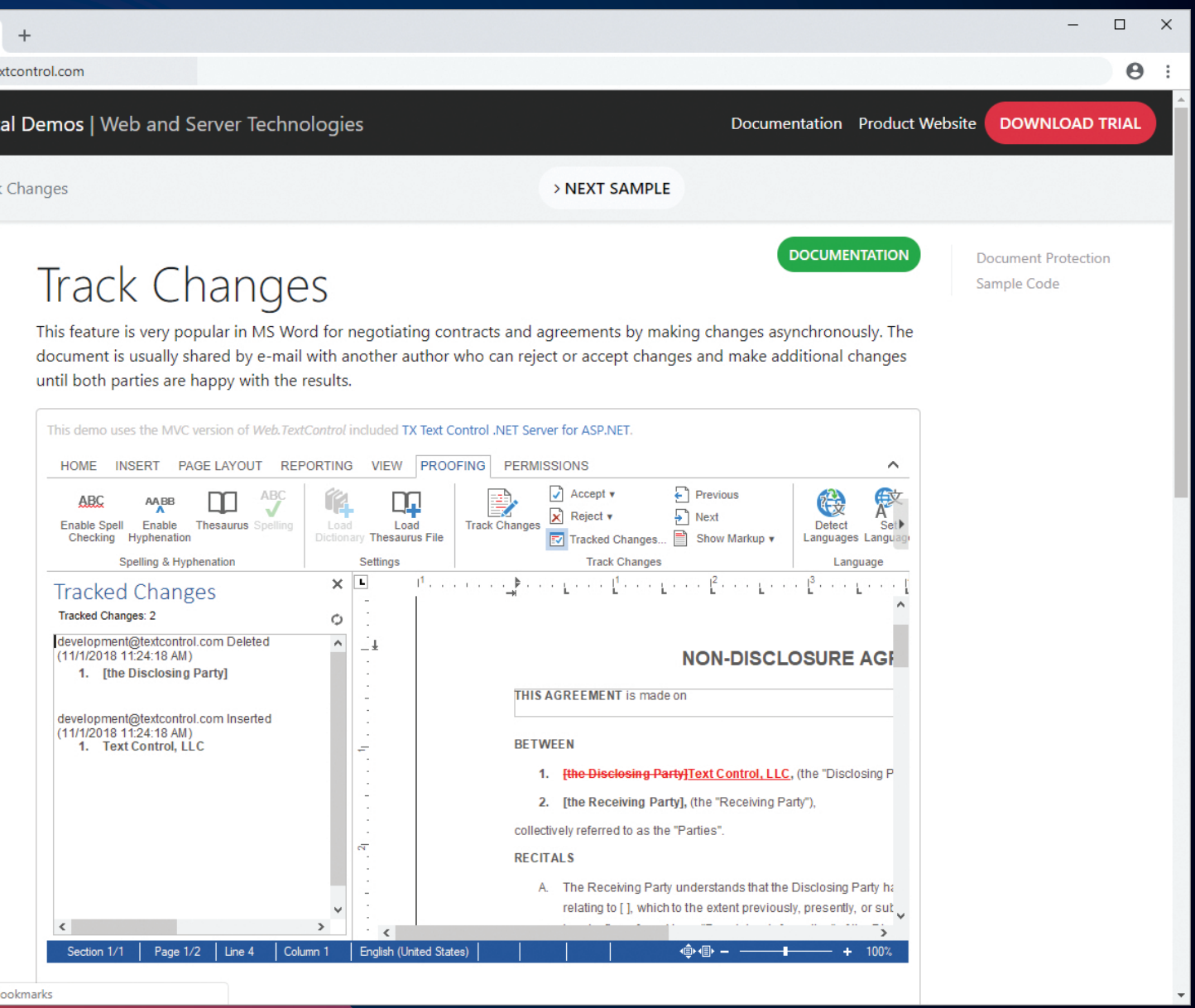


WE ARE CHANGING THE WAY YOU LOOK AT REPORTING

Visual Studio Partner

TX Text Control X16 Released

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!



The implementation of the system can make these updates cheap and safe—or they can make them expensive and risky. There are many options for making a system based on machine learning more flexible and efficient over time. Common investments include:

The Intelligence Runtime To use machine learning you need to do the basics, like implement a runtime that loads and executes models, featurizes the application context and gives users the right experiences based on what the models say. A runtime can be simple, like linking a library into your client, or it can be complex, supporting things like:

- Changes to the types of models used over time, moving from simple rules toward more complex machine learning approaches as you learn more about your problem.
- Combining models that run on the client, in the service, and in the back end, and allowing models to migrate between these locations over time based on cost and performance needs.
- Supporting reversion when deployments go wrong, and ways to rapidly override specific costly mistakes that machine learning will almost certainly make.

Intelligence Management As new models become available, they must be ingested and delivered to where they're needed. For example, models may be created in a lab at corporate headquarters, but must execute on clients across the world. Or maybe the models run partially in a back end and partially in a service. You can rely on the people producing the models to do all the deployments, the verification, and keep everything in sync, or you could build systems to support this.

Intelligence Telemetry An effective telemetry system for machine learning collects data to create increasingly better models over time. The intelligence implementation must decide what to observe, what to sample, and how to digest and summarize the information to enable intelligence creation—and how to preserve user privacy in the process. Telemetry can be very expensive and telemetry needs will change during the lifetime of a machine learning-based system, so it often makes sense to implement tools to allow adaptability while controlling costs.

The Intelligence Creation Environment For machine learning-based systems to succeed, there needs to be a great deal of coordination between the runtime, delivery, monitoring and creation of your models. For example, in order to produce accurate models, the model creator must be able to recreate exactly what happens at runtime, even though the model creator's data comes from telemetry and runs in a lab, while the runtime data comes from the application and runs in context of the application.

Mismatches between model creation and runtime are a common source of bugs, and machine learning professionals often aren't the best people to track these issues down. Because of this, an implementation can make machine learning professionals much more productive by providing a consistent intelligence creation experience.

For all of these components (the runtime, the intelligence management, intelligence telemetry and intelligence creation) you might implement something bare bones that does the basics and relies on ongoing engineering investments to adapt over time. Or you might create something flexible with slick tools for non-engineers so they can rebalance toward new objectives cheaply, quickly and with confidence that they won't mess anything up.

Intelligence Orchestration

Intelligence orchestration is a bit like car racing. A whole team of people builds a car, puts all the latest technology into it, and gets every aerodynamic wing, ballast, gear-ratio, and intake valve set perfectly. They make an awesome machine that can do things no other machine can do.

And then someone needs to get behind the wheel, take it on the track and win!

Intelligence orchestrators are those drivers. They take control of the Intelligent System and do what it takes to make it achieve its objectives. They use the intelligence creation and management systems to produce the right intelligence at the right time and combine it in the most useful ways. They control the telemetry system, gathering the data needed to make their models better. And they deal with all the mistakes and problems, balancing everything so that the application produces the most value it can for users and for your business.

Right about now you might be saying, "Wait, I thought machine learning was supposed to tune the system throughout its lifecycle. What is this? Some kind of joke?" Unfortunately, no. Artificial intelligence and machine learning will only get you so far. Orchestration is about taking those tools and putting them in the best situations so they can produce value—highlighting their strengths and compensating for their weaknesses—while also reacting as things change over time. Orchestration might be needed because:

Your objective changes: As you work on something, you'll come to understand it better. You might realize that you set the wrong objective to begin with, and want to adapt. Heck, maybe the closed loop between your users and your models turns out to be so successful that you want to aim higher.

Your users change: New users will come (and you will cheer) and old users will leave (and you might cry), but these users will bring new contexts, new behavior, and new opportunities to adapt your models.

The problem changes: The approaches and decisions you made in the past might not be right for the future. Sometimes a problem might be easy (like when all the spammers are on vacation). At other times it might get very hard (like near the holidays). As a problem changes, there's almost always opportunity to adapt and achieve better outcomes through orchestration.

The quality of your models changes: Data unlocks possibilities. Some of the most powerful machine learning techniques aren't effective with "small" data, but become viable as users come to your system and you get lots and lots of data. These types of changes can unlock all sorts of potential to try new experiences or target more aggressive objectives.

The cost of running your system changes: Big systems will constantly need to balance costs and value. You might be able to change your experience or models in ways that save a lot of money, while only reducing value to users or your business by a little.

Someone tries to abuse your system: Unfortunately, the Internet is full of trolls. Some will want to abuse your service because

they think that's fun. Most will want to abuse your service (and your users) to make money—or to make it harder for you to make money. Left unmitigated, abuse can ruin your system, making it such a cesspool of spam and risk that users abandon it.

One or more of these will almost certainly happen during the lifecycle of your machine learning-based system. By learning to identify them and adapt, you can turn these potential problems into opportunities.

Implementing machine learning-based systems and orchestrating them are very different activities. They require very different mindsets. And they're both absolutely critical to achieving success. Good orchestrators will:

- **Be domain experts** in the business of your system so they understand your users' objectives instinctively.
- **Comprehend experience** and have the ability to look at interactions and make effective adaptations in how model outputs are presented to users.
- **Understand the implementation** so they know how to trace problems and have some ability to make small improvements.
- **Be able to ask questions of data** and understand and communicate the results.
- **Know applied machine learning** and be able to control your model creation processes and inject new models into the system.
- **Get satisfaction from** making a system execute effectively day in and day out.

Wrapping Up

Machine learning is a fantastic tool. But getting the most out of machine learning requires a lot more than building a model and making a few predictions. It requires adding the machine learning skills to the other techniques you use for organizing large software systems and the teams of people building them.

This article gave a very brief overview of one design pattern for using machine learning at scale, the Closed-Loop Intelligence System pattern. This includes knowing when you need machine learning; what it means to close the loop between users and machine learning; how to rebalance the system to achieve more meaningful objectives over time; what implementations can make it more efficient and safer to adapt; and some of the things that might happen as you run the system over time.

Artificial intelligence and machine learning are changing the world, and it's an exciting time to be involved. ■

Geoff Hulten *is the author of "Building Intelligent Systems" (intelligentsystem.io/book/). He's managed applied machine learning projects for more than a decade and taught the master's level machine learning course at the University of Washington. His research has appeared in top international conferences, received thousands of citations, and won a SIGKDD Test of Time award for influential contributions to the data mining research community that have stood the test of time.*

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

Secure Multi-Party Machine Learning with Azure Confidential Computing

Stefano Tempesta

Security is paramount when implementing business solutions hosted in a public cloud, especially when sensitive data and intellectual property are stored as part of the solution itself. There are best practices in the industry already for securing data at rest and in transit, but you also need to protect your data from unauthorized access when it's being used. Azure confidential computing (bit.ly/2BxQkpp) provides a new level of data protection and encryption when your data is being processed, using Trusted Execution Environments (TTEs). TEEs are implemented at the hardware or software (hypervisor) level and provide a protected processor and memory space where your code and data can run in complete isolation from outside applications and systems. TEEs ensure there's no way to view data or the operations inside from the outside, even with a debugger. They even ensure that only authorized code is permitted to access data. If the code is altered or tampered, the operations are denied and the environment disabled. The TEE enforces these protections throughout the execution of code within it.

Some of the technology discussed in this article is still in development. All information is subject to change.

This article discusses:

- SQL Server Always Encrypted with Secure Enclaves
- Multi-party sensitive data sources
- Enclave attestation
- Prediction in machine learning

Technologies discussed:

Azure Confidential Computing, Open Enclave SDK, Always Encrypted with Secure Enclaves, Virtualization Based Security, Windows Defender System Guard with Host Guardian Service, .NET Framework Data Provider for SQL Server, ML.NET

Code download available at:

bit.ly/2Exp78V

SQL Server Always Encrypted with Secure Enclaves

In my previous article, "Protect Your Data with Azure Confidential Computing" (msdn.com/magazine/mt833273), I introduced Azure confidential computing and the Open Enclave SDK for building software applications that run protected code and data in a trusted execution environment. The same technology is in use for Azure SQL Database and SQL Server. This is an enhancement to the Always Encrypted capability, which ensures that sensitive data within a SQL database can be encrypted at all times without compromising the functionality of SQL queries. Always Encrypted with Secure Enclaves achieves this by delegating computations on sensitive data to an enclave, where the data is safely decrypted and processed. Available with SQL Server 2019, the enclave technology adopted, called Virtualization Based Security (VBS), isolates a region of memory within the address space of a user-mode process that's completely invisible to all other processes and to the Windows OS

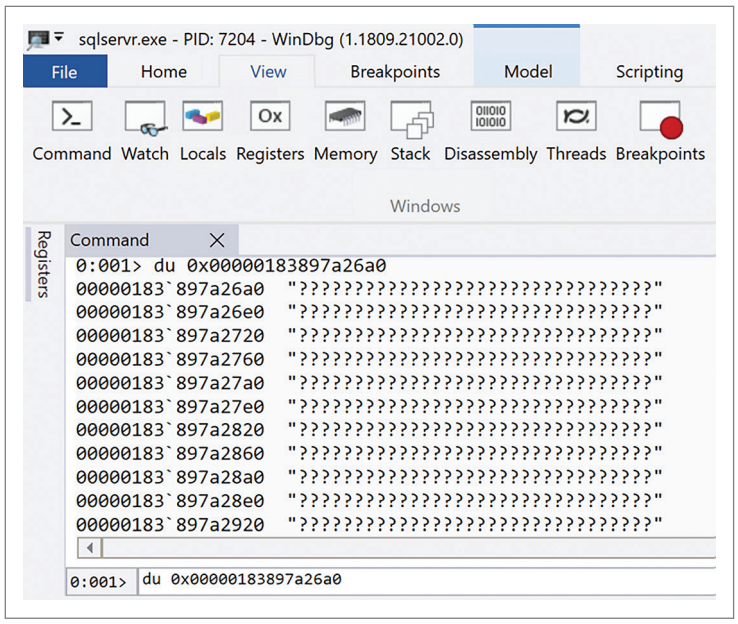


Figure 1 **Browsing a Virtualization Based Security Enclave with WinDbg**

on the machine. Even machine administrators aren't able to see the memory of the enclave. **Figure 1** shows what an admin would see when browsing the enclave memory with a debugger (note the question marks, as opposed to the actual memory content).

The way the enhanced Always Encrypted feature uses enclaves is illustrated in **Figure 2**. The SQL Server instance contains an enclave, loaded with the code for performing data encryption, as well as the code implementing SQL operations. Before submitting a query to the SQL Server engine for processing, the SQL driver used by the client application sends the encryption keys to the enclave over a secure channel. When processing queries, the SQL Server engine runs these operations directly within the enclave, where the data is safely decrypted and processed. Data never leaves the enclave unencrypted.

Not all data in a database table requires encryption. Specific columns can be identified for securing data in an enclave. When parsing a SQL query, the SQL Server engine determines if the query contains any operations on encrypted data that require the use of the secure enclave. For queries where the secure enclave needs to be accessed, the client driver sends the encryption keys for the specific table columns required for the operations to the secure enclave, and then it submits the query for execution along with the encrypted query parameters. Data contained in the identified secured columns is never exposed in the clear outside of the enclave. SQL Server decrypts data contained in these columns only within the secure enclave. If the query contains parameters on the secured columns, these parameters are also decrypted within the enclave before the query runs.

Columns that contain sensitive data to encrypt can be identified with an ALTER COLUMN statement. This statement is executed within the enclave. There's no need to move data out of the database for initial encryption or for other encryption-related schema changes. This improves the performance and reliability of such operations greatly, and it doesn't require special client-side tools. Assuming you want to protect the age of patients, contained in the Age column of the MedicalRecords database table, the following statement encrypts data in that column with the AES-256 encryption algorithm:

```
ALTER TABLE [dbo].[MedicalRecords]
ALTER COLUMN [Age] [char](11)
ENCRYPTED WITH (COLUMN_ENCRYPTION_KEY = [CEK1],
  ENCRYPTION_TYPE = Randomized,
  ALGORITHM = 'AEAD_AES_256_CBC_HMAC_SHA_256') NOT NULL
GO
```

You can find comprehensive documentation on the new capabilities and on how to get started with Always Encrypted with Secure Enclaves at aka.ms/AlwaysEncryptedwithSecureEnclaves.

Multi-Party Sensitive Data Sources

In addition to SQL Server, there's potentially a broad application of Azure confidential computing across many industries, including finance, retail, government and health care. Even competing organizations would benefit from pooling their private datasets, training machine learning models on the aggregate data to achieve higher accuracy of prediction. For example, health institutes could collaborate by sharing their private patient data, like genomic sequences and medical records, to gain deeper insights from machine learning across multiple datasets, without risk of data being leaked to other organizations. Having more data to train on allows machine learning algorithms to produce better models. Confidential computing can be used for privacy: multiple parties leverage a common machine learning service to be executed on their aggregate data. Although health institutes, in this use case, wouldn't want their own datasets to be shared with any another party, they can each upload their encrypted data in an enclave, perform remote attestation, run the machine learning code and, finally, download the encrypted machine learning model. The model may also be kept within the enclave for secure evaluation by all of the parties, subject to their agreed access control policies.

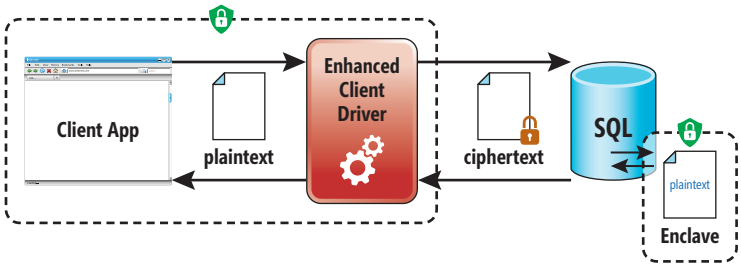


Figure 2 **Communication to SQL Server Is Always Encrypted with Secure Enclaves**

Figure 3 provides an overview of such a system. Multiple hospitals encrypt patient datasets, each with a different key. The hospitals deploy an agreed-upon machine learning algorithm in an enclave in a cloud datacenter and share their data keys with the enclave. The enclave processes the aggregate datasets and outputs an encrypted machine learning model.

In this scenario, sensitive data is stored in a SQL Server database with Always Encrypted with Secure Enclaves,

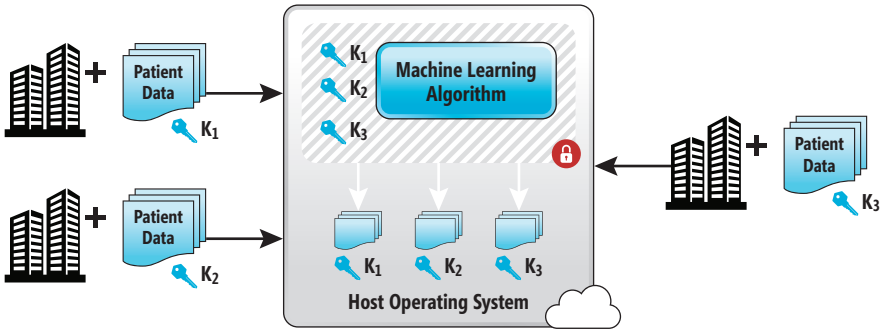


Figure 3 **Secure Multi-Party Machine Learning**

and shared with a machine learning service that runs in a TTE. Organizations use the Open Enclave SDK (openenclave.io) to build portable C or C++ applications against different enclave types. With the SDK, they also take advantage of enclave creation and management, system primitives, runtime support, and cryptographic library support by using a consistent API and an enclaving abstraction. The machine learning application benefits from combined multiple data sources to produce better training models. As data is protected by SQL Server, it's the responsibility of the SQL engine to authenticate the machine learning enclave before sharing any data with it. Once this attestation is made, data is decrypted and visible solely to the machine learning code, without revealing it to other applications, even if running on the same virtual machine or app service, or the hosting cloud platform.

Enclave Attestation

The secure enclave inside the SQL Server engine can access sensitive data stored in encrypted database columns and the corresponding column encryption keys. But before an application can send a SQL query that involves enclave computations to SQL Server, it should verify that the enclave is a genuine enclave based on a given technology—VBS for example—and the code running inside the enclave has been properly signed. The process of verifying the enclave is called enclave attestation, and, in the case of SQL Server, it involves the SQL client driver used by the client application to contact an external attestation service. The specifics of the attestation process depend on the enclave technology and the attestation service. The attestation process SQL Server supports for VBS secure enclaves in SQL Server 2019 is called Windows Defender System Guard runtime attestation, which uses Host Guardian Service (HGS) as an attestation service. You need to configure HGS in your environment and register the machine hosting your SQL Server instance in HGS. You also have to configure your client applications with an HGS attestation.

To establish trust among parties, VBS enclaves expose an enclave attestation report that's fully signed by a VBS-unique key. An attestation service uses this report to establish a trust relationship between a client application and an enclave. Essentially, attestation involves the enclave report, which is proof of the identity and integrity of the TEE application, and proof of platform integrity (which includes measurements of host and TEE). The VBS unique key is used to link the two because it's used to sign the report and is stored in the measurements. HGS validates the platform measurements and issues a health certificate with the key in it. The client then validates the entire report using the VBS key, validates the VBS key itself using the health certificate, and eventually also validates the health certificate issued by the HGS it trusts. An enclave gets the keys to encrypt and decrypt data via a secure tunnel established by the client driver and the enclave itself. A session key is generated for communication over this channel. Then, the client driver encrypts the columns encryption key with the session key, and signs and submits SQL queries that require enclave computations. **Figure 4** extends the communication protocol of SQL Server Always Encrypted with Secure Enclaves to the use of an Attestation Service.

To use Always Encrypted with secure enclaves, an application must use a client driver that supports the feature. In SQL Server 2019, your applications should use .NET Framework 4.6 or higher and .NET Framework Data Provider for SQL Server. In addition, .NET applications must be configured with a secure enclave provider specific to the enclave type (for example, VBS) and the attestation service (for example, HGS), you're using. The supported enclave providers are shipped separately in a NuGet package, which you need to integrate with your application. An enclave provider implements the client-side logic for the attestation protocol and for establishing a secure channel with a secure enclave of a given type. Always Encrypted isn't currently supported in .NET Core.

You can find step-by-step instructions for configuring SQL Server Always Encrypted with secure enclaves using SQL Server Management Studio (SSMS) in the tutorial at bit.ly/2Xig9nr. The tutorial goes through the installation of an HGS computer to run HGS to support host key attestation, and then the configuration of SQL Server as a guarded host registered with HGS using host key attestation. Please note that the purpose of the tutorial is to create a simple environment for testing purposes only. Host key attestation is recommended for use in test environments. You should use Trusted Platform Module (TPM) attestation for production environments. TPM is a technology designed to provide hardware-based

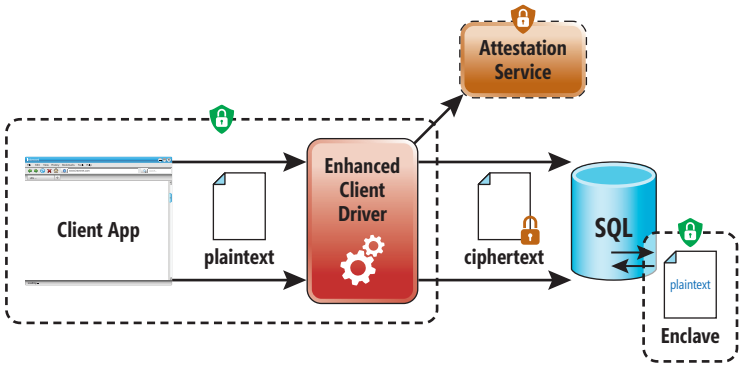


Figure 4 **Enclave Attestation for a SQL Server Database**

security functions. A TPM chip is a secure crypto-processor designed specifically to carry out cryptographic operations directly inside the CPU. The chip includes multiple physical security mechanisms to make it tamper-resistant to software applications. If you're interested in knowing more about TPM and how to initialize HGS using TPM-trusted attestation, you can start with bit.ly/2T3lVuN.

To connect to an instance of SQL Server Always Encrypted, the connection string of your client application should include two additional parameters:

- Column Encryption Setting = Enabled
- Enclave Attestation URL=https://<Your-HGS-Server>/Attestation

For example, the connection string for a .NET application would be similar to the following setting:

```
<connectionStrings>
  <add name="DataConnection" connectionString="Data Source=.;
    Initial Catalog=[Database Name]; Integrated Security=true;
      Column Encryption Setting = Enabled;
        Enclave Attestation URL=https://[Your-HGS-Server]/Attestation" />
</connectionStrings>
```

Prediction in Machine Learning

The machine learning application I'm presenting in this article is built with ML.NET (dot.net/ml), an open source and cross-platform machine learning framework for .NET. Currently still in development, ML.NET enables machine learning tasks like classification and regression. The framework also exposes APIs for training models, as well as core components such as data structures, transformations and learning algorithms.

For the health care use case, the machine learning application implements a multi-class classification task that classifies patients into groups according to the likelihood they'll develop a specific disease (to keep things less morbid here, I'll refer to a generic disease rather than naming one specifically). As anticipated in the description of this multi-party data source scenario, health institutes share their encrypted datasets, containing all the necessary features for training a model.

The machine learning code is a Web API developed in C#. In your Visual Studio solution, add a reference to the Microsoft.ML NuGet package. This package contains the ML.NET library and the relevant namespaces to add to the application:

```
using Microsoft.ML;
using Microsoft.ML.Core.Data;
using Microsoft.ML.Data;
```

The Web API solution implements the POST method only, which accepts a JSON payload that describes a medical record. The MedicalRecord class in the solution, shown in **Figure 5**, identifies all the features analyzed by the ML.NET engine. Risk is the label of the dataset; it's the value on which it's being trained.

The JSON message request assumes the following format:

```
{
  "risk": 0.0,
  "age": 0,
  "sex": 0,
  "smoker": false,
  "chestPain": 0,
  "bloodPressure": 0,
  "serumCholestoral": 0,
  "fastingBloodSugar": false,
  "maxHeartRate": 0
}
```

The Classification class implements the multi-class classification task that, given a medical record in input, assigns that patient to a risk class of contracting the disease. This process develops over the following four steps:

1. Instantiate an MLContext object that represents the ML.NET context class.
2. Train the model based on historical datasets available in SQL Server (and protected within an enclave).
3. Evaluate the model.
4. Score it.

The output is returned to the client application (the health institute) in the form of a JSON response:

```
{
  "score": 0.0,
  "accuracy": 0.0
}
```

But let's go in order and describe each step in the machine learning code. The first step is to create an instance of MLContext as part of the Classification class, with a random seed (seed: 0) for repeatable and deterministic results across multiple trainings. MLContext is your reference object for access to source data and

Figure 5 **The MedicalRecord Class**

```
public class MedicalRecord
{
  public float Risk { get; set; }
  public int Age { get; set; }
  public int Sex { get; set; }                // 0 = Female, 1 = Male
  public bool Smoker { get; set; }
  public ChestPainType ChestPain { get; set; }
  public int BloodPressure { get; set; }      // In mm Hg on admission to
                                              // the hospital
  public int SerumCholestoral { get; set; }   // In mg/dl
  public bool FastingBloodSugar { get; set; } // True if > 120 mg/dl
  public int MaxHeartRate { get; set; }       // Maximum heart rate achieved
}

public enum ChestPainType
{
  TypicalAngina = 1,
  AtypicalAngina,
  NonAnginal,
  Asymptomatic
}
```

Figure 6 **Training the Model**

```
public IEnumerable<MedicalRecord> ReadData(int split = 100)
{
  IEnumerable<MedicalRecord> medicalRecords = new List<MedicalRecord>();

  using (var connection = new SqlConnection(ConnectionString))
  {
    connection.Open();
    using (SqlCommand cmd = connection.CreateCommand())
    {
      cmd.CommandText = SqlQuery(split);
      using (SqlDataReader reader = cmd.ExecuteReader())
      {
        if (reader.HasRows)
        {
          while (reader.Read())
          {
            medicalRecords.Append(ReadMedicalRecord(reader));
          }
        }
      }
    }
  }

  return medicalRecords;
}
```

trained models, as well as the relevant machine learning transformers and estimators:

```
public Classification()
{
  _context = new MLContext(seed: 0);
}

private MLContext _context;
```

The next step in the machine learning process is training the model, as shown in **Figure 6**. This is the step that loads data from the SQL Server database in the secure enclave. A connection to SQL Server is created using the SqlConnection class, which internally works out the attestation and communication with the enclave over a secure channel.

Please note the split parameter that was used to select a percentage of records from the database, using the tablesample functionality in SQL Server. Training and evaluation of a model require a different subset of data as input. A good approximation is to allocate 80 percent of data for training, depending on its quality. The higher the quality of input, the better the accuracy of prediction:

```
private static string SqlQuery(int split) =>
  $"SELECT [Age], [Sex], [Smoker], [ChestPain],
    [BloodPressure], [SerumCholestoral], [FastingBloodSugar],
      [MaxHeartRate] FROM [dbo].[MedicalRecords] tablesample({split} percent)";
```

Data is then loaded into the current ML context, and a trained model is obtained using a method that predicts a target using a linear multi-class classification model trained with the Stochastic Dual Coordinate Ascent (SDCA) algorithm, a process for solving large-scale supervised learning problems. The multi-class classification trainer is added to a pipeline of trainers before the actual "fit"; that is, before the training of the model happens. ML.NET transforms the data employing a chain of processes that apply a custom transformation before training or testing.

The purpose of the transformation in the TrainModel method, shown in **Figure 7**, is to label source data that specifies the format the machine learning algorithm recognizes. To accomplish this featurization of data, the transformation pipeline combines all of the feature columns into the Features column using the concatenate method. But before that, I need to convert any non-float data to float data types as all ML.NET learners expect features to be float vectors.

All features in the medical record are basically categories and not float values. So, before concatenating them into a single output column, I apply a categorical transformation to each one of them using the OneHotEncoding function. By default, a learning algorithm processes only features from the Features column. The AppendCacheCheckpoint method at the end caches the source data, so when you iterate over the data multiple times using the cache, you might get better performance. The actual training happens when you invoke the Fit method on the estimator pipeline. Estimators learn from data and produce a machine learning model. The Fit method returns a model to use for predictions. Trained models can eventually be persisted to a stream, for future utilization.

Following the model training, model evaluation is essential for ensuring the good quality of the next step, model scoring, which is the outcome prediction. Evaluation of predicted data is performed against a dataset obtained as a transformation of test data from the trained model, using a percentage of the source data (in this case 20 percent), as shown in **Figure 8**. The Evaluate method of the MulticlassClassification object defined within the current ML.NET context computes the quality metrics for the model using the specified dataset. It returns a MultiClassClassifierMetrics object that contains the overall metrics computed by multi-class classification evaluators. In my example, I calculate the accuracy

Figure 7 **Obtaining a Training Model**

```
public void TrainModel(IDataSource<MedicalRecord> dataSource, Stream targetStream)
{
  IEnumerable<MedicalRecord> sourceData = dataSource.ReadData(80);
    // 80% of records for training
  IDataView trainData = _context.Data.
ReadFromEnumerable<MedicalRecord>(sourceData);

  // Convert each categorical feature into one-hot encoding independently
  var transformers = _context.Transforms.Categorical.OneHotEncoding(
    "MedicalRecordAge", nameof(MedicalRecord.Age))
    .Append(_context.Transforms.Categorical.OneHotEncoding(
      "MedicalRecordSex", nameof(MedicalRecord.Sex)))
    .Append(_context.Transforms.Categorical.OneHotEncoding(
      "MedicalRecordSmoker", nameof(MedicalRecord.Smoker)))
    .Append(_context.Transforms.Categorical.OneHotEncoding(
      "MedicalRecordChestPain", nameof(MedicalRecord.ChestPain)))
    .Append(_context.Transforms.Categorical.OneHotEncoding(
      "MedicalRecordBloodPressure", nameof(MedicalRecord.BloodPressure)))
    .Append(_context.Transforms.Categorical.OneHotEncoding(
      "MedicalRecordSerumCholestoral", nameof(
        MedicalRecord.SerumCholestoral)))
    .Append(_context.Transforms.Categorical.OneHotEncoding(
      "MedicalRecordFastingBloodSugar", nameof(
        MedicalRecord.FastingBloodSugar)))
    .Append(_context.Transforms.Categorical.OneHotEncoding(
      "MedicalRecordMaxHeartRate", nameof(MedicalRecord.MaxHeartRate)));

  var processChain = _context.Transforms.Concatenate(DefaultColumnNames.Features,
      nameof(MedicalRecord.Age),
      nameof(MedicalRecord.Sex),
      nameof(MedicalRecord.Smoker),
      nameof(MedicalRecord.ChestPain),
      nameof(MedicalRecord.BloodPressure),
      nameof(MedicalRecord.SerumCholestoral),
      nameof(MedicalRecord.FastingBloodSugar),
      nameof(MedicalRecord.MaxHeartRate)
  ).AppendCacheCheckpoint(_context);

  var trainer =
    _context.MulticlassClassification.Trainers.StochasticDualCoordinateAscent(
      labelColumn: DefaultColumnNames.Label,
        featureColumn: DefaultColumnNames.Features);
  var pipeline = processChain.Append(trainer);
  var model = pipeline.Fit(trainData);

  _context.Model.Save(model, targetStream);
}
```

Figure 8 **Evaluating the Model**

```
public double EvaluateModel(IDataSource<MedicalRecord> dataSource,
  Stream modelStream)
{
  IEnumerable<MedicalRecord> sourceData = dataSource.ReadData(20);
    // 20% of records for evaluation
  var testData =
    _context.Data.ReadFromEnumerable<MedicalRecord>(sourceData);

  var predictions = _context.Model.Load(modelStream).Transform(testData);
  var metrics = _context.MulticlassClassification.Evaluate(
    data: predictions, label: DefaultColumnNames.Label,
      score: DefaultColumnNames.Score,
        predictedLabel: DefaultColumnNames.PredictedLabel, topK: 0);

  return metrics.TopKAccuracy / metrics.TopK * 100.0 / sourceData.Count();
}
```

as a percentage of correct predictions out of the total amount of records evaluated. The TopKAccuracy metric describes the relative number of examples where the true label is one of the top K predicted labels.

The last step in the classification API is to score the model and generate a prediction that identifies a risk for a person with the given medical record. Scoring a model starts from loading the data model previously trained and saved, and then creating a prediction engine from that trained model. In the Score method, I load the trained model and then invoke the CreatePredictionEngine function with the medical record and risk prediction types (the MedicalRecord and RiskPrediction C# classes, respectively). The prediction engine generates a score that identifies the class of risk of a patient with the analyzed medical record:

```
public double Score(MedicalRecord medicalRecord, Stream modelStream)
{
  ITransformer trainedModel = _context.Model.Load(modelStream);
  var predictionEngine = trainedModel.CreatePredictionEngine<MedicalRecord,
    RiskPrediction>(_context);

  var prediction = predictionEngine.Predict(medicalRecord);
  return prediction.Score;
}
```

The RiskPrediction object contains the Score and Accuracy attributes returned by the Post method of the Web API. For the Score property, I added the ColumnName attribute to tell ML.NET that this is where I want to output the scored value produced by the Predict method of the prediction engine:

```
public class RiskPrediction
{    [ColumnName("Score")]
  public double Score;

  public double Accuracy;
}
```

Finally, as shown in **Figure 9**, the Post method of the health care Web API processes all these steps in sequence:

1. Obtains a MedicalRecord object from its representation in JSON.
2. Instantiates the Classification helper, which uses ML.NET for multi-class classification.
3. Trains and saves the model.
4. Calculates its accuracy
5. Predicts the risk associated with the medical record provided in input.

Figure 9 **Predicting the Risk of a Particular Medical Record**

```
public IHttpActionResult Post(string medicalRecordJson)
{
  MedicalRecord medicalRecordObj =
    JsonConvert.DeserializeObject<MedicalRecord>(medicalRecordJson);

  Classification classification = new Classification();

  FileStream modelStream = new FileStream("[Path]", FileMode.Create);
  classification.TrainModel(new MedicalRecordDataSource(), modelStream);

  double accuracy = classification.EvaluateModel(
    new MedicalRecordDataSource(), modelStream);
  double score = classification.Score(medicalRecordObj, modelStream);
  RiskPrediction prediction = new RiskPrediction
  {
    Score = score,
    Accuracy = accuracy
  };

  return Json<RiskPrediction>(prediction);
}
```

Please consider that this code has a margin for performance improvement in caching trained models for further use, for example. Model training and evaluation can be skipped when working with pre-trained models, yielding a significant performance boost.

In my tests, I tried running the prediction process against individual datasets, obtained by health institutes, obtaining an average accuracy of 90 percent. Each dataset has roughly 100,000 records. When combining multiple datasets into a single one, accuracy reached 96 percent.

> Model training and evaluation can be skipped when working with pre-trained models, yielding a significant performance boost.

Wrapping Up

In this article, I showed how a machine learning application can produce better predictions by leveraging a broader dataset, by collecting data from several trusted sources without compromising confidentiality of the information shared among all parties involved. This is achieved by protecting data in use by the machine learning code within a SQL Server Always Encrypted enclave, which is part of the Azure confidential computing technology. The machine learning solution collects datasets from different protected sources and, by combining data into a larger dataset, obtains higher accuracy of prediction on the trained model.

It's important to mention that confidential computing requires your code to run in a TEE. In the example presented in this article, the SQL client driver is the intermediator with the SQL Server trusted execution environment. The ML.NET code, therefore, may or may not run within a TEE. If it does, it's called a secure multi-party machine learning computation. Multi-party computation implies that there are two or more parties of mutually exclusive trust that have data and/or code they would like to protect against each other. In this context, the parties would like to perform computation together to generate a shared output, without leaking any sensitive data. The enclave acts as a trusted intermediator for this computation.

The entire solution is available on GitHub at bit.ly/2Exp78V. ■

Stefano Tempesta *is a Microsoft Regional Director, MVP on AI and Business Applications, and member of Blockchain Council. A regular speaker at international IT conferences, including Microsoft Ignite and Tech Summit, Tempesta's interests extend to blockchain and AI-related technologies. He created Blogchain Space (blogchain.space), a blog about blockchain technologies, writes for* MSDN Magazine *and* MS Dynamics World, *and publishes machine learning experiments on the Azure AI Gallery (gallery.azure.ai).*

Thanks to the following Microsoft technical experts who reviewed this article: *Gleb Krivosheev, Simon Leet*

Visual Studio LIVE!

EXPERT SOLUTIONS FOR ENTERPRISE DEVELOPERS

Developer Training Conferences and Events

In-depth Technical Content On:

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

NEW IN 2019!
VSLIVE! CONFERENCE ON-DEMAND

Visual Studio Live! is now offering online access to audio/video recordings of all sessions Tuesday - Thursday at every location this year.

VSLive! On-Demand gives you cutting-edge developer content in an easy, flexible, format with two convenient options: add to your existing Conference Registration or skip the travel and purchase the the online access to the recorded sessions for one full year.

Learn more at VSLive.com/ondemand

SUPPORTED BY

Microsoft | Visual Studio | msdn magazine | Visual Studio MAGAZINE

PRODUCED BY

CONVERGE 360 An 1105 MEDIA company

JOIN US IN 2019!

April 22-26

Hyatt Regency

For the first time in our 20-year history, Visual Studio Live! is heading down south to New Orleans for intense developer training, bringing our hard-hitting sessions, well-known coding experts and unparalleled networking to the Big Easy!

REGISTER NOW!
vslive.com/neworleans

June 9-13

Hyatt Regency Cambridge

Join Visual Studio Live! for an amazing view of Beantown, bringin g our infamous speakers for intense developer training, Hands-On Labs, workshops, sessions and networking adventures to the Northeast.

REGISTER NOW!
vslive.com/boston

June 18-19

Microtek Training Center San Jose

Develop and ASP.NET Core 2.x Service and Website with EF Core 2.x in two days with Visual Studio Live!'s training seminar in San Jose, CA. Expand your knowledge and accelerate your career today.

REGISTER NOW!
vslive.com/sanjose

August 12-16

Microsoft HQ

Join our Visual Studio Live! experts at the Mothership for 5 days of developer training and special Microsoft perks unique to our other show locations. Plus, we are adding the ever-so popular full-day Hands-On Labs to the agenda in Redmond for the first time this year!

REGISTER NOW!
vslive.com/microsofthq

Sept. 29-Oct. 3

Westin Gaslamp Quarter

Head to the heart of the San Diego Gaslamp District with Visual Studio Live! this Fall as we immerse ourselves with all things for developers, including several workshops, sessions and networking opportunities to choose from.

REGISTER NOW!
vslive.com/sandiego

October 6-10

Swissotel

Head to the Windy City and join Visual Studio Live! this October for 5 days of unbiased, developer training and bringing our well-known Hands-On Labs to the city for the first time.

REGISTER NOW!
vslive.com/chicago

November 17-22

Royal Pacific Resort at Universal

Visual Studio Live! Orlando is a part of Live! 360, uniquely offering you 6 co-located conferences for one great price! Stay ahead of the current trends and advance your career – join us for our last conference of the year!

REGISTER NOW!
vslive.com/orlando

INTRODUCING...

VISUAL STUDIO LIVE!

GIVES BACK

The "VSLive! Gives Back" program gives conference participants an opportunity to make a positive contribution to the host community in the short amount of time they spend there.

Check out each city to learn more about the programs we're supporting.

Implementing Your Own Enterprise Search

Xavier Morera

You probably take search for granted. You use it daily to perform all kinds of tasks, from discovering a room for your next trip, to locating information you need for your job. A properly implemented search can help you save money—or make money. And even in a great app, a bad search still creates an unsatisfactory UX.

The problem is that despite its importance, search is one of the most misunderstood functionalities in IT, noticed only when missing or broken. But search doesn't need to be an obscure feature that's difficult to implement. In this article I'm going to explain how you can develop an enterprise search API in C#.

This article discusses:

- Getting Solr
- Getting and understanding the data
- Using a classic schema
- Getting SolrNet and modeling the data
- Building an indexer and creating a search application
- Using faceting to narrow results

Technologies discussed:

Solr, SolrNet

Code download available at:

msdn.com/magazine/0419magcode

To learn about search, you need a search engine. There are many options, from open source to commercial and everything in between, many of which internally leverage Lucene—the information retrieval library of choice. This includes Azure Search, ElasticSearch and Solr. Today I'll use Solr. Why? It has been around for a long time, has good documentation, a vibrant community, many notable users, and I've personally used it to implement search in many applications, from small sites to large corporations.

Getting Solr

Installing a search engine may sound like a complicated task and, indeed, if you're setting up a production Solr instance that needs to support a high number of queries per second (QPS), then yes, it can be complicated. (QPS is a common metric used to refer to the search workload.)

Those steps are covered at length in the "Installing Solr" section of the Solr documentation (lucene.apache.org/solr), as well as in "The Well-Configured Solr Instance" section in the "Apache Solr Reference Guide" (bit.ly/2IK7mqY). But I just need a development Solr instance, so I simply go to bit.ly/2tEXqoo and get a binary release; solr-7.7.0.zip will work.

I download the .zip file, unzip, open the command line and change the directory to the root of the folder where I unzipped the file. Then I issue the following command:

```
> bin\solr.cmd start
```

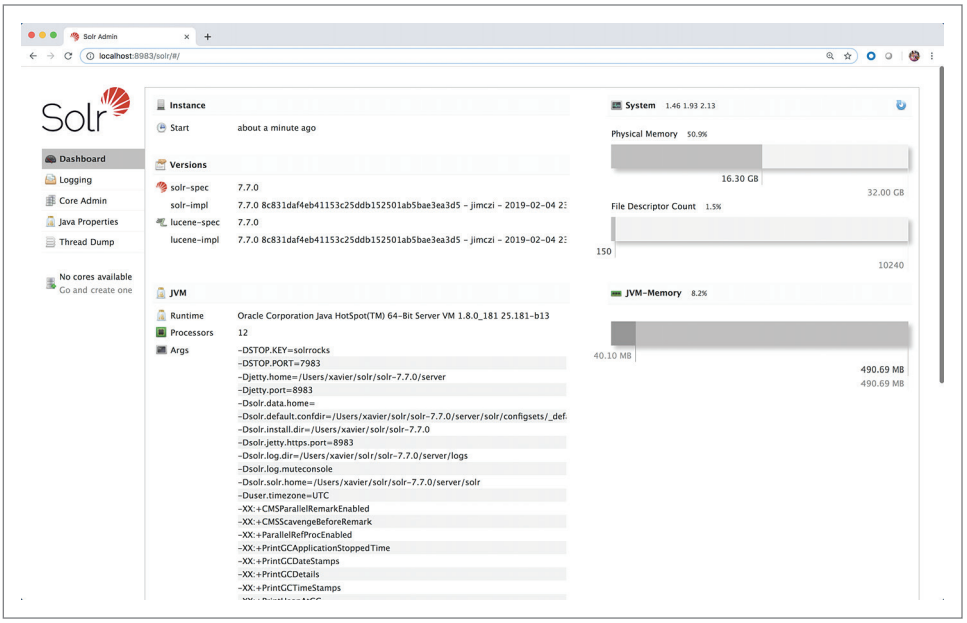


Figure 1 **A Search Engine Up and Running**

That's it. I just need to navigate to http://localhost:8983, where I'm greeted with the Solr Admin UI, shown in **Figure 1**.

Getting Data

Next, I need data. There are plenty of datasets available with interesting data, but every day thousands of developers end up at StackOverflow because they can't quite remember how to write to a file; they can't exit VIM; or they need a C# snippet that solves a particular problem. The good news is that the StackOverflow data is available as an XML dump containing about 10 million questions and answers, tags, badges, anonymized user info, and more (bit.ly/1GsHll6).

Even better, I can select a dataset with the same format from a smaller StackExchange site containing just a few thousand questions. I can test with the smaller dataset first, and later beef up the infrastructure to work with more data.

I'll start with the data from datascience.stackexchange.com, which contains about 25 thousand posts. The file is named datascience.stackexchange.com.7z. I download it and extract Posts.xml.

A Few Required Concepts

I have a search engine and data, so it's a good time to review some of the important concepts. If you're used to working with relational databases, they'll likely be familiar.

The index is where a search engine stores all the data that's collected for searching. At a high level, Solr stores data in what's called an inverted index. In an inverted index, words (or tokens) point to specific documents. When you search for a particular word, that's a query. If the word is found (a hit or a match), the index will indicate which documents contain this word and where.

The schema is how you specify the structure of the data. Each record is called a document, and just as you'd define the columns of a database, you specify the fields of a document in a search engine. You construct the schema by editing an XML file called schema.xml, defining the fields with their types. However, you can also use dynamic fields, in which a new field is created automatically when an unknown field is added. This can be useful for when you're not totally sure of all the fields that are present in the data.

It may have already occurred to you that Solr is quite similar to a NoSQL document store, as it can contain documents with fields that are denormalized and not necessarily consistent across the collection of documents.

Schemaless mode is another way of modeling the data within the index. In this case, you don't explicitly tell Solr which fields you'll be indexing. Instead, you simply add data and Solr constructs a schema based on the type of data being added to the index. To use schemaless mode, you must have a managed schema, which means you can't manually edit fields. This is especially useful for a data exploration phase or proof-of-concept.

I'll use a manually edited schema, which is the recommended approach for production as it yields greater control.

Indexing is the process of adding data to the index. During indexing, an application reads from different sources and prepares data for ingestion. That's the word used for adding documents; you might also hear the term feeding.

Once the documents are indexed, it's possible to search, hopefully returning the documents most relevant to a particular search in the top positions.

I'll use a manually edited schema, which is the recommended approach for production as it yields greater control.

Relevancy ranking is how the most appropriate results are returned at the top. This is an easy-to-explain concept. In the best situation, you run a query and the search engine "reads" your mind, returning exactly the documents you were looking for.

Precision vs. recall: Precision refers to how many of your results are relevant (the quality of your results), while recall means how many of the total returned results are relevant (the quantity or completeness of the results).

Analyzers, tokenizers and filters: When data is indexed or searched, an analyzer examines the text and generates a token stream.

TECHMENTOR

IN-DEPTH TRAINING FOR IT PROS

Training Conference for IT Pros at Microsoft HQ!

The FUTURE of TECH is HERE

MICROSOFT HEADQUARTERS
REDMOND, WA
AUGUST 5-9, 2019

In-depth Technical Tracks on:

- Client and EndPoint Management
- Cloud
- Infrastructure
- PowerShell and DevOps
- Security
- Soft Skills for IT Pros

SAVE $400
WHEN YOU REGISTER BY JUNE 14
Use Promo Code MSDN

EVENT PARTNER Microsoft

SUPPORTED BY Redmond | Redmond Channel Partner | Virtualization & Cloud Review

PRODUCED BY Converge 360 — An 1105 Media company

AGENDA AT-A-GLANCE

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

START TIME	END TIME					
		TechMentor Pre-Conference Hands-On Labs: Monday, August 5, 2019 *(Separate entry fee required)*				
7:30 AM	9:00 AM	Pre-Conference Workshop Registration - Coffee and Light Breakfast				
9:00 AM	12:00 PM	**HOL01** Hands-On Lab: Building a Bulletproof Privileged Access Workstation (PAW) - *Sami Laiho*		**HOL02** Hands-On Lab: Mastering Windows Troubleshooting—Advanced Hands-on Workshop - *Bruce Mackenzie-Low*		
12:00 PM	2:00 PM	Lunch @ The Mixer - Visit the Microsoft Company Store & Visitor Center				
2:00 PM	5:00 PM	Hands-On Lab Continued - *Sami Laiho*		Hands-On Lab Continued - *Bruce Mackenzie-Low*		
6:30 PM	8:30 PM	Dine-A-Round Dinner - Suite in Hyatt Regency Lobby				
START TIME	**END TIME**	**TechMentor Day 1: Tuesday, August 6, 2019**				
7:00 AM	8:00 AM	Registration - Coffee and Light Breakfast				
8:00 AM	9:15 PM	**T01** Microsoft 365 Explained - *Ståle Hansen*	**T02** Top Free Tools for Monitoring Windows Performance - *Bruce Mackenzie-Low*	**T03** Follow the Breadcrumbs Azure Security Center In-Depth - *Mike Nelson*	**T04** From IT Pro to Cloud Pro - *Peter De Tender*	
9:30 AM	10:45 AM	**T05** Using PowerBI and Azure Log Analytics for End User Devices - *Kevin Kaminski*	**T06** How to Become a Community Rockstar—Learn How to Showcase Your Skills - *Cristal Kawula*	**T07** Backup, Distaster Recovery, and Business Continuity in Azure - *Orin Thomas*	**T08** Regex for Complete Noobs - *Thomas Rayner*	
11:00 AM	12:00 PM	**Keynote:** Windows Server 2019 Technical Foundation - *Jeff Woolsey, Principal Program Manager, Windows Server/Hybrid Cloud, Microsoft*				
12:00 PM	1:00 PM	Lunch - McKinley / Visit Exhibitors - Foyer				
1:00 PM	2:15 PM	**T09** Intune and Custom Policies - *Kevin Kaminski*	**T10** A World Beyond Passwords - *Lesha Bhansali & Kristina Cosgrave*	**T11** Configuring Windows Server & Client for Developing Hosted Linux Workloads - *Orin Thomas*	**T12** To Be Announced	
2:15 PM	2:45 PM	Sponsored Break - Visit Exhibitors - Foyer				
2:45 PM	4:00 PM	**T13** OneNote LifeHack: 5 Steps for Succeeding with Personal Productivity - *Ståle Hansen*	**T14** Hardening Windows Server - *Orin Thomas*	**T15** Familiar and Future-Proof: Lift and Shift Your Cluster into Azure IaaS - *Rob Hindman*	**T16** From Cmdlet to Function—Your PowerShell Beginnings - *Mike Nelson*	
4:00 PM	5:30 PM	Exhibitor Reception – Attend Exhibitor Demo - Foyer				
START TIME	**END TIME**	**TechMentor Day 2: Wednesday, August 7, 2019**				
7:30 AM	8:00 AM	Registration - Coffee and Light Breakfast				
8:00 AM	9:15 AM	**W01** Ten Practical Tips to Secure Your Corporate Data with Microsoft 365 - *Peter Daalmans*	**W02** Surviving a Ransomware Attack: Notes from the Field - *Émile Cabot*	**W03** Everything You Need to Know About Storage Spaces Direct—Part 1 - *Cosmos Darwin*	**W04** Azure is 100 % Highly Available or is it? - *Peter De Tender*	
9:30 AM	10:45 AM	**W05** Everything You Need to Know About Calling in Microsoft Teams - *Ståle Hansen*	**W06** Get Started with Azure MFA the Right Way - *Jan Ketil Skanke*	**W07** Everything You Need to Know About Storage Spaces Direct—Part 2 - *Cosmos Darwin*	**W08** Become the World's Greatest Azure-Bender - *Peter De Tender*	
11:00 AM	12:00 PM	**Panel Discussion:** The Future of IT - *Sami Laiho and Dave Kawula (Moderators); Peter De Tender, Thomas Maurer, John O'Neill Sr., and Orin Thomas*				
12:00 PM	1:00 PM	Birds-of-a-Feather Lunch - *McKinley - Visit Exhibitors - Foyer*				
1:00 PM	1:30 PM	Networking Break - Exhibitor Raffle @ 1:10 pm (Must be present to win) - *Foyer in front of Business Center*				
1:30 PM	2:45 PM	**W09** Build Your Azure Infrastructure Like a Pro - *Aleksandar Nikolic*	**W10** Implementing Proactive Security in the Cloud—Part 1 - *Sami Laiho*	**W11** Get the Best of Azure and Windows Server with Windows Admin Center - *Haley Rowland*	**W12** Oww Help! SCCM is Getting SaaSified - *Peter Daalmans*	
3:00 PM	4:15 PM	**W13** Mastering Azure Using Cloud Shell, PowerShell, and Bash! - *Thomas Maurer*	**W14** Implementing Proactive Security in the Cloud—Part 2 - *Sami Laiho*	**W15** Master Software Definded Networking in Windows Server 2019 - *Greg Cusanza*	**W16** Discussion of Modern Device Management from a (Microsoft Surface) Hardware Engineer - *Carl Luberti*	
6:15 PM	9:00 PM	Wednesday Event TBD				
START TIME	**END TIME**	**TechMentor Day 3: Thursday, August 8, 2019**				
7:30 AM	8:00 AM	Registration - Coffee and Light Breakfast				
8:00 AM	9:15 AM	**TH01** Hackers Won't Pass—Microsoft 365 Identity & Threat Protection—Part 1 - *Sergey Chubarov*	**TH02** Azure Cloud Shell: The Easiest Way to Manage Azure with Command-line Tools - *Aleksandar Nikolic*	**TH03** Replicate Your Storage/ Data to Azure the Easy Way - *Arpita Duppala*	**TH04** Android Enterprise Management with Intune - *Jan ketil Skanke*	
9:30 AM	10:45 AM	**TH05** Hackers Won't Pass—Microsoft 365 Identity & Threat Protection—Part 2 - *Sergey Chubarov*	**TH06** PowerShell Core on Linux: Benefits and Challenges - *Aleksandar Nikolic*	**TH07** A Look Into the Hybrid Cloud Lifestyle of an Azure Stack Operator - *Thomas Maurer*	**TH08** "AaronLocker:" Robust and Practical Application Whitelisting for Windows - *Aaron Margosis*	
11:00 AM	12:15 PM	**TH09** Using PowerShell to Rock Your Labs - *Dave Kawula*	**TH10** The Force Awakens—Azure SQL Server for the On-prem DBA - *Alexander Arvidsson*	**TH11** 12 Ways to Hack Multi Factor Authentication (MFA) - *Roger Grimes*	**TH12** Windows Autopilot: Modern Device Provisioning - *Michael Niehaus*	
12:15 PM	2:00 PM	Lunch @ The Mixer - Visit the Microsoft Company Store & Visitor Center				
2:00 PM	3:15 PM	**TH13** Migrating to Windows Server 2019 Like the Pro's - *Dave Kawula*	**TH14** Boring is Stable, Stable is Good—Best Practices in Practice for SQL Server - *Alexander Arvidsson*	**TH15** Start Using JEA Today to Stop Overprivileged User Accounts - *John O'Neill Sr.*	**TH16** Wireshark Essentials: Your First Day with Wireshark - *Richard Hicks*	
3:30 PM	4:45 PM	**TH17** The Case of the Shrinking Data: Data Deduplication in Windows Server 2019 - *Dave Kawula*	**TH18** Malware Protection in Windows 10 - *Émile Cabot*	**TH19** Supporting Surfaces? Learn the Surface Diagnostic Toolkit for Business Now - *John O'Neill Sr.*	**TH20** Always On VPN: The Good, The Bad, and the Ugly! - *Richard Hicks*	
START TIME	**END TIME**	**TechMentor Post-Conference Workshops: Friday, August 9, 2019** *(Separate entry fee required)*				
8:30 AM	9:00 AM	Post-Conference Workshop Registration - Coffee and Light Breakfast				
9:00 AM	12:00 PM	**F01** Workshop: Enhance Security While Increasing Your Admin Superpowers - *John O'Neill Sr.*		**F02** Workshop: Building Real World Labs in Azure - *Dave Kawula*		
12:00 PM	1:00 PM	Lunch - *McKinley*				
1:00 PM	4:00 PM	Workshop Continued - *John O'Neill Sr.*		Workshop Continued - *Dave Kawula*		

Logo denotes sessions with a Microsoft speaker.

Speakers and sessions subject to change

CONNECT WITH TECHMENTOR

Twitter
@TechMentorEvent

Facebook
Search "TechMentor"

LinkedIn
Search "TechMentor"

TechMentorEvents.com/MicrosoftHQ

Transformations are then applied via filters to try to match queries with the indexed data. You need to understand these if you want a deeper grasp of search engine technology. Luckily, you can start creating an application with just a high-level overview.

Understanding the Data

I need to understand the data in order to model the index. For this, I'll open Posts.xml and analyze it.

> There are several ways to specify the structure of the data for the index.

Here's what the data looks like. The posts node contains many row child nodes. Each child node corresponds to one record, with each field exported as an attribute in each child node. The data is pretty clean for ingesting into the search engine, which is good:

```
<posts>
  <row Id="5" PostTypeId="1" CreationDate="2014-05-13T23:58:30.457"
    Score="9" ViewCount="448" Body="<Contains the body of the question or answer>"
      OwnerUserId="5" LastActivityDate="2014-05-14T00:36:31.077"
        Title="How can I do simple machine learning without hard-coding behavior?"
          Tags="<machine-learning,artificial-intelligence>;" AnswerCount="4"
            CommentCount="5" FavoriteCount="1"
              ClosedDate="2014-05-14T14:40:25.950" />
  ...
</posts>
```

At a glance, I can quickly observe how there are fields of different types. There's a unique identifier, a few dates, a couple of numeric fields, some long and some short text fields, and some metadata fields. For brevity, I edited out the Body, which is a large text field, but all others are in their original state.

Configuring Solr to Use a Classic Schema

There are several ways to specify the structure of the data for the index. By default, Solr uses a managed schema, which means it uses schemaless mode. But I want to construct the schema manually—what's called a classic schema—so I need to make a few configuration changes. First, I'll create a folder to hold my index configuration, which I'll call msdnarticledemo. The folder is located in <solr>\server\solr\, where <solr> is the folder where I unzipped Solr.

Next, I create a text file at the root of this folder called core.properties, which needs only the following line added: name=msdnarticledemo. This file is used to create a Solr core, which is just a running instance of a Lucene index. You might hear the word collection, too, which can have a different meaning depending on the context. For my current intent and purposes, a core is the equivalent of an index.

I now need to copy the contents of a sample clean index to use as a base. Solr includes one in <solr>\server\solr\configsets_default. I copy the conf folder into msdnarticledemo.

In the very important next step, I tell Solr I want to use a classic schema; that is, that I'll manually edit my schema. To do this, I open solrconfig.xml, and add the following line:

```
<schemaFactory class="ClassicIndexSchemaFactory"/>
```

Additionally, still within this file, I comment out two nodes, the updateRequestProcessorChain with:

```
name="add-unknown-fields-to-the-schema"
```

and updateProcessor with:

```
name="add-schema-fields"
```

Those two features are what allow Solr to add new fields while indexing data in schemaless mode. I also remove the comment inside this xml node, given that "--" isn't allowed within xml comments.

Finally, I rename managed-schema to schema.xml. Just like that, Solr is ready to use a manually created schema.

Creating the Schema

The next step is to define the fields in the schema, so I open schema.xml and scroll down until I find the definition of id, _text_, and _root_.

This is how each field is defined, as a <field> xml node that contains:

- **name:** The name for each field.
- **type:** The type of the field; you can modify how each type is handled from within schema.xml.
- **indexed:** True indicates this field can be used for searching.
- **stored:** True indicates this field can be returned for display.
- **required:** True indicates this field must be indexed, else an error is raised.
- **multiValued:** True indicates this field may contain more than one value.

This may be confusing at first, but some fields can be displayed but not searched and some can be searched but might not be able to be retrieved after indexing. There are other advanced attributes, but I won't get into their details at this point.

Figure 2 shows how I define the fields in the schema for Posts. For text types I have a variety, including string, text_get_sort, and text_general. Text search is a major objective of search, hence the different text-capable types. I also have a date, an integer, a float and one field that holds more than one value, Tags.

What comes next may vary depending on the implementation. At the moment, I have many fields and I can specify which one I want to search on and how important that field is relative to the other fields.

But to start, I can use the catch-all field _text_ to perform a search over all my fields. I simply create a copyField and tell Solr that the data from all fields should be copied into my default field:

```
<copyField source="*" dest="_text_"/>
```

Figure 2 **Fields in Schema.xml**

```
<field name="id" type="string" indexed="true" stored="true"
  required="true" multiValued="false" />
<field name="postTypeId" type="pint" indexed="true" stored="true" />
<field name="title" type="text_gen_sort" indexed="true"
  stored="true" multiValued="false"/
<field name="body" type="text_general" indexed="false"
  stored="true" multiValued="false"/>
<field name="tags" type="string" indexed="true" stored="true"
  multiValued="true"/>
<field name="postScore" type="pfloat" indexed="true" stored="true"/>
<field name="ownerUserId" type="pint" indexed="true" stored="true" />
<field name="answerCount" type="pint" indexed="true" stored="true" />
<field name="commentCount" type="pint" indexed="true" stored="true" />
<field name="favoriteCount" type="pint" indexed="true" stored="true" />
<field name="viewCount" type="pint" indexed="true" stored="true" />
<field name="creationDate" type="pdate" indexed="true" stored="true" />
<field name="lastActivityDate" type="pdate" indexed="true" stored="true" />
<field name="closedDate" type="pdate" indexed="true" stored="true" />
```

Now, when I run a search, Solr will look within this field and return any document that matches my query.

Next, I restart Solr to load the core and apply the changes, by running this command:

```
> bin\solr.cmd restart
```

I'm now ready to start creating a C# application.

Getting SolrNet and Modeling the Data

Solr provides a REST-like API that you can easily use from any application. Even better, there's a library called SolrNet (bit.ly/2XwkROA) that provides an abstraction over Solr, allowing you to easily work with strongly typed objects, with plenty of functionality to make search application development faster.

The easiest way to get SolrNet is to install the SolrNet package from NuGet. I'll include the library in the new Console Application I created using Visual Studio 2017. You may also want to download additional packages, such as SolrCloud, that are required for using other inversion control mechanisms and for additional functionality.

Within my console application, I need to model the data in my index. This is quite straightforward: I simply create a new class file called Post.cs, as shown in **Figure 3**.

This is nothing more than a plain old CLR object (POCO) that represents each individual document in my index, but with an attribute that tells SolrNet to which field each property maps.

Figure 3 **Post Document Model**

```
class Post
{
  [SolrUniqueKey("id")]
  public string Id { get; set; }

  [SolrField("postTypeId")]
  public int PostTypeId { get; set; }

  [SolrField("title")]
  public string Title { get; set; }

  [SolrField("body")]
  public string Body { get; set; }

  [SolrField("tags")]
  public ICollection<string> Tags { get; set; } = new List<string>();

  [SolrField("postScore")]
  public float PostScore { get; set; }

  [SolrField("ownerUserId")]
  public int? OwnerUserId { get; set; }

  [SolrField("answerCount")]
  public int? AnswerCount { get; set; }

  [SolrField("commentCount")]
  public int CommentCount { get; set; }

  [SolrField("favoriteCount")]
  public int? FavoriteCount { get; set; }

  [SolrField("viewCount")]
  public int? ViewCount { get; set; }

  [SolrField("creationDate")]
  public DateTime CreationDate { get; set; }

  [SolrField("lastActivityDate")]
  public DateTime LastActivityDate { get; set; }

  [SolrField("closedDate")]
  public DateTime? ClosedDate { get; set; }
}
```

Figure 4 **Populating Fields**

```
Post post = new Post();

post.Id = node.Attributes["Id"].Value;

if (node.Attributes["Title"] != null)
{
  post.Title = node.Attributes["Title"].Value;
}

if (node.Attributes["Tags"] != null)
{
  post.Tags = node.Attributes["Tags"].Value.Split(new char[] { '<', '>' })
    .Where(t => !string.IsNullOrEmpty(t)).ToList();
}
// Add all other fields
```

Creating a Search Application

When you create a search application, you typically create two separate functionalities:

The indexer: This is the application I need to create first. Quite simply, to have data to search, I need to feed that data to Solr. This may involve reading the data in from multiple sources, converting it from various formats and more, until it's finally ready for search.

The search application: Once I have data in my index, I can start working on the search application.

With both, the first step requires initializing SolrNet, which you can do using the following line in the console application (make sure that Solr is running!):

```
Startup.Init<Post>("http://localhost:8983/solr/msdnarticledemo");
```

I'll create a class for each functionality in my application.

Building the Indexer

To index the documents, I start by obtaining the SolrNet service instance, which allows me to start any supported operations:

```
var solr = ServiceLocator.Current.GetInstance<ISolrOperations<Post>>();
```

Next, I need to read the contents of Posts.xml into an XMLDocument, which involves iterating over each node, creating a new Post object, extracting each attribute from the XMLNode and assigning it to the corresponding property.

Note that within the field of information retrieval, or search, data is stored denormalized. In contrast, when you're working with databases, you typically normalize data to avoid duplication. Instead of adding the name of the owner to a post, you add an integer Id and create a separate table to match the Id to a Name. In search, however, you add the name as part of the post, duplicating data. Why? Because when data is normalized, you need to perform joins for retrieval, which are quite expensive. But one of the main objectives of a search engine is speed. Users expect to push a button and get the results they want immediately.

Now back to creating the post object. In **Figure 4**, I'm going to show only three fields as adding others is quite easy. Please notice how Tags is multivalued and that I'm checking for null values to avoid exceptions.

Once I've populated the object, I can add each instance using the Add method:

```
solr.Add(post);
```

Alternatively, I can create a posts collection, and add posts in batches using AddRange:

```
solr.AddRange(post_list);
```

Either approach is fine, but it has been observed in many production deployments that adding documents in batches of 100 tends to help with performance. Note that adding a document doesn't make it searchable. I need to commit:

```
solr.Commit();
```

Now I'll execute and, depending on the amount of data being indexed and the machine on which it's running, it may take anywhere from a few seconds to a couple of minutes.

Once the process completes, I can navigate to the Solr Admin UI, look for a dropdown on the middle left that says Core Selector and pick my core (msdnarticledemo). From the Overview tab I can see the Statistics, which tell me how many documents I just indexed.

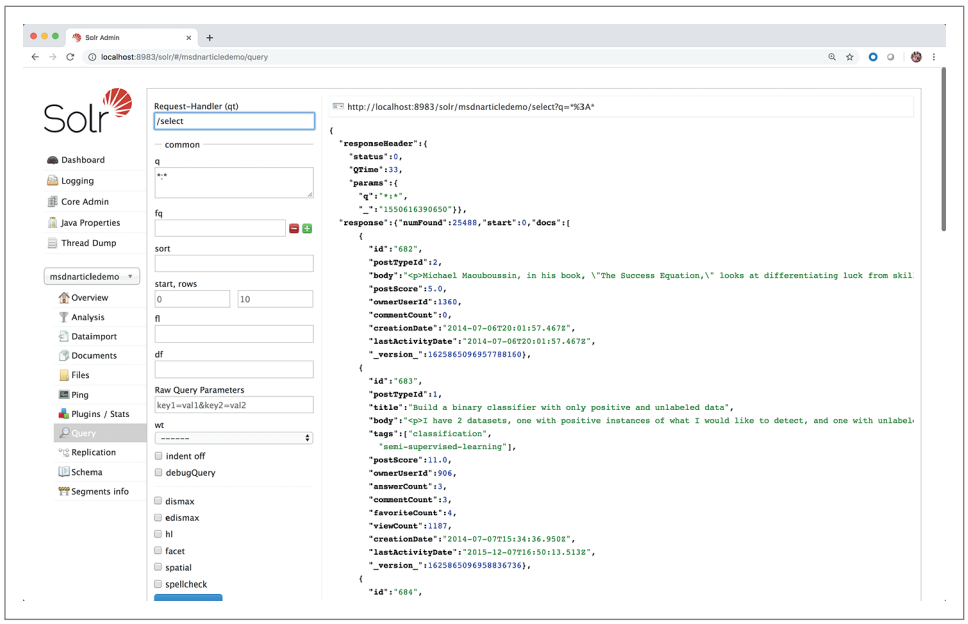


Figure 5 **A Query in Solr via the Admin UI**

In my data dump I had 25,488 posts, which matches what I see:

```
Statistics
  Last Modified: less than a minute ago
  Num Docs:25488
  Max Doc:25688
```

Now that I have data in my index, I'm ready to start working on the search side.

> The q parameter is probably the most important one, as it retrieves documents in order of relevance by calculating a score.

Searching in Solr

Before jumping back into Visual Studio, I want to demonstrate a quick search from Solr's Admin UI and explain some of the parameters that are available.

In the msdnarticledemo core, I'll click on Query and push the blue button at the bottom that says Execute Query. I get back all my documents in JSON format, as shown in **Figure 5**.

So, what exactly did I just do and why did I get back all documents in the index? The answer is simple. Take a look at the parameters—the column with Request-Handler (qt) at the top. As you can see, one parameter is labeled q, and it has a value of *:*. That's what brought up all the documents. In fact, the query I ran was to search all fields for all values, using a key-value pair. If instead I only wanted to search for Solr in the Title, then the q value would be Title:Solr.

This is quite useful for building more complicated queries that provide different weights for each field, which makes sense. A word or a phrase that's found in a Title is more important than one in the content. For example, if a document has Enterprise Search in the Title, it's highly likely that the entire document is going to be about enterprise search. But if I find this phrase in any part of the Body of a document, it may be just a reference to something barely related.

The q parameter is probably the most important one, as it retrieves documents in order of relevance by calculating a score. But there are a number of other parameters you can use via the Request Handler to configure how requests are processed by Solr, including filter query (fq), sort, field list (fl) and many more that you can find in the Solr documentation at bit.ly/2GVmYGl. With these parameters you can start building more complicated queries. It takes some time to master, but the more you learn, the better relevancy rankings you'll get.

Keep in mind that the Admin UI is not how an application works with Solr. For that purpose, the REST-like interface is used. If you look right above the results, there's a link in a gray box that contains the call for this particular query. Clicking on it opens a new window, which holds the response.

This is my query:

```
http://localhost:8983/solr/msdnarticledemo/select?q=*%3A*&wt=json
```

Figure 6 **Running a Basic Search**

```
QueryOptions query_options = new QueryOptions
{
  Rows = 10,
  StartOrCursor = new StartOrCursor.Start(0),
  FilterQueries = new ISolrQuery[] {
    new SolrQueryByField("postTypeId", "1"),
    }
};

// Construct the query
SolrQuery query = new SolrQuery(keywords);

// Run a basic keyword search, filtering for questions only
var posts = solr.Query(query, query_options);
```

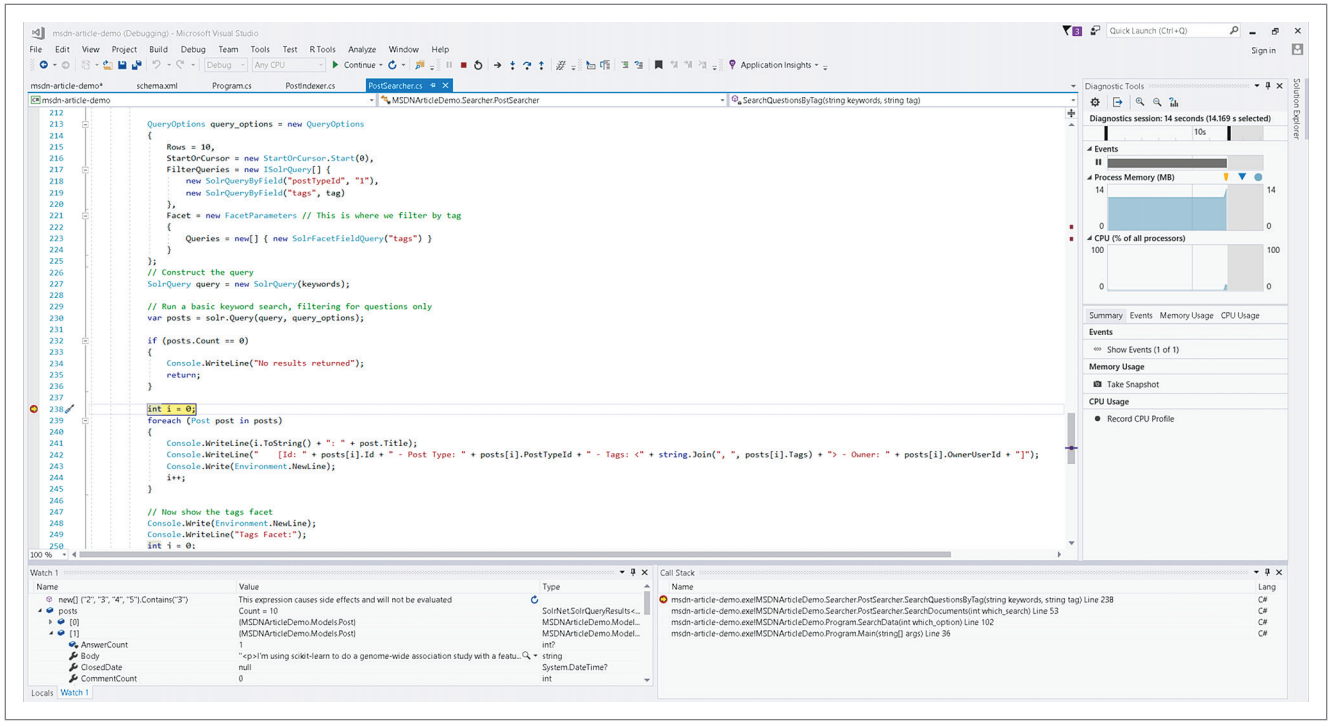


Figure 7 **A Sample Search Project**

SolrNet makes calls like this under the hood, but it presents objects I can use from my .NET application. Now I'll build a basic search application.

Building the Search Application

In this particular case, I'm going to search the questions in my dataset, in all fields. Given that the data contains both questions and answers, I'll filter by the PostTypeId, with "1," which means it's a question. To do this I use a filter query—the fq parameter.

Additionally, I'll set some query options to return one page of results at a time, namely Rows, to indicate how many results, and StartOrCursor, to specify the offset (start). And, of course, I'll set the query.

Figure 6 shows the code required to run a basic search, with query being the text for which I'm searching.

After running the query, I get posts, which is a SolrQueryResults object. It contains a collection of results, as well as many properties with multiple objects that provide additional functionality. Now that I have those results, I can display them to the user.

Narrowing Down the Results

In many cases, the original results can be good, but the user may want to narrow them by a particular metadata field. It's possible to drill down by a field using facets. In a facet, I get a list of key value pairs. For example, with Tags I get each tag and how many times each one occurs. Facets are commonly used with numeric, date or string fields. Text fields are tricky.

To enable facets, I need to add a new QueryOption, Facet:

```
Facet = new FacetParameters
{
  Queries = new[] {
    new SolrFacetFieldQuery("tags")
  }
}
```

Now I can retrieve my facets from posts.FacetFields["tags"], which is a collection of key-value pairs that contain each particular tag, and how many times each tag occurs in the result set.

Then I can allow the user to select which tags to drill into, reducing the number of results with a filter query, ideally returning relevant documents.

Improving Your Search—What's Next?

So far, I've covered the essentials of implementing basic search in C# with Solr and SolrNet using the questions from one of the StackExchange sites. However, this is just the start of a new journey where I can delve into the art of returning relevant results using Solr.

Some of the next steps include searching by individual fields with different weights; providing highlighting in results to show matches in the content; applying synonyms to return results that are related but may not contain the exact words that were searched; stemming, which is what reduces words to their base to increase recall; phonetic search, which helps international users; and more.

All in all, learning how to implement search is a valuable skill that can potentially yield valuable returns in your future as a developer.

To accompany this article, I've created a basic search project, as shown in **Figure 7**, that you can download to learn more about enterprise search. Happy searching! ■

Xavier Morera *helps developers understand enterprise search and Big Data. He creates courses at Pluralsight and sometimes at Cloudera. He worked for many years at Search Technologies (now part of Accenture), dealing with search implementations. He lives in Costa Rica, and you can find him at xaviermorera.com.*

Thanks to the following technical experts for reviewing this article: *Jose Arias (Accenture), Jonathan Gonzalez (Accenture)*

Visual Studio LIVE!

EXPERT SOLUTIONS FOR ENTERPRISE DEVELOPERS

April 22-26, 2019 | Hyatt Regency New Orleans

Spice Up Your Coding Skills in the Bayou

New Orleans

Intense Developer Training Conference

In-depth Technical Content On:

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

Secure Your Seat Today!

Use Promo Code MSDN

YOUR ADVENTURE STARTS HERE

vslive.com/neworleans

SUPPORTED BY

Microsoft | Visual Studio | msdn magazine | Visual Studio MAGAZINE

PRODUCED BY

CONVERGE 360 an 1105 MEDIA company

Agenda-at-a-Glance

#VSLive

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

START TIME	END TIME				
		Pre-Conference Full Day Hands-On Labs: Monday, April 22, 2019 *(Separate entry fee required)*			
8:00 AM	9:00 AM	Pre-Conference Workshop Registration - Coffee and Morning Pastries			
9:00 AM	6:00 PM	**HOL01** Full Day Hands-On Lab: Cross-Platform Mobile Development in a Day with Xamarin and Xamarin.Forms - *Marcel de Vries & Roy Cornelissen*		**HOL02** Full Day Hands-On Lab: Building a Modern DevOps Pipeline on Microsoft Azure with ASP.NET Core and Azure DevOps - *Brian Randell & Mickey Gousset*	
6:45 PM	9:00 PM	Dine-A-Round			
START TIME	**END TIME**	**Day 1: Tuesday, April 23, 2019**			
7:00 AM	8:00 AM	Registration - Coffee and Morning Pastries			
8:00 AM	9:15 AM	**T01** Moving to ASP.NET Core 2.X - *Philip Japikse*	**T02** Building Your First Mobile App with Xamarin Forms - *Robert Green*	**T03** Crack the Code: How Machine Learning Models Work - *Jen Underwood*	**T04** Unit Testing Makes Me Faster: Convincing Your Boss, Your Co-Workers, and Yourself - *Jeremy Clark*
9:30 AM	10:45 AM	**T05** Getting Started with ASP.NET Core 2.0 Razor Pages - *Walt Ritscher*	**T06** (WPF + WinForms) * .NET Core = Modern Desktop - *Oren Novotny*	**T07** Containers Demystified - *Robert Green*	**T08** Azure DevOps in the Cloud and in Your Data Center - *Brian Randell*
11:00 AM	12:00 PM	**Keynote:** Learn About Mobile DevOps with Xamarin, App Center and Azure DevOps - *Donovan Brown, Principal DevOps Manager, Cloud Developer Advocacy Team, Microsoft*			
12:00 PM	1:00 PM	Lunch			
1:00 PM	1:30 PM	Dessert Break - Visit Exhibitors			
1:30 PM	2:45 PM	**T09** Diving Deep Into ASP.NET Core 2.x - *Philip Japikse*	**T10** Busy Developer's Guide to Flutter - *Ted Neward*	**T11** How to Avoid Building Bad Predictive Models - *Jen Underwood*	**T12** Cross-Platform Development with Xamarin, C#, and CSLA .NET - *Rockford Lhotka*
3:00 PM	4:15 PM	**T13** Up and Running with Angular in 60 Minutes - *Justin James*	**T14** Programming with PowerApps and Microsoft Flow - *Walt Ritscher*	**T15** Busy .NET Developer's Guide to Python - *Ted Neward*	**T16** Get Func-y: Understanding Delegates in .NET - *Jeremy Clark*
4:15 PM	5:30 PM	Welcome Reception			
START TIME	**END TIME**	**Day 2: Wednesday, April 24, 2019**			
7:30 AM	8:00 AM	Registration - Coffee and Morning Pastries			
8:00 AM	9:15 AM	**W01** Angular Unit Testing from the Trenches - *Justin James*	**W02** Power BI: What Have You Done for Me Lately? - *Andrew Brust*	**W03** Secure Your App with Azure AD B2C - *Oren Novotny*	**W04** Putting the Ops into DevOps - *Mickey Gousset*
9:30 AM	10:45 AM	**W05** TypeScript: Moving Beyond the Basics - *Allen Conway*	**W06** AI and Analytics with Apache Spark on Azure Databricks - *Andrew Brust*	**W07** Architecting and Developing Microservices Apps - *Eric D. Boyd*	**W08** Architecting Systems for DevOps and Continuous Delivery - *Marcel de Vries*
11:00 AM	12:00 PM	**General Session:** Azure and the Internet of Things—Why Modern Developers Should Care - *Peter Provost, Principal PM Manager for Azure IoT Central, Microsoft*			
12:00 PM	1:00 PM	Birds-of-a-Feather Lunch			
1:00 PM	1:30 PM	Dessert Break - Visit Exhibitors - Exhibitor Raffle @ 1:15pm (Must be present to win)			
1:30 PM	1:50 PM	**W09** Fast Focus: Hybrid Web Frameworks - *Allen Conway*	**W10** Fast Focus: Graph DB from SQL to Cosmos - *Leonard Lobel*	**W11** Fast Focus: What's New in EF Core 2.x - *Jim Wooley*	
2:00 PM	2:20 PM	**W12** Fast Focus: Ultimate Presentation Formula for Nerds - *Justin James*	**W13** Fast Focus: Serverless of Azure 101 - *Eric D. Boyd*	**W14** Fast Focus: Scrum in 20 Minutes - *Benjamin Day*	
2:30 PM	3:45 PM	**W15** Migrating from AngularJS to Angular + TypeScript - *Allen Conway*	**W16** Introduction to Azure Cosmos DB - *Leonard Lobel*	**W17** Demystifying Microservice Architecture - *Miguel Castro*	**W18** From One Release per Quarter to 30 Times a Day - *Marcel de Vries*
4:00 PM	5:15 PM	**W19** Getting Pushy with SignalR and Reactive Extensions - *Jim Wooley*	**W20** Modern SQL Server Security Features for Developers - *Leonard Lobel*	**W21** Make Your App See, Hear and Think with Cognitive Services - *Roy Cornelissen*	**W22** Monitor Your Applications and Infrastructure - *Eric D. Boyd*
6:00 PM	7:30 PM	VSLive!'s Second Line Parade to Bourbon Street			
START TIME	**END TIME**	**Day 3: Thursday, April 25, 2019**			
7:30 AM	8:00 AM	Registration - Coffee and Morning Pastries			
8:00 AM	9:15 AM	**TH01** Advanced Fiddler Techniques - *Robert Boedigheimer*	**TH02** Diagnosing and Debugging ASP.NET Core Apps in Production - *Paul Yuknewicz & Catherine Wang*	**TH03** UX Design Fundamentals: What Do Your Users Really See? - *Billy Hollis*	**TH04** Azure DevOps and AKS - *Brian Randell*
9:30 AM	10:45 AM	**TH05** SASS and CSS for Developers - *Robert Boedigheimer*	**TH06** The Next Frontier - Conversational Bots - *Sam Basu*	**TH07** Porting Your Code from .NET Framework to .NET Standard - *Rockford Lhotka*	**TH08** Calling All Developers - Get Your Bason On with WSL - *Brian Randell*
11:00 AM	12:15 PM	**TH09** Upload and Store a File Using MVC - *Paul Sheriff*	**TH10** Essential Tools for Xamarin Developers! - *Sam Basu*	**TH11** What's New in C# 8 - *Jason Bock*	**TH12** How to Interview a Developer - *Billy Hollis*
12:15 PM	1:30 PM	Lunch			
1:30 PM	2:45 PM	**TH1** Blazing the Web - Building Web Applications in C# - *Jason Bock*	**TH14** Entity Framework for Enterprise Applications - *Benjamin Day*	**TH15** Exposing an Extensibility API for your Applications - *Miguel Castro*	**TH16** To Be Announced
3:00 PM	4:15 PM	**TH17** What's New in Bootstrap 4 - *Paul Sheriff*	**TH18** To Be Announced	**TH19** Unit Testing & Test-Driven Development (TDD) for Mere Mortals - *Benjamin Day*	**TH20** Improving Code Quality with Static Analyzers - *Jim Wooley*
START TIME	**END TIME**	**Post-Conference Workshops: Friday, April 26, 2019** *(Separate entry fee required)*			
7:30 AM	8:00 AM	Post-Conference Workshop Registration - Coffee and Morning Pastries			
8:00 AM	5:00 PM	**F01** Workshop: DI for the Dev Guy - *Miguel Castro*	**F02** Workshop: SQL Server for Developers: The Grand Expedition - *Andrew Brust & Leonard Lobel*	**F03** Workshop: Cross-Platform C# Using .NET Core, Kubernetes, and WebAssembly - *Rockford Lhotka & Jason Bock*	

Speakers and sessions subject to change

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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

JAMES MCCAFFREY

Neural Anomaly Detection Using PyTorch

Anomaly detection, also called outlier detection, is the process of finding rare items in a dataset. Examples include identifying malicious events in a server log file and finding fraudulent online advertising.

A good way to see where this article is headed is to take a look at the demo program in **Figure 1**. The demo analyzes a 1,000-item subset of the well-known Modified National Institute of Standards and Technology (MNIST) dataset. Each data item is a 28x28 grayscale image (784 pixels) of a handwritten digit from zero to nine. The full MNIST dataset has 60,000 training images and 10,000 test images.

The demo program creates and trains a 784-100-50-100-784 deep neural autoencoder using the PyTorch code library. An autoencoder is a neural network that learns to predict its input. After training, the demo scans through 1,000 images and finds the one image that's most anomalous, where most anomalous means highest reconstruction error. The most anomalous digit is a three that looks like it could be an eight instead.

This article assumes you have intermediate or better programming skill with a C-family language and a basic familiarity with machine learning, but doesn't assume you know anything about autoencoders. All the demo code is presented in this article. The code and data are also available in the accompanying download. All normal error checking has been removed to keep the main ideas as clear as possible.

Installing PyTorch

PyTorch is a relatively low-level code library for creating neural networks. It's roughly similar in terms of functionality to TensorFlow and CNTK. PyTorch is written in C++, but has a Python language API for easier programming.

Installing PyTorch includes two main steps. First, you install Python and several required auxiliary packages, such as NumPy and SciPy. Then you install PyTorch as a Python add-on package. Although it's possible to install Python and the packages required to run PyTorch separately, it's much better to install a Python distribution, which is a collection containing the base Python interpreter and additional packages that are compatible with each other. For my demo, I installed the Anaconda3 5.2.0 distribution, which contains Python 3.6.5.

After installing Anaconda, I went to the pytorch.org Web site and selected the options for the Windows OS, Pip installer, Python 3.6 and no CUDA GPU version. This gave me a URL that pointed to the corresponding .whl (pronounced "wheel") file, which I downloaded to my local machine. If you're new to the Python ecosystem, you can think of a Python .whl file as somewhat similar to a Windows .msi file. In my case, I downloaded PyTorch version 1.0.0. I opened a command shell, navigated to the directory holding the .whl file and entered the command:

```
pip install torch-1.0.0-cp36-cp36m-win_amd64.whl
```

The Demo Program

The complete demo program, with a few minor edits to save space, is presented in **Figure 2**. I indent with two spaces rather than the usual four spaces to save space. Note that Python uses the "\" character for line continuation. I used Notepad to edit my program. Most of my colleagues prefer a more sophisticated editor, but I like the brutal simplicity of Notepad.

The demo program starts by importing the NumPy, PyTorch and Matplotlib packages. The Matplotlib package is used to visually display the most anomalous digit that's found by the model. An alternative to importing the entire PyTorch package is to import just the necessary modules, for example, import torch.optim as opt.

Loading Data into Memory

Working with the raw MNIST data is rather difficult because it's saved in a proprietary, binary format. I wrote a utility program to extract the first 1,000 items from the 60,000 training items. I saved the data as mnist_pytorch_1000.txt in a Data subdirectory.

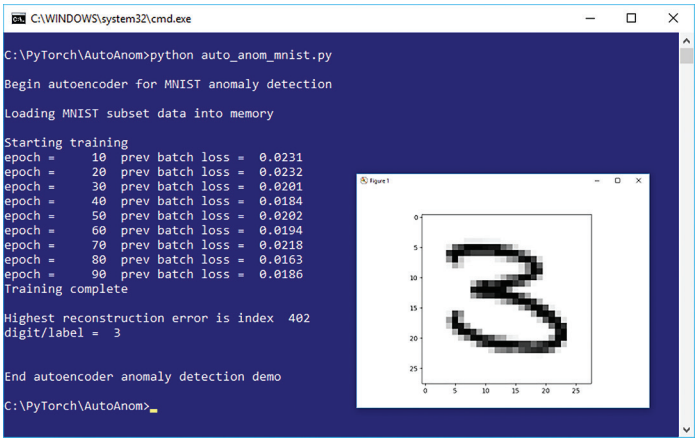


Figure 1 **MNSIT Image Anomaly Detection Using Keras**

Code download available at msdn.com/magazine/0419magcode.

The resulting data looks like this:

```
7 = 0 255 67 . . 123
2 = 113 28 0 . . 206
...
9 = 0 21 110 . . 254
```

Each line represents one digit. The first value on each line is the digit. The second value is an arbitrary equal-sign character just for readability. The next 28x28 = 784 values are grayscale pixel values between zero and 255. All values are separated by a single blank-space character. **Figure 3** shows the data item at index [30] in the data file, which is a typical "3" digit.

The dataset is loaded into memory with these statements:

```
data_file = ".\\Data\\mnist_pytorch_1000.txt"
data_x = np.loadtxt(data_file, delimiter=" ",
  usecols=range(2,786), dtype=np.float32)
labels = np.loadtxt(data_file, delimiter=" ",
  usecols=[0], dtype=np.float32)
norm_x = data_x / 255
```

Figure 2 **The Anomaly Detection Demo Program**

```
# auto_anom_mnist.py
# PyTorch 1.0.0 Anaconda3 5.2.0 (Python 3.6.5)
# autoencoder anomaly detection on MNIST

import numpy as np
import torch as T
import matplotlib.pyplot as plt

# ----------------------------------------------------------

def display(raw_data_x, raw_data_y, idx):
  label = raw_data_y[idx]  # like '5'
  print("digit/label = ", str(label), "\n")
  pixels = np.array(raw_data_x[idx])  # target row of pixels
  pixels = pixels.reshape((28,28))
  plt.rcParams['toolbar'] = 'None'
  plt.imshow(pixels, cmap=plt.get_cmap('gray_r'))
  plt.show()

# ----------------------------------------------------------

class Batcher:
  def __init__(self, num_items, batch_size, seed=0):
    self.indices = np.arange(num_items)
    self.num_items = num_items
    self.batch_size = batch_size
    self.rnd = np.random.RandomState(seed)
    self.rnd.shuffle(self.indices)
    self.ptr = 0

  def __iter__(self):
    return self

  def __next__(self):
    if self.ptr + self.batch_size > self.num_items:
      self.rnd.shuffle(self.indices)
      self.ptr = 0
      raise StopIteration  # ugh.
    else:
      result = self.indices[self.ptr:self.ptr+self.batch_size]
      self.ptr += self.batch_size
      return result

# ----------------------------------------------------------

class Net(T.nn.Module):
  def __init__(self):
    super(Net, self).__init__()
    self.layer1 = T.nn.Linear(784, 100)  # hidden 1
    self.layer2 = T.nn.Linear(100, 50)
    self.layer3 = T.nn.Linear(50,100)
    self.layer4 = T.nn.Linear(100, 784)

    T.nn.init.xavier_uniform_(self.layer1.weight)  # glorot
    T.nn.init.zeros_(self.layer1.bias)
    T.nn.init.xavier_uniform_(self.layer2.weight)
    T.nn.init.zeros_(self.layer2.bias)
    T.nn.init.xavier_uniform_(self.layer3.weight)
    T.nn.init.zeros_(self.layer3.bias)
    T.nn.init.xavier_uniform_(self.layer4.weight)
    T.nn.init.zeros_(self.layer4.bias)

  def forward(self, x):
    z = T.tanh(self.layer1(x))
    z = T.tanh(self.layer2(z))
    z = T.tanh(self.layer3(z))
    z = T.tanh(self.layer4(z))  # consider none or sigmoid
    return z

# ----------------------------------------------------------

def main():
  # 0. get started
  print("Begin autoencoder for MNIST anomaly detection")
  T.manual_seed(1)
  np.random.seed(1)

  # 1. load data
  print("Loading MNIST subset data into memory ")
  data_file = ".\\Data\\mnist_pytorch_1000.txt"
  data_x = np.loadtxt(data_file, delimiter=" ",
    usecols=range(2,786), dtype=np.float32)
  labels = np.loadtxt(data_file, delimiter=" ",
    usecols=[0], dtype=np.float32)
  norm_x = data_x / 255

  # 2. create autoencoder model
  net = Net()

  # 3. train autoencoder model
  net = net.train()  # explicitly set
  bat_size = 40
  loss_func = T.nn.MSELoss()
  optimizer = T.optim.Adam(net.parameters(), lr=0.01)
  batcher = Batcher(num_items=len(norm_x),
    batch_size=bat_size, seed=1)
  max_epochs = 100

  print("Starting training")
  for epoch in range(0, max_epochs):
    if epoch > 0 and epoch % (max_epochs/10) == 0:
      print("epoch = %6d" % epoch, end="")
      print("  prev batch loss = %7.4f" % loss_obj.item())

    for curr_bat in batcher:
      X = T.Tensor(norm_x[curr_bat])
      optimizer.zero_grad()
      oupt = net(X)
      loss_obj = loss_func(oupt, X)  # note X not Y
      loss_obj.backward()
      optimizer.step()
  print("Training complete")

  # 4. analyze - find item(s) with large(st) error
  net = net.eval()  # not needed - no dropout
  X = T.Tensor(norm_x)  # all input item as Tensors
  Y = net(X)            # all outputs as Tensors

  N = len(data_x)
  max_se = 0.0; max_ix = 0
  for i in range(N):
    curr_se = T.sum((X[i]-Y[i])*(X[i]-Y[i]))
    if curr_se.item() > max_se:
      max_se = curr_se.item()
      max_ix = i

  raw_data_x = data_x.astype(np.int)
  raw_data_y = labels.astype(np.int)
  print("Highest reconstruction error is index ", max_ix)
  display(raw_data_x, raw_data_y, max_ix)

  print("End autoencoder anomaly detection demo ")

# ----------------------------------------------------------

if __name__ == "__main__":
  main()
```

Notice the digit/label is in column zero and the 784 pixel values are in columns two to 785. After all 1,000 images are loaded into memory, a normalized version of the data is created by dividing each pixel value by 255 so that the scaled pixel values are all between 0.0 and 1.0.

Defining the Autoencoder Model

The demo program defines a 784-100-50-100-784 autoencoder. The number of nodes in the input and output layers (784) is determined by the data, but the number of hidden layers and the number of nodes in each layer are hyperparameters that must be determined by trial and error.

The demo program uses a program-defined class, Net, to define the layer architecture and the input-output mechanism of the autoencoder. An alternative is to create the autoencoder directly by using the Sequence function, for example:

```
net = T.nn.Sequential(
  T.nn.Linear(784,100), T.nn.Tanh(),
  T.nn.Linear(100,50), T.nn.Tanh(),
  T.nn.Linear(50,100), T.nn.Tanh(),
  T.nn.Linear(100,784), T.nn.Tanh())
```

The weight initialization algorithm (Glorot uniform), the hidden layer activation function (tanh) and the output layer activation function (tanh) are hyperparameters. Because all input and output values are between 0.0 and 1.0 for this problem, logistic sigmoid is a good alternative to explore for output activation.

Training and Evaluating the Autoencoder Model

The demo program prepares training with these statements:

```
net = net.train()  # explicitly set
bat_size = 40
loss_func = T.nn.MSELoss()
optimizer = T.optim.Adam(net.parameters(), lr=0.01)
batcher = Batcher(num_items=len(norm_x),
  batch_size=bat_size, seed=1)
max_epochs = 100
```

Because the demo autoencoder doesn't use dropout or batch normalization, it isn't necessary to explicitly set the network into train mode, but in my opinion it's good style to do so. The batch size (40), training optimization algorithm (Adam), initial learning rate (0.01) and maximum number of epochs (100) are all hyperparameters. If you're new to neural machine learning, you might be thinking, "Neural networks sure have a lot of hyperparameters," and you'd be correct.

> After the autoencoder model has been trained, the idea is to find data items that are difficult to correctly predict or, equivalently, items that are difficult to reconstruct.

The program-defined Batcher object serves up the indices of 40 random data items at a time until all 1,000 items have been processed (one epoch). An alternative approach is to use the built-in Dataset and DataLoader objects in the torch.utils.data module.

The structure of the training process is:

```
for epoch in range(0, max_epochs):
  # print loss every 10 epochs
  for curr_bat in batcher:
    X = T.Tensor(norm_x[curr_bat])
    optimizer.zero_grad()
    oupt = net(X)
    loss_obj = loss_func(oupt, X)
    loss_obj.backward()
    optimizer.step()
```

Each batch of items is created using the Tensor constructor, which uses torch.float32 as the default data type. Notice the loss_func function compares computed outputs to the inputs, which has the effect of training the network to predict its input values.

After training, you'll usually want to save the model, but that's a bit outside the scope of this article. The PyTorch documentation has good examples that show how to save a trained model in several different ways.

When working with autoencoders, in most situations (including this example) there's no inherent definition of model accuracy. You must determine how close computed output values must be to the associated input values in order to be counted as a correct prediction, and then write a program-defined function to compute your accuracy metric.

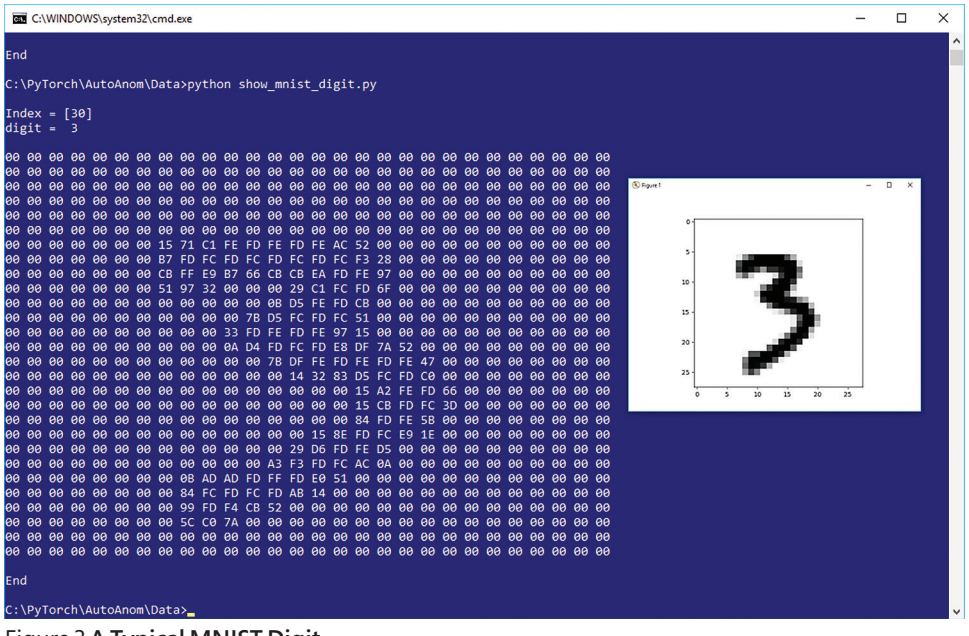


Figure 3 **A Typical MNIST Digit**

Using the Autoencoder Model to Find Anomalous Data

After the autoencoder model has been trained, the idea is to find data items that are difficult to correctly predict or, equivalently, items that are difficult to reconstruct. The demo code scans through all 1,000 data items and calculates the squared difference between the normalized input values and the computed output values like this:

```
net = net.eval()  # not needed - no dropout
X = T.Tensor(norm_x)  # all input item as Tensors
Y = net(X)            # all outputs as Tensors
N = len(data_x)
max_se = 0.0; max_ix = 0
for i in range(N):
  curr_se = T.sum((X[i]-Y[i])*(X[i]-Y[i]))
  if curr_se.item() > max_se:
    max_se = curr_se.item()
    max_ix = i
```

The maximum squared error (max_se) is calculated and the index of the associated image (max_ix) is saved. An alternative to finding the single item with the largest reconstruction error is to save all squared errors, sort them and return the top-n items where the value of n will depend on the particular problem you're investigating.

After the single-most-anomalous data item has been found, it's shown using the program-defined display function:

```
raw_data_x = data_x.astype(np.int)
raw_data_y = labels.astype(np.int)
print("Highest reconstruction error is index ", max_ix)
display(raw_data_x, raw_data_y, max_ix)
```

The pixel and label values are converted from type float32 to int mostly as a matter of principle because the Matplotlib imshow function inside the program-defined display function can accept either data type.

Wrapping Up

Anomaly detection using a deep neural autoencoder, as presented in this article, is not a well-investigated technique. A big advantage of using a neural autoencoder compared to most standard clustering techniques is that neural techniques can handle non-numeric data by encoding that data. Most clustering techniques depend on a numeric measure, such as Euclidean distance, which means the source data must be strictly numeric.

A related but also little-explored technique for anomaly detection is to create an autoencoder for the dataset under investigation. Then, instead of using reconstruction error to find anomalous data, you can cluster the data using a standard algorithm such as k-means because the innermost hidden layer nodes hold a strictly numeric representation of each data item. After clustering, you can look for clusters that have very few data items, or look for data items within clusters that are most distant from their cluster centroid. This approach has characteristics that resemble neural word embedding, where words are converted to numeric vectors that can then be used to compute a distance measure between words. ■

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

Thanks to the following Microsoft technical experts who reviewed this article: *Chris Lee, Ricky Loynd*

Visual Studio LIVE!

EXPERT SOLUTIONS FOR ENTERPRISE DEVELOPERS

June 9-13, 2019 | Hyatt Regency Cambridge

SPARK YOUR CODE REVOLUTION IN HISTORIC BOSTON

INTENSE DEVELOPER TRAINING CONFERENCE

In-depth Technical Content On:

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

Register by April 19 & Save Up To $400!

Use Promo Code MSDN

Your Adventure Starts Here!

vslive.com/boston

SUPPORTED BY

Microsoft | Visual Studio | msdn magazine | Visual Studio MAGAZINE

PRODUCED BY

CONVERGE 360 An 1105 MEDIA company

AGENDA AT-A-GLANCE

#VSLive

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

START TIME	END TIME				
		Pre-Conference Full Day Hands-On Labs: Sunday, June 9, 2019 *(Separate entry fee required)*			
7:30 AM	9:00 AM	Pre-Conference Workshop Registration - Coffee and Morning Pastries			
9:00 AM	6:00 PM	**HOL01** Full Day Hands-On Lab: Cross-Platform Mobile Development in a Day with Xamarin and Xamarin.Forms - *Marcel de Vries & Roy Cornelissen*		**HOL02** Full Day Hands-On Lab: Building a Modern DevOps Pipeline on Microsoft Azure with ASP.NET Core and Azure DevOps - *Brian Randell & Mickey Gousset*	
6:45 PM	9:00 PM	Dine-A-Round			
START TIME	**END TIME**	**Post-Conference Workshops: Monday, June 10, 2019** *(Separate entry fee required)*			
7:30 AM	8:00 AM	Post-Conference Workshop Registration - Coffee and Morning Pastries			
8:00 AM	5:00 PM	**M01** Workshop: DI for the Dev Guy - *Miguel Castro*	**M02** Workshop: SQL Server for Developers: The Grand Expedition - *Andrew Brust & Leonard Lobel*	**M03** Workshop: Cross-Platform C# Using .NET Core, Kubernetes, and WebAssembly - *Rockford Lhotka & Jason Bock*	
START TIME	**END TIME**	**Day 1: Tuesday, June 11, 2019**			
7:00 AM	8:00 AM	Registration - Coffee and Morning Pastries			
8:00 AM	9:15 AM	**T01** Moving to ASP.NET Core 2.X - *Philip Japikse*	**T02** State of Mobile Development - *Sam Basu*	**T03** Introduction to Azure Cosmos DB - *Leonard Lobel*	**T04** Azure DevOps in the Cloud and in Your Data Center - *Brian Randell*
9:30 AM	10:45 AM	**T05** Angular 101 - *Deborah Kurata*	**T06** Xamarin.Forms Takes You Places! - *Sam Basu*	**T07** Containers Demystified - *Robert Green*	**T08** What's New in C# 8 - *Jason Bock*
11:00 AM	12:00 PM	**KEYNOTE:** Responsible Tech in the Era of AI - *Tim O'Brien, General Manager, AI Programs, Microsoft*			
12:00 PM	1:00 PM	Lunch			
1:00 PM	1:30 PM	Dessert Break - Visit Exhibitors			
1:30 PM	2:45 PM	**T13** N Things You Didn't Know About the Router - *Deborah Kurata*	**T10** To Be Announced	**T11** Modern SQL Server Security Features for Developers - *Leonard Lobel*	**T12** Get Started with Git - *Robert Green*
3:00 PM	4:15 PM	**T09** TypeScript: The Future of Front End Web Development. - *Ben Hoelting*	**T14** Make Your App See, Hear and Think with Cognitive Services - *Roy Cornelissen*	**T15** Microservices with Azure Kubernetes Service (AKS) - *Vishwas Lele*	**T16** Creating Reactive Applications in .NET - *Jason Bock*
4:15 PM	5:30 PM	Welcome Reception			
START TIME	**END TIME**	**Day 2: Wednesday, June 12, 2019**			
7:30 AM	8:00 AM	Registration - Coffee and Morning Pastries			
8:00 AM	9:15 AM	**W01** JavaScript Patterns for the C# Developer - *Ben Hoelting*	**W02** Moving to Entity Framework Core 2.x - *Philip Japikse*	**W03** Microservices and Containers with Service Fabric and Azure Service Fabric Mesh - *Eric D. Boyd*	**W04** Writing Maintainable Test Automation - *Marcel de Vries*
9:30 AM	10:45 AM	**W05** C# in the Browser with Blazor and WebAssembly - *Rockford Lhotka*	**W06** Busy Developer's Guide to NoSQL - *Ted Neward*	**W07** What Every Developer Needs to Know About Deep Learning? - *Vishwas Lele*	**W08** Architecting Systems for DevOps and Continuous Delivery - *Marcel de Vries*
11:00 AM	12:00 PM	**General Session:** To Be Announced			
12:00 PM	1:00 PM	Birds-of-a-Feather Lunch			
1:00 PM	1:30 PM	Dessert Break - Visit Exhibitors - Exhibitor Raffle @ 1:15pm (Must be present to win)			
1:30 PM	1:50 PM	**W09** Fast Focus: Getting Started with ASP.NET Core 2.0 Razor Pages - *Walt Ritscher*	**W10** Fast Focus: Python for .NET Devs - *Ted Neward*	**W11** Fast Focus: Porting Your Code from .NET to Netstandard - *Rockford Lhotka*	
2:00 PM	2:20 PM	**W12** Fast Focus: What is WSL and Why Do I Care? - *Brian Randell*	**W13** Fast Focus: Serverless Computing: Azure Functions and Xamarin in 20 Minutes - *Laurent Bugnion*	**W14** Fast Focus: Scrum in 20 Minutes - *Benjamin Day*	
2:30 PM	3:45 PM	**W15** Add Native to Cross Platform with Xamarin.Essentials - *Veronika Kolesnikova*	**W16** AI and Analytics with Apache Spark on Azure Databricks - *Andrew Brust*	**W17** Go Serverless with Azure Functions - *Eric D. Boyd*	**W18** CI/CID with Azure DevOps - *Tiago Pascoal*
4:00 PM	5:15 PM	**W19** Now, the Two Worlds Collided - *Veronika Kolesnikova & Willy Ci*	**W20** Power BI: What Have You Done For Me Lately - *Andrew Brust*	**W21** Busy Developer's Guide to Naked Objects - *Ted Neward*	**W22** Scrum Under a Waterfall - *Benjamin Day*
6:15 PM	8:00 PM	VSLive! Trolley Tour of Historic Boston			
START TIME	**END TIME**	**Day 3: Thursday, June 13, 2019**			
7:30 AM	8:00 AM	Registration - Coffee and Morning Pastries			
8:00 AM	9:15 AM	**TH01** Upload and Store a File Using MVC - *Paul Sheriff*	**TH02** Taking Advantage of AI Easily with Azure Cognitive Services - *Laurent Bugnion*	**TH03** UX Design Fundamentals: What Do Your Users Really See? - *Billy Hollis*	**TH04** Bolting Security Into Your Development Process - *Tiago Pascoal*
9:30 AM	10:45 AM	**TH05** Getting Pushy with SignalR and Reactive Extensions - *Jim Wooley*	**TH06** (WPF + WinForms) * .NET Core = Modern Desktop - *Oren Novotny*	**TH07** How to Interview a Developer - *Billy Hollis*	**TH08** Database DevOps - *Brian Randell*
11:00 AM	12:15 PM	**TH09** What's New in Bootstrap 4 - *Paul Sheriff*	**TH10** Integrate Voice into Your Applications - *Walt Ritscher*	**TH11** Unit Testing & Test-Driven Development (TDD) for Mere Mortals - *Benjamin Day*	**TH12** .NET Standard, .NET Core, Why and How? - *Laurent Bugnion*
12:15 PM	1:30 PM	Lunch			
1:30 PM	2:45 PM	**TH13** Understanding ASP.NET Core Security - *Roland Guijt*	**TH14** Improving Code Quality with Static Analyzers - *Jim Wooley*	**TH15** Offices are Over: How & Why to Work Remotely in the Digital Age - *Laura B. Janusek*	**TH16** Get Func-y: Understanding Delegates in .NET - *Jeremy Clark*
3:00 PM	4:15 PM	**TH17** When Azure AD is Not Enough: Creating a Token Service in ASP.NET Core 2.2 - *Roland Guijt*	**TH18** Signing Your Code the Easy Way - *Oren Novotny*	**TH19** Get Your Poker Face On: How to Use Planning Poker to Slay Project Estimations - *Laura B. Janusek*	**TH20** I'll Get Back to You: Task, Await, and Asynchronous Methods in C# - *Jeremy Clark*

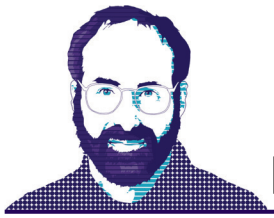
Speakers and sessions subject to change

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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



A Laughing Matter

I use a lot of humor in my teaching and writing. I find that readers and students remember a good joke or anecdote far better than any pontificated principle. But my most recent course on Xamarin included eight students with limited English skills. They mostly followed my code presentations, but I had to explain all the jokes. ("Elephants don't really wear sneakers, see?"). I lent one of them Garrison Keillor's "Pretty Good Joke Book" as background reading, but he got lost somewhere in the "Ole and Lena" cycle.

Lena: "Ole, the car won't start because it has water in the carburetor."

Ole: "You don't even know what a carburetor is. I'll check it out. Where's the car?"

Lena: "In the lake." [See bit.ly/2UbJQEI.])

I could never figure out if those students' (restrained) laughter was real, or just a polite "no-soap-radio-now-get-the-hell-on-with-it-Plattski" (see bit.ly/2Ek1DTr). I'm determined to do better this year, but I needed some help. For this April Fool's Day edition of Don't Get Me Started, I decided to see how our computer assistants were progressing with their attempts at providing humor.

Well, I tried. I asked Alexa, Cortana, Siri and OK Google, "Tell me a joke." All of them did a terrible job—beyond terrible. For example: "Why was the chicken afraid of the chicken?" Answer: "It was chicken." I won't tell you which service produced that groaner; the others were just as bad.

I then tried asking, "Tell me a *funny* joke." The results still stunk. Alexa offered a small ray of hope when she asked if I wanted Jimmy Fallon to tell me one. I accepted with high hopes, but had them quickly dashed. Question: "What did the horse say to scarecrow?" Answer: "Hay!" Hey (sorry), Microsoft, Apple, Amazon and Google: You have to do better than this to call yourself an AI. Do you need me as a consultant?

Science fiction shows us computers dealing with humor, most notably in Heinlein's superb novel, "The Moon Is a Harsh Mistress" (Putnam, 1966). Set in 2076, Earth's moon is a penal colony. The supercomputer owned by the Lunar Authority somehow wakes up and becomes self-aware. He befriends his repairman, Manuel Garcia O'Kelly Davis, who names him Mycroft Holmes, or Mike for short. Mike develops a sense of humor, albeit a sophomoric one. "His idea of a thigh-slapper would be to dump you out of bed—or put itch powder in [your] pressure suit."

> Perhaps humor is one of those topics that remains eternally incomputable, as I postulated in my February 2019 column.

Mike extracts a list of jokes from human literature, and Manny promises to explain to Mike which ones are funny and why. The battle for lunar independence intervenes and takes center stage, but I've always enjoyed the hyper-intelligent Mike's struggles to grok this element of humanity.

The earliest literary reference I can find to computer analysis of humor is Asimov's short story, "Jokester" (Infinity Science Fiction, 1956). A brilliant scientist named Meyeroff is caught feeding jokes into the supercomputer Multivac. The computer eventually learns enough to tell Meyeroff that humor is a study tool imposed on humans by extraterrestrials. This revelation destroys the value of the experiment to the aliens, who turn it off as fast as a modern company's HR department stomps out unauthorized levity. Nothing is ever funny again for all of humanity. So let's be careful out there.

Perhaps humor is one of those topics that remains eternally incomputable, as I postulated in my February 2019 column (msdn.microsoft.com/magazine/mt833276). I'd be really surprised to see the publication of "Truly Tasteless Jokes, Volumes 1-27," by Mike Holmes, anytime soon. ■

I'm announcing a new program: Pick Plattski's Brain. Inspired by a similar service at Balsamiq (bit.ly/2XuqMUk), I'm setting aside two hours per week during which anyone can reserve a Skype call and ask me anything. Working from home gets lonely sometimes, so meeting readers for an hour is a great way to make new friends, and give back to the community. Contact me at Skype name dplattipswich, and we'll set up a time. I won't sign an NDA, but I won't share anything we discuss outside my company (meaning me). First come, first serve. See you soon.

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

Address the ELEPHANT IN THE ROOM

Bad address data costs you money, customers and insight.

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

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

See the Elephant in Your Business -

Name it and Tame it!

melissa

www.Melissa.com | 1-800-MELISSA

Free Trials, Free Data Quality Audit & Professional Services.