THE MICROSOFT JOURNAL FOR DEVELOPERS

MAY 2019 VOL 34 NO 5

msdn magazine

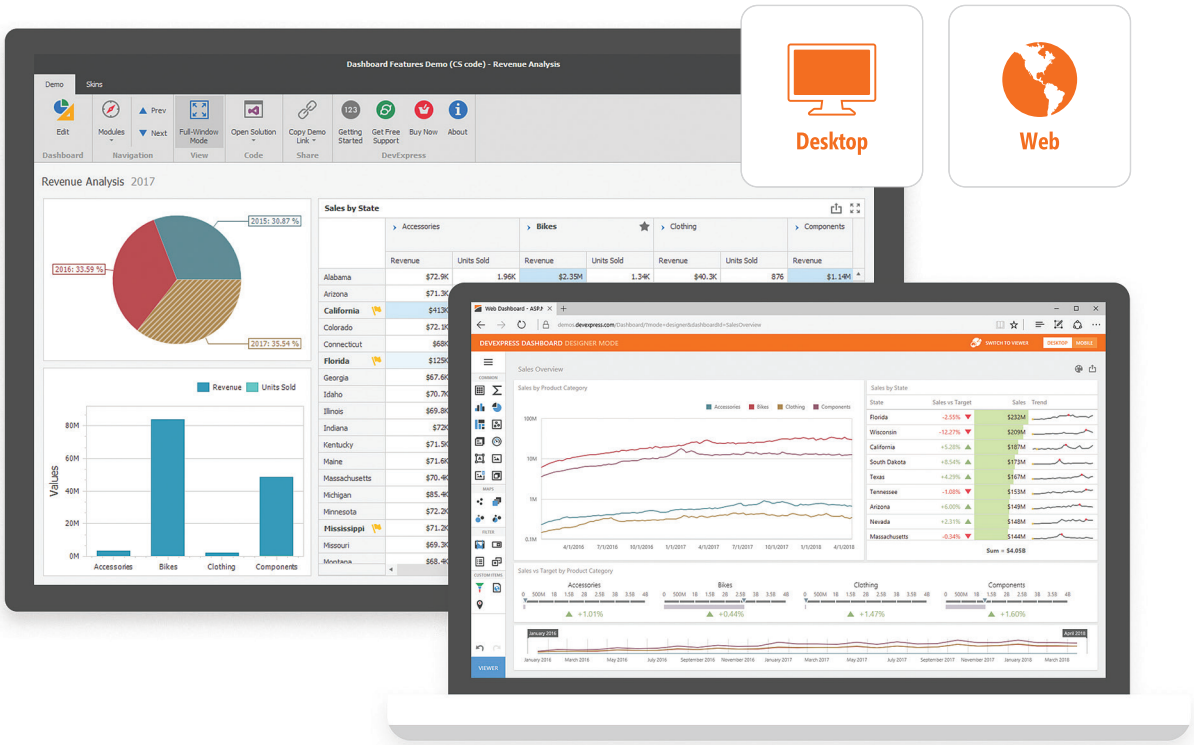
C#

New Features in C# 8.0.....16

DevExpress®

DevExpress Dashboard for .NET

Create and distribute royalty-free decision support systems and effortlessly share business intelligence across your entire enterprise.



Data Grid · Pivot Grid · Card View · Charts · Gauges · Treemap · Map · Range Control · Filter Control · Rich Text Editor · Image

With DevExpress Dashboard for .NET, creating insightful and information-rich decision support systems for executives and business users across platforms and devices is a simple matter of selecting the appropriate UI element (Chart, Pivot Table, Data Card, Gauge, TreeMap, Map, Grid or simple Filter elements) and dropping data fields onto corresponding arguments, values, and series. And because DevExpress Dashboard automatically provides the best data visualization option for you, results are immediate, accurate and always relevant.

Learn More Today — Try DevExpress Dashboard Risk Free for 30 days

devexpress.com/dashboard

THE MICROSOFT JOURNAL FOR DEVELOPERS

MAY 2019 VOL 34 NO 5

msdn magazine

C#

New Features in C# 8.0.....16

Pattern Matching in C# 8.0
Filip Ekberg.....16

Custom XAML Controls
Jerry Nixon.....22

Create a Centralized Pull Request Hub with WinForms in .NET Core 3.0
Eric Fleming.....30

Using Survival Analysis for Predictive Maintenance
Zvi Topol.....38

COLUMNS

DATA POINTS
EF Core in a Docker Containerized App, Part 2
Julie Lerman, page 6

THE WORKING PROGRAMMER
Coding Naked: Naked Collections
Ted Neward, page 13

CUTTING EDGE
Routing and Route Templates in Blazor
Dino Esposito, page 46

TEST RUN
Weighted k-NN Classification Using C#
James McCaffrey, page 50

DON'T GET ME STARTED
Calc or Stats?
David S. Platt, page 56

Microsoft

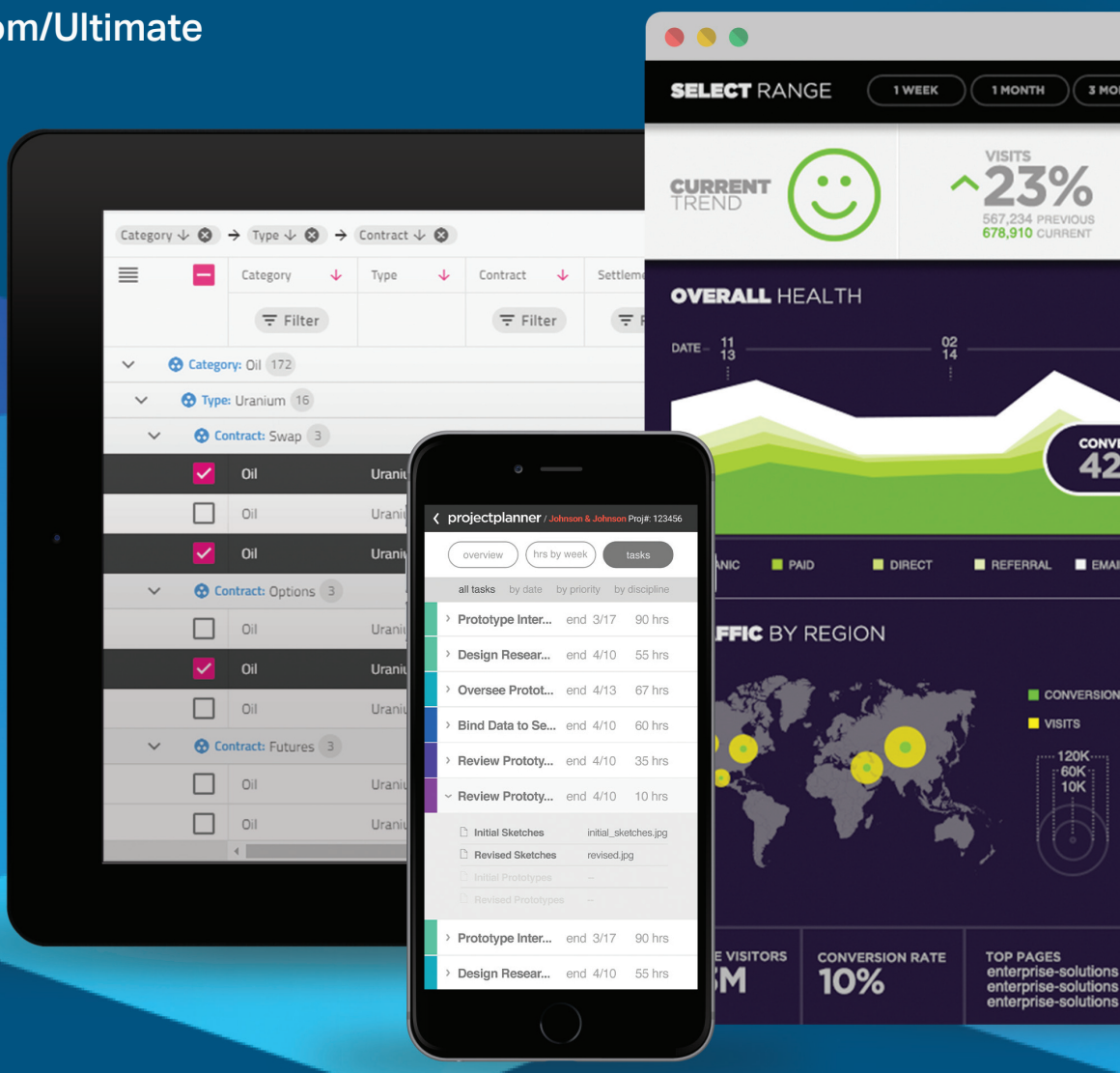
INFRAGISTICS

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

New Release

Infragistics Ultimate 19.1

- Fastest **grids** & **charts** on the market – any device, any platform
- Build Spreadsheets with Charts in WPF, Windows Forms, Angular & JavaScript
- Get Angular code from Sketch designs with Indigo.Design
- 100% support for .NET Core 3

msdn magazine

MAY 2019 VOLUME 34 NUMBER 5

ID STATEMENT MSDN Magazine (ISSN 1528-4859) is published 13 times a year, monthly with a special issue in November by 1105 Media, Inc., 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

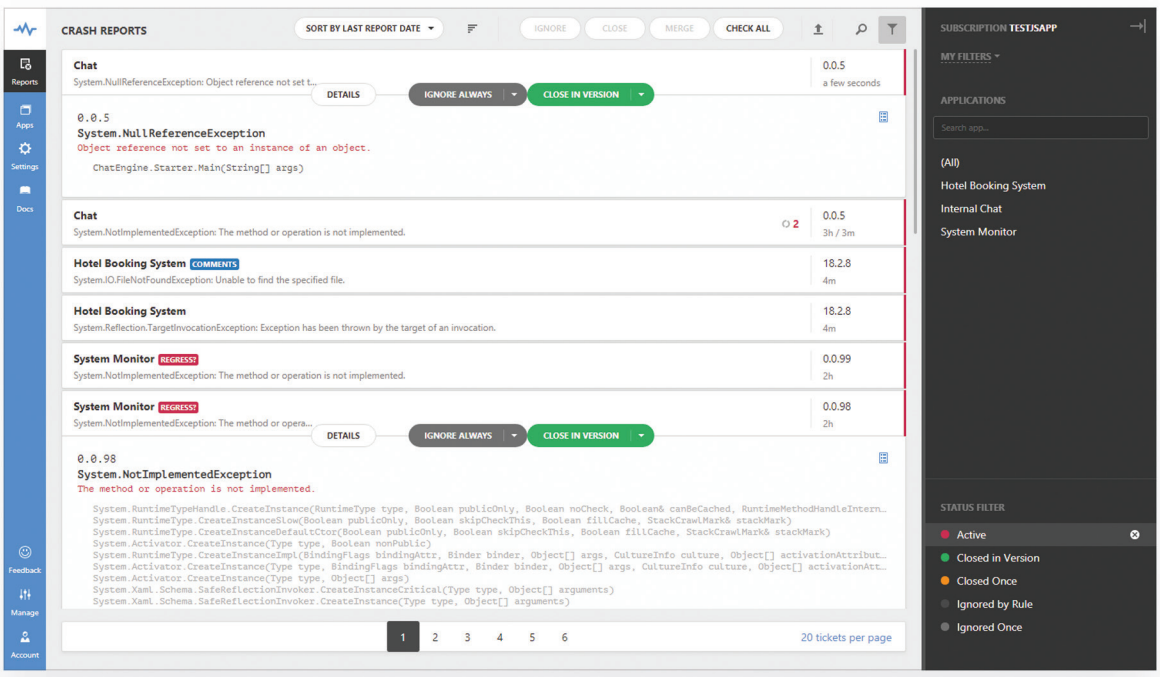
The opinions expressed within the articles and other contentsherein do not necessarily express those of the publisher.

Microsoft

logify™

Detect Errors and Exceptions
with 24/7 Application Monitoring

Logify's cloud-based application monitoring service allows you to automatically collect app crash events and runtime exceptions. With Logify, you will never have to deal with Visual Studio's® internal exception information. Logify presents all relevant exception data in an easy-to-understand, clutter-free manner. From loaded modules and cookies to browser info, OS build, user activity — Logify will organize results so you can address application issues in the shortest possible time.



Learn More Today — Try Logify Risk Free for 15 days

devexpress.com/logify

DevExpress®

MICHAEL DESMOND

Flight of Failure

Two fatal crashes in five months. Not since the Dehavilland Comet in 1954 (bit.ly/2D0z51d) has a new commercial jet airliner suffered this kind of calamity so early in its life. A pair of Boeing 737 MAX aircraft, introduced just last year, crashed minutes after takeoff, when Lion Air 610 (LI610) plunged into the Java Sea off Indonesia in October 2018 and Ethiopian Airlines 302 (ET302) crashed just outside of Addis Ababa in February. In all, 346 lives were lost.

I've written about airline accidents before ("Chain of Disaster," msdn.com/magazine/mt573708), exploring the role that automation and instrumentation played in the 2009 crash of Air France 447. And in the recent 737 MAX accidents, some of the same issues are cropping up. Commercial aviation has become significantly—even remarkably—safer over the past 30 years, but when accidents do happen, it's often at the intersection of automated systems and the pilots who command them.

In the case of the 737 MAX, a system introduced to ease the aircraft's nose down at certain points of flight malfunctioned at the worst possible moment—just after takeoff. The Maneuvering Characteristics Augmentation System (MCAS) was designed to make the 737 MAX behave more like its predecessor, the widely deployed 737NG, by counteracting a tendency in the new jet to at times pitch up when under power with the nose high. The goal was to make the 737 MAX and NG so similar to fly that no simulator training was needed for NG pilots, potentially saving airlines millions of dollars.

And there's the rub: MCAS had to be pretty much invisible. Adding instrumentation to the cockpit or publishing procedures specific to erroneous MCAS activation could mandate simulator training. So when an angle of attack (AoA) sensor failed on LI610 and tricked MCAS into thinking the plane's nose was way too high soon after takeoff, it caught the crew unprepared. The pilots repeatedly countermanded the nose-down trim from MCAS, only for the system to redouble the down force moments later, ultimately resulting in a high-speed dive.

Five months later ET302 encountered the same phenomenon. Investigators reported that the crew responded appropriately, disabling the trim motors in the tail to keep MCAS from forcing the nose further down. But now the pilots had to wrestle with a physical trim wheel, using cables to move the stabilizers at the back of the plane. Unfortunately, at the high speed the jet was traveling, aerodynamic forces made this impossible. When they re-engaged the trim motors in a bid to raise the aircraft's nose, MCAS almost immediately pushed the nose down and the crew lost control.

The investigation into the accidents is ongoing, but hard lessons are emerging. Among issues being discussed in its implementation:

Data validation: The AoA sensor data in each accident was clearly anomalous, yet MCAS remained activated throughout each event. By contrast, automated systems like autopilot typically fall back to manual control when fed incoherent data.

Redundancy: MCAS activated based on input from just one of the two AoA sensors on the 737 MAX fuselage. Implementing a second sensor would significantly reduce the risk of inadvertent activation.

Conditional code: MCAS activated in response solely to high AoA values. Adding logic to account for airspeed, altitude and pilot input might prevent unwanted activation—for instance, MCAS could be coded to disable if a pilot commands trim up after an MCAS input.

Mission scope: MCAS has been described as an enhancement to make the 737 MAX fly and feel to pilots like a 737NG. Yet the software repeatedly counteracted pilot commands and ultimately trimmed to maximum deflection.

Transparency: MCAS had no UI—no visual or aural indication that it had activated or encountered a failure state—and no published failure checklist for pilots. The assumption was pilots would interpret unwanted MCAS activation as a runaway trim motor and use that checklist to shut down the motor.

While the flaws in MCAS are likely resolvable, the larger question remains of how this system ended up in an FAA-approved aircraft design. I expect the lessons of the 737 MAX and MCAS will inform software and systems development for decades to come.

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

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

Intellifront BI | from $23,246.35

Christiansteven

Cost-Effective Data Analytics & Business Intelligence.

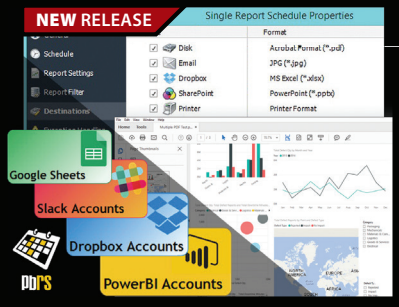
- Design and serve visually stunning real-time reports using Grids, Charts, Gauges, Pies & more
- Add automatically updating KPI cards with aggregation, rounding, units, goals, colors & glyphs
- Create & assign Reporting, KPIs and Dashboards as consolidated "Canvases" to user groups
- Give your users 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

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



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

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

JULIE LERMAN

EF Core in a Docker Containerized App, Part 2

In last month's column (msdn.com/magazine/mt833405), I created an ASP.NET Core API project to run inside a Docker container. The project used EF Core to persist data. In that column, I began by using SQL Server LocalDB, but when it came time to run the app inside the Docker container, I hit a snag because the current setup assumed the SQL Server LocalDB existed inside the container. As a quick solution, I switched to using SQLite, which would be installed when the new project got built within the container. With this approach, you could see the full solution succeeding during debug.

Switching to SQLite was simply a means for instant gratification. Let's continue the journey by learning about proper production-worthy solutions for targeting SQL Server when publishing an API inside a Docker image. This column will focus on targeting a consistently available Azure SQL Database that can be used from anywhere.

Pointing the API to Azure SQL Database

My starting point is the solution as I left it at the end of my previous column. If you haven't read that yet, you might want to start there for context. Throughout this article, I'll evolve that solution bit by bit.

While EF Core can create a database for you on Azure SQL Database, the SQL Server must pre-exist. I already have a few SQL Servers set up on Azure, so I'll use one of these for this column's demos.

First, I'll add a new connection string named MagsConnectionAzSql—pointing to that server—into appsettings.json, which already contains connection strings I used in Part 1. I've wrapped the long lines for readability although JSON doesn't honor returns. I'm also including fake credentials:

```
"ConnectionStrings": {
  "MagsConnectionMssql":"Server=(localdb)\\mssqllocaldb;
    Database=DPO419Mags;Trusted_Connection=True;",
  "MagsConnectionSqlite": "Filename=DPO419Mags.db;",
  "MagsConnectionAzSql": "Server=tcp:msdnmaglerman.database.windows.net,1433;
    Initial Catalog=DPO419Mags;Persist Security Info=False;
    User ID=lerman;Password=eiluj;
    MultipleActiveResultSets=False;Encrypt=True;
    TrustServerCertificate=False;Connection Timeout=30;"
```

Next, I'll change the DbContext configuration in the startup file's ConfigureServices method to use the SQL Server provider and this connection string:

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

This will allow EF Core migrations to find the connection string at design time so I can run any needed commands. At run time, the app will be able to find the connection string, as well. In fact, because my last work was with SQLite, I need to reset the migrations, which in my demo means deleting the Migrations folder and running *add-migration initSqlAz* to get the correctly described migration for the database.

> While EF Core can create a database for you on Azure SQL Database, the SQL Server must pre-exist.

Once the migration exists, I'll run the app using the Docker profile. This will prove to me that the Migrate method in program.cs is able to create the database in the cloud and the controller is able to query it—all from within the Docker container—as expected. The first time the app runs Migrate and must create the database, expect a short delay. When done, not only does the browser relay the three magazines with which I seeded the database (in the previous column), as **Figure 1** shows, but I can see the database listed in both the Azure Portal and in the Visual Studio SQL Server Object Explorer.

Note that the default configuration of the database created by EF Core was set up with the pricing tier: Standard S0: 10 DTUs. You can make changes to that in the Portal or when using Azure CLI for a production app. In fact, for production, you probably want to create the database explicitly in order to ensure its settings are aligned to your needs. Then you can use EF Core migrations to manage the schema.

Considerations for Handling Secrets

While this worked so nicely, it's not yet production-ready. There are a few problems to consider.

The first is that the connection string and secrets are hardcoded into the appsettings.json file. It's easy enough to modify that connection string in the JSON file without having to recompile the project, so "hardcoded" is a bit of an exaggeration. But it's not dynamic as far as Docker is concerned because the appsettings file will be "baked in" to the Docker image. You'll probably want to have more control over connection strings for development, staging, testing and production. This isn't a new problem and various

Code download available at msdn.com/magazine/0519magcode.

LEADTOOLS®
Cloud Services

Introducing LEADTOOLS Cloud Services

LEADTOOLS Cloud Services is a high-powered and scalable Web API that gives developers a hassle-free interface for implementing OCR, Document Conversion, MICR, and Barcode into any application.

Get Started Today

GET 50 FREE PAGES WHEN YOU SIGN UP

SERVICES.LEADTOOLS.COM

solutions have existed for quite some time. With respect to securing the secrets, the new secrets management tool in ASP.NET Core can help during development. See the documentation at bit.ly/2HeVgVr for details.In addition, the appsettings.json file is text and anyone can read it from your source control if you accidentally push it to a public repository along with an app.

A better path for the containerized solution is to use Docker environment variables, which the app can read at run time while continuing to be able to run migration commands at design time. This also gives you the flexibility to provide values to those variables dynamically.

> For running and debugging locally in Docker, I can switch to a SQL Server on my network or point to the Azure SQL Database.

Here's the plan: I'll use SQL Server LocalDB at design time, as well as for testing out the app "on metal"; that is, when debugging against the Kestrel or IIS servers. And because LocalDB doesn't need credentials, I don't have to worry about secrets in its connection string. For running and debugging locally in Docker, I can switch to a SQL Server on my network or point to the Azure SQL Database. Finally, for the production version of the containerized app, I'll be sure it's pointing to the Azure SQL Database. Throughout, you'll see how you can use Windows and Docker environment variables to keep your secrets secret.

And here's something that's a great convenience: Because all of these approaches use some flavor of SQL Server, I can always use the same provider, Microsoft.EntityFrameworkCore.SqlServer.

Moving the Development Connection String to Development Settings

ASP.NET Core will default to the appsettings.Development.json settings file when running from Visual Studio, but in production it defaults to appsettings.json.

I'll remove the entire connectionStrings section from appsettings.json and, instead, add the LocalDB connection string into appsettings.Development.json. You'll find this file if you expand the arrow glyph next to appsettings.json in Solution Explorer:

```
"ConnectionStrings": {
    "MagsConnectionMssql":
      "Server=(localdb)\\mssqllocaldb;Database=
        DPO419Mags;Trusted_Connection=True;"
  }
```

Because I want the app to be able to read both this connection string at design time and the environment variable provided by Dockerfile at run time, I need to employ a different syntax for the UseSqlServer options. Currently (and most commonly), you use Configuration.GetConnectionString to read the string from appsettings files. That won't work, however, for environment variables, whether they're from Windows or Docker. GetConnectionString is a helper method that supplants referencing the property name directly.

But I can read both the appsettings values and any environment values as key-value pairs using this syntax:

```
services.AddDbContext<MagContext>(options =>
  options.UseSqlServer(
    Configuration["ConnectionStrings:MagsConnectionMssql"]));
```

Let's verify that EF Core migrations can find the connection string in appsettings.Development.json, which you can do by running the PowerShell migration command Get-DbContext. This forces EF Core to do the same work as with any of the other migration commands, and results in output that shows the provider name, database name and data source:

```
providerName                           databaseName dataSource            options
------------                           ------------ ----------            -------
Microsoft.EntityFrameworkCore.SqlServer DPO419Mags  (localdb)\mssqllocaldb  None
```

Creating a Connection String for Docker's Eyes Only

So now I know appsettings works at design time. What about letting Docker find its alternate connection string when the container is running, without having to modify startup.cs as you go back and forth?

It's helpful to know that the CreateWebHostBuilder method called in program.cs calls AddEnvironmentVariables, which will read available environment variables and store them as key-value pairs in Configuration data.

Docker has a command called ENV that lets you set a key-value pair. I'll begin by hardcoding this into the Dockerfile. I can set a new key with the same name as the one in the JSON file, which is what the UseSqlServer configuration is expecting. I can even include the colon in the key's name. I put my ENV variables in the file before the build image is created. **Figure 2** shows the Dockerfile, including the new ENV variable. I described the contents of this file in Part 1 of this series, in case you need to go back for a refresher. Note that I've abbreviated the connection string here for readability.

Let's see what impact this has. Be sure the debug profile is pointed to Docker and let's actually debug this time. I'll set a breakpoint just after the AddDbContext command in Startup.cs and debug, and then inspect the value of *Configuration["ConnectionStrings:MagsConnectionMssql"]*. I can see that it now returns the Azure SQL Database connection string, not the LocalDb connection string. Debugging into all of the available configuration data, I can see that the appsettings connection string is also loaded into the Configuration object. But as explained by Randy Patterson in his blog post at

Figure 1 **The API's Output, Listing Three Magazines Defined Using Seeding in the DbContext**

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

Figure 2 **The Dockerfile for the DataAPIDocker Project with the Connection String in Place**

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

ENV ConnectionStrings:MagsConnectionMssql=
  "Server=tcp:msdnmaglerman.database.windows.net ..."

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

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

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

bit.ly/2F5jfE8, the last setting overrides earlier settings. And environment variables are read after appsettings. Therefore, even though there are two *ConnectionStrings:MagsConnectionMssql* values, the one specified in the Dockerfile is the one being used. If you were running this in Kestrel or IIS, the Dockerfile wouldn't be executed and its environment variables wouldn't exist in Configuration.

Creating Placeholders for the Secrets

But the ENV variable isn't variable. Because it's hardcoded, it's only static right now. Also, remember, it still contains my secrets (login and password). Rather than having them in the connection string, I'll begin by extracting these two secrets into their own ENV variables. For the placeholder names, I'll use ENVID and ENVPW. Then I'll create two more ENV variables in the Dockerfile for the user ID and password and, as a first pass, I'll specify their values directly:

```
ENV ConnectionStrings:MagsConnectionAzSql=
  "Server=tcp:msdnmaglerman.database.windows.net,1433;
  Initial Catalog=DP0419Mags;
  User ID=ENVID;Password=ENVPW; [etc...]
ENV DB_UserId="lerman"
ENV DB_PW="eiluj"
```

Back in Startup.cs ConfigureServices, I'll read all three environment variables and build up the connection string with its credentials:

```
var config = new StringBuilder
   (Configuration["ConnectionStrings:MagsConnectionMssql"]);
string conn = config.Replace("ENVID", Configuration["DB_UserId"])
                    .Replace("ENVPW", Configuration["DB_PW"])
                    .ToString();
services.AddDbContext<MagContext>(options => options.UseSqlServer(conn));
```

This works easily because all the needed values are in the Dockerfile. But still the variables are static and the secrets are exposed in Dockerfile.

Moving the Secrets out of Dockerfile and into Docker-Compose

Because a Docker container can only access environment variables inside the container, my current setup doesn't provide a way to define the value of the password or other ENV variables specified in the Dockerfile. But there is a way, one that lets you step up your Docker expertise a little bit further. Docker has a feature called Docker Compose, which enables you to work with multiple images. This is controlled by a docker-compose instruction file that can trigger and run one or more Dockerfiles, with each Dockerfile controlling its own Docker image. Currently, I have only a single image and I'm going to stick with that. But I can still take advantage of Docker Compose to pass values into the image's environment variables.

Why is this so important? It allows me to keep my secrets out of the Dockerfile and out of the image, as well. I can pass in the secrets when the container instance is starting up, along with any dynamic configuration information, such as varying connection strings. There's an excellent article on this topic at bit.ly/2Uuhu8F, which I found very helpful.

Using a docker-compose file to coordinate multiple containers is referred to as container orchestration. The Visual Studio tooling for Docker can help with this. Right-click on the project in Solution Explorer, then select Add and choose Container Orchestrator Support. You'll be presented with a dropdown from which you should select Docker Compose and, when prompted, choose Linux as the Target OS. Because the Dockerfile already exists, you'll be asked if you'd like to rename it and create a new Dockerfile. Answer **No** to that question to keep your existing Dockerfile intact. Next, you'll be asked if you want to overwrite the hidden .dockerignore file. Because you haven't touched that file, either option is OK.

Using a docker-compose file to coordinate multiple containers is referred to as container orchestration.

When this operation completes, you'll see a new solution folder called docker-compose with two files in it: .dockerignore and docker-compose.yml. The yml extension refers to the YAML language (yaml.org), a very sleek text format that relies on indentation to express the file schema.

The tooling created the following in docker-compose.yml:

```
version: '3.4'

services:
  dataapidocker:
    image: ${DOCKER_REGISTRY-} dataapidocker
    build:
      context: .
      dockerfile: DataAPIDocker/Dockerfile
```

It has only one service: for the dataapidocker project. It notes that the image name is dataapidocker and the location of the docker file for when it's time to build that image.

There are a lot of ways to use docker-compose to pass environment variables into a container (dockr.ly/2TwfZub). I'll start by putting the variable directly in the docker-compose.yml file. First, I'll add an environment section inside the dataapidocker service—at the same level as image and build. Then, within the new section, I'll

define the DP_PW variable using the specific format shown here:

```
version: '3.4'

services:
  dataapidocker :
    image: ${DOCKER_REGISTRY-}dataapidocker
    build:
      context: .
      dockerfile: DataAPIDocker/Dockerfile
    environment:
      - DB_PW=eiluj
```

Don't forget to remove the DB_PW variable completely from the Dockerfile. Docker-compose will make sure the variable gets passed into the running container, but it won't exist in the image itself.

Now, to run the project, you'll need to be sure that the docker-compose solution folder is set as the startup project. Notice that the debug button is set to Docker Compose. To see the magic unfold, put a breakpoint in startup where the code is building up the connection string and then debug the app. You should see that *Configuration["DB_PW"]* is indeed able to find the value passed in from docker-compose.

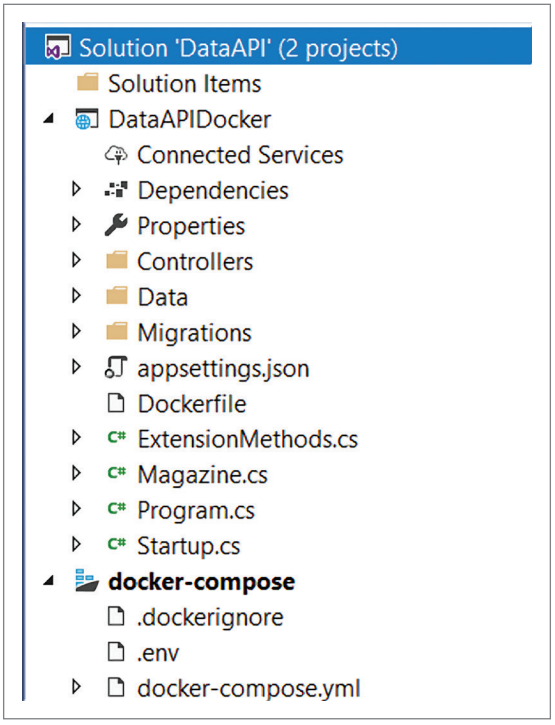


Figure 3 **The Final Solution Including the New .env File**

And, Finally, Moving the Secret Value out of Docker-Compose

But I still have my secrets in the docker-compose file and you know and I know that at some point I'm going to push that to my public source control by mistake. Docker-compose runs on my machine, not inside the Docker image. That means docker-compose can access information on the host. I could create an environment variable on my dev machine to store the password and let docker-compose discover it. You can even create temporary environment variables in the Visual Studio Package Manager Console. But Docker offers an even better option with its support for reading .env files.

> Now, to run the project, you'll need to be sure that the docker-compose solution folder is set as the startup project.

By default, docker-compose will read a file called ".env." There's no base to the file name, just ".env." It's also possible to name .env files and in docker-compose, use the env_file mapping to specify it in the service description. See my blog post at bit.ly/2CR40x3 for more information on named .env files.

You can use these to store variables, such as a connection strings, for example, dev.env, test.env or production.env; or for secrets. When I used the tooling to add the container orchestration, the docker.ignore file that the tooling created lists .env files, so those won't get accidentally pushed to your source control.

Visual Studio won't let you add a file to the docker-compose project, so I got around that by adding the new text file to the Solution Items folder, then moving it into the docker-compose project.

I'll put my secret into the .env file and the file contents are simply:

```
DB_PW=eiluj
```

Figure 3 shows what my final solution looks like.

In this way, I can just set the password while I'm developing and debugging the app and not worry about having it in any files I might share. Plus, I have options to provide other variable configuration information.

My secret is still plain text, however, which is fine on my machine. You'll likely want to encrypt these in production, though. Elton Stoneman provides guidance for this in his book, "Docker on Windows, Second Edition" (Packt Publishing, February 2019).

Next Steps

One obvious next step for me would be to deploy the container and work out how to get the environment variable with the password for my Azure SQL database into a container instance. This challenge took a lot of reading and experimenting and as I've run out of room for this installment, I've blogged about doing this fully in Azure at bit.ly/2FHdbAM. I've also written about publishing to Docker and hosting in an Azure Virtual Machine for the Docker blog. I'll update the online version of this article with the URL for that when its available.

The plan for the next installment of this multi-part column is to transition from targeting the Azure SQL Database to a SQL Server database in its own container. This will combine what's been learned thus far about docker-compose with lessons from an earlier column ("On-the-Fly SQL Servers with Docker" at msdn.com/magazine/mt784660). The two referenced blog posts will cover publishing the images and running the containers in the cloud. ■

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 technical experts for reviewing this article: *Steven Green and Mike Morton (Microsoft), Elton Stoneman (Docker)*

Coding Naked: Naked Collections

Welcome back, NOFers. Last time, I augmented the Speaker domain type with a number of properties, along with a number of annotations and conventions about those properties that provide hints (or, to be more honest, directions) to the UI as to how to validate or present those properties to the user. One thing I didn't discuss, however, is how a given domain object can have references to more than one of something. For example, Speakers often have multiple Talks they can give, or can profess expertise in one or more Topics. NOF refers to these as "collections," and there are a few rules surrounding how they work that are a little different from the previous conversation.

Let's see about giving Speakers some Talks and Topics, shall we?

Naked Concepts

To start with, abandon all arrays, all ye who enter here. NOF doesn't make use of arrays for collection properties, and instead relies entirely on collection objects (IEnumerable<T>-derived) to hold zero-to-many of some other type. The NOF manual strongly recommends that these collections be strongly typed (using generics), and NOF doesn't allow for multiple associations of value types (like strings, enumerated types and so on) because NOF believes that if the type is something "important" enough to warrant association, it should be a full-fledged domain type.

Thus, for example, if I want to capture the notion of a Topic (like "C#," "Java" or "Distributed Systems") in the conference system, where other programming approaches may allow you to get away with a simple "list-of-strings" as a property type, NOF insists that Topic be a full domain object type (which is to say, a public class with properties), complete with its own domain rules. It's reasonable that the list of Topics might be a fixed set, however, so I'll seed the database with the complete set of Topics that my conference wants to consider.

Likewise, although a Talk could be just a title, it's really a series of things: a title, a description, a Topic (to which it belongs or refers), and is given by one (or more) Speakers. Clearly, I have a little domain modeling in front of me yet.

Naked Collections

In many ways, the easiest way to begin is with the domain classes for Talk and Topic themselves, absent any connections between them (or Speakers). By now, much of what I write here for each of these should be pretty trivial and straightforward, as **Figure 1** shows.

So far, this is pretty straightforward. (There're obviously other things that could and/or should be added to each of these two classes, but this gets the point across pretty well.) The one new attribute used, [Bounded], is an indication to NOF that the complete (and immutable) list of instances can and should be held in memory on the client, and presented to the user as a dropdown list from which to choose. Correspondingly, then, the full list of Topics needs to be established in the database, which is easiest done in the DbInitializer class from the "SeedData" project, in the Seed method (as discussed in prior columns in this series), shown in **Figure 2**.

This will provide a (rather small, but useful) list of Topics from which to work. By the way, if you're playing the home game and writing the code by hand, remember to add the TalkRepository to the main menu by adding it to the MainMenus method in NakedObjectsRunSettings.cs in the Server project. In addition, make

Figure 1 **Domain Classes for Talk and Topic**

```
public class Talk
  {
    [NakedObjectsIgnore]
    public virtual int Id { get; set; }

    [Title]
    [StringLength(100, MinimumLength = 1,
       ErrorMessage = "Talks must have an abstract")]
    public virtual string Title { get; set; }

    [StringLength(400, MinimumLength = 1,
       ErrorMessage = "Talks must have an abstract")]
    public virtual string Abstract { get; set; }
  }
  public class TalkRepository
  {
    public IDomainObjectContainer Container { set; protected get; }

    public IQueryable<Talk> AllTopics()
    {
      return Container.Instances<Talk>();
    }
  }

  [Bounded]
  public class Topic
  {
    [NakedObjectsIgnore]
    public virtual int Id { get; set; }

    [Title]
    [StringLength(100, MinimumLength = 1,
       ErrorMessage = "Topics must have a name")]
    public virtual string Name { get; set; }

    [StringLength(400, MinimumLength = 1,
       ErrorMessage = "Topics must have a description")]
    public virtual string Description { get; set; }
  }
  public class TopicRepository
  {
    public IDomainObjectContainer Container { set; protected get; }

    public IQueryable<Topic> AllTopics()
    {
      return Container.Instances<Topic>();
    }
  }
```

sure that the two Repository types are also listed in the Services method in the same file.

Fundamentally, a Talk is given by a Speaker and is on a given Topic. I'm going to ignore the more complicated scenario when a Talk is given by two Speakers, or if a Talk will cross multiple Topics, to keep it simple for the time being. Thus, for the first step, let's add Talks to the Speaker:

```
private ICollection<Talk> _talks = new List<Talk>();
public virtual ICollection<Talk> Talks
{
  get { return _talks; }
  set { _talks = value; }
}
```

If you build and run the project, you'll see "Talks" show up as a collection (table) in the UI, but it will be empty. I could, of course, add some Talks in SeedData, but in general, Speakers need to be able to add Talks to their profiles.

Naked Actions

This can be done by adding an action to the Speaker class: A method that will appear, "magically," as a selectable item in the "Actions" menu when a Speaker object is in the display. Like properties, actions are discovered via the magic of Reflection, so all that needs to happen is to create a public method on the Speaker class:

```
public class Speaker
{
  // ...
  public void SayHello()
  {
  }
}
```

Now, when built and run, after bringing up a Speaker, the "Actions" menu is displayed and inside of it, "SayHello" appears. Currently it does nothing; it would be nice, as a starting point, to at least put a message back to the user. In the NOF world, this is done by making use of a service, an object whose purpose is to provide some additional functionality that doesn't belong to a particular domain object. In the case of general "messages back to the user," this is provided by a generic service, defined by NOF itself in the IDomainObjectContainer interface. I need an instance of one of these in order to do anything, however, and NOF uses dependency injection to provide one on demand: Declare a property on the Speaker class of type IDomainObjectContainer, and NOF will make sure each instance has one:

```
public class Speaker
{
  public TalkRepository TalkRepository { set; protected get; }
  public IDomainObjectContainer Container { set; protected get; }
```

Figure 2 **Creating a List of Topics**

```
protected override void Seed(ConferenceDbContext context)
{
  this.Context = context;

  Context.Topics.Add(new Topic() { Name = "C#",
    Description = "A classical O-O language on the CLR" });
  Context.Topics.Add(new Topic() { Name = "VB",
    Description = "A classical O-O language on the CLR" });
  Context.Topics.Add(new Topic() { Name = "F#",
    Description = "An O-O/functional hybrid language on the CLR" });
  Context.Topics.Add(new Topic() { Name = "ECMAScript",
    Description = "A dynamic language for browsers and servers" });
  Context.SaveChanges();
  // ...
}
```

The Container object has an "InformUser" method used to convey general messages back to the user, so using it from the SayHello action is as simple as:

```
public class Speaker
{
  // ...
  public void SayHello()
  {
    Container.InformUser("Hello!");
  }
}
```

> NOF needs to be able to inject "hooks" into each domain object in order to work its magic.

But I started with a desire to allow the user to add a Talk to a given Speaker's repertoire; specifically, I need to capture the title of the talk, the abstract (or description, because "abstract" is a reserved word in C#), and the Topic to which this Talk belongs. Calling this method "EnterNewTalk," then, I have the following implementation:

```
public void EnterNewTalk(string title, string description, Topic topic)
{
  var talk = Container.NewTransientInstance<Talk>();
  talk.Title = title;
  talk.Abstract = description;
  talk.Speaker = this;
  Container.Persist<Talk>(ref talk);
  _talks.Add(talk);
}
```

Several things are happening here, so let's unpack. First, I use the IDomainObjectContainer to create a transient (non-persisted) instance of the Talk. This is necessary because NOF needs to be able to inject "hooks" into each domain object in order to work its magic. (This is why all properties must be virtual, so that NOF can manage the UI-to-object synchronization, for example.) Then, the Talk's properties are set, and the Container is used again to Persist the talk; if this isn't done, the Talk isn't a persistent object and won't be stored when I add the Talk to the Speaker's list of Talks.

It's fair to ask, however, how the user specified this information to the EnterNewTalk method itself. Once again, the wonders of Reflection are at work: NOF dug the parameter names and types out of the method parameters, and constructed a dialog to capture those items, including the Topic itself. Remember when Topic was annotated with "Bounded"? That instructed NOF to build the list of Topics in this dialog to be a dropdown list, making it incredibly easy to select a topic from the list. (It should be easy to infer at this point that as I add Topics to the system, they'll all be added to this dropdown list without any additional work required.)

Now, it's reasonable to suggest that creating Talks should be something supported by the TalkRepository, not on the Speaker class itself, and, as you can see in **Figure 3**, it's an easy refactor to do.

What's more, by doing this, the "Talks" menu will be automatically adorned with a new menu item, "CreateTalk," which will—again, through the magic of Reflection—automatically create a dialog to enter the data necessary to create a Talk. Speakers, of course, can't just be typed in, so NOF will make that a "droppable" field, meaning that NOF will expect a Speaker object to be dragged-and-dropped

Figure 3 **Supporting Talk Creation with TalkRepository**

```
public class Speaker
  {
    public TalkRepository TalkRepository { set; protected get; }
    public IDomainObjectContainer Container { set; protected get; }

    public void EnterNewTalk(string title, string description, Topic topic)
    {
      var talk = TalkRepository.CreateTalk(this, title, description, topic);
      _talks.Add(talk);
    }
  }

  public class TalkRepository
  {
    public IDomainObjectContainer Container { set; protected get; }

    public Talk CreateTalk(Speaker speaker, string title, string description,
                Topic topic)
    {
      var talk = Container.NewTransientInstance<Talk>();
      talk.Title = title;
      talk.Abstract = description;
      talk.Speaker = speaker;
      Container.Persist<Talk>(ref talk);
      return talk;
    }
  }
```

into this field. (To see this in action in the default NOF Gemini interface, fire up the app, select a Speaker, then click the "Swap Pane" button—the double-arrowed button at the bottom of the display. The Speaker selected will shift to the right of the screen, and the Home interface will appear, allowing selection of the "Talks/Create Talk" item. Drag the Speaker's name to the Speaker field in the Create Talk dialog, and voila—the Speaker is selected.)

It's absolutely crucial to understand what's happening here: I now have two different UI paths (creating a Talk from the top-level menu, or creating a Talk from a given Speaker) that allow for two different user-navigation scenarios, with little effort and zero duplication. The TalkRepository worries about all things "CRUD" and "Talk" related, and the Speaker uses that code, all while keeping the user interaction entirely within the Speaker if that's how the user wants to arrange it.

Wrapping Up

This is not your grandfather's UI toolkit. In only a few lines of code, I have a workable interface (and, considering that data is being stored and retrieved from a standard SQL back end, a data storage system) that could, at least for the moment, be used directly by users. More importantly, none of this is in a proprietary format or language—this is straight C#, straight SQL Server, and the UI itself is Angular. There are a few more things to discuss about NOF's default interface, which I'll get to in the next piece, including authentication and authorization facilities. In the meantime, however … happy coding! ■

Ted Neward *is a Seattle-based polytechnology consultant, speaker and mentor. He has written a ton of articles, authored and co-authored a dozen books, and speaks all over the world. Reach him at ted@tedneward.com or read his blog at blogs.tedneward.com.*

Thanks to the following technical expert for reviewing this article: *Richard Pawson*

Pattern Matching in C# 8.0

Filip Ekberg

Over the years we've seen a ton of features in C# that improve not only the performance of our code, but more importantly its readability. Given the fast pace of the software industry, the language certainly needs to keep up and evolve with its user base. Something that's been widely used in different programming languages, such as Haskell, Swift or Kotlin sometimes find its way into C#. One of these being pattern matching—a concept that has been around for a long time, and something for which a lot of developers in the .NET space have long waited.

As of C# 7.0, developers got a taste of the power that comes with pattern matching. We saw a pattern starting to take form, which later has become an extremely powerful and interesting addition to the language. Just as other language features have drastically changed the way we write our software, I expect that pattern matching in C# will have a similar effect.

Do we really need another language feature, though? Can't we just use traditional approaches? Of course we could. Although an addition like pattern matching will most definitely change the way a lot of us choose to write our code, the same question could be said for other language features that have been introduced over the years.

One that changed the C# language drastically was the introduction of Language-Integrated Query (LINQ). Nowadays when processing data, people choose which flavor they personally like. Some choose to use LINQ, which in some cases constructs less-verbose code, while others opt for traditional loops. I expect similar uptake for pattern matching, as the new functionality will change the way developers work as they move away from more verbose approaches. Mind you, the traditional approaches aren't going anywhere, as many developers will opt to stick with tried-and-true solutions. But the additional language features should offer a way to complement C# code projects, rather than deprecate current code.

This article discusses:

- How pattern matching is evolving in C# 8.0
- The new switch expression
- How to leverage tuples together with the pattern matching changes in C# 8.0

Technologies discussed:

C# 8.0, Visual Studio 2019 Preview

Introducing Pattern Matching

If you've ever tried languages like Kotlin or Swift, you've likely seen examples of pattern matching in action. It's very commonly used among a lot of different programming languages on the market—mostly, of course, to make the code a bit more readable. So what is pattern matching?

It's rather simple. You look at a given structure and based on the way it looks, you identify it and you then can immediately use it. If you get a bag of fruit, you look down and immediately see the difference between the apples and the pears. Even if both are green. At the same time, you can look down into your bag of fruit and identify which fruits are green, as we know all fruits have a color.

Distinguishing between the *type* of fruit and an *attribute* of the fruit is exactly what pattern matching is about. Developers have different ways of expressing themselves when identifying this.

Traditionally, I could check all of this using a simple condition. But what if I need to explicitly use the apple? I'd end up in a situation where I must first validate the type, attribute and then cast to an apple. This code quickly ends up a bit messy and, frankly, it's error prone.

For its part, C# 7.0 introduced a lightweight version of pattern matching that can be helpful, though it lacks many of the nice features present in other languages.

Here's an example where I validate the specific type of fruit to be an apple. I apply an attribute constraint, and then to use it I have to cast it, like so:

```
if(fruit.GetType() == typeof(Apple) && fruit.Color == Color.Green)
{
  var apple = fruit as Apple;
}
```

Another approach I could take is to use the is keyword, which gives me a little bit more flexibility. As opposed to the previous example, the is keyword will also match on derived apples:

```
if(fruit is Apple)
{
  MakeApplePieFrom(fruit as Apple);
}
```

In this case, if the fruit is a derived type of apple, I'll be able to make an apple pie out of it. Whereas in the earlier example, it would have to be a very specific type of apple.

Fortunately, there's a better way. As I mentioned, languages like Swift and Kotlin allow you to use pattern matching. For its part, C# 7.0 introduced a lightweight version of pattern matching that can be helpful, though it lacks many of the nice features present in other languages. You can refactor the previous expression into the C# 7.0 code that follows, which allows you to use a switch to match your different patterns. It isn't perfect, but it does improve on what was previously available. Here's the code:

```
switch(fruit)
{
  case Apple apple:
    MakeApplePieFrom(apple);
    break;

  default:
    break;
}
```

A few things here are interesting. First, notice that I don't have a single type cast anywhere in this code, and also that I can use the apple just matched on in the case context. Just as with the is keyword, this will match on derived apples, as well.

This C# 7.0 code also reads better, and is much easier to have conversations around, than similar code in C# 6.0. The code is just saying, "Based on the fact that fruit is an apple, I want to use this apple." Each case can match on a type that shares similar traits, meaning they inherit from the same class, for instance, or implement the same interface. In this case, an apple, pear and banana are all fruits.

What's missing is a way to filter out the green apples. Have you seen exception filters? That's a feature introduced in C# 6.0 that allows you to catch certain exceptions only when a certain condition is met. This feature introduced the when keyword, which is applicable in pattern matching, as well. I can match the apple using pattern matching, and only enter the case when the condition is met. **Figure 1** shows this.

As **Figure 1** illustrates, the order matters. I first look for an apple with the color green, because this characteristic is the most important to me. If there's another color, let's say brown, this would indicate that my apple has gone bad and I want to throw it out. As for all other apples, I don't want them in the pie, so I'll just eat them. The final apple pattern is a "catch all" for all apples that have neither a green nor a brown color.

You'll also see that if I get an orange, I'll just peel the skin off. I'm not limited to handling one particular type; as long as the types all inherit from fruit, we're good to go.

Everything else works like the normal switch that you've been using since C# 1.0. This example was written entirely in C# 7.0, so the question is, is there room for improvement? I would say so. The code is still a bit on the expressive side, and it could be made more readable by improving the way patterns are expressed. Also, it would help to have other ways to express constraints against what my data "looks like." Let's now jump into C# 8.0 and look at the changes that have been introduced to make our lives easier.

The Evolution of Pattern Matching in C# 8.0

The latest version of C#, currently in preview, introduces some important pattern-matching improvements. To try C# 8.0, you'll have to use Visual Studio 2019 Preview, or enable the preview language features in Visual Studio 2019. The general availability of C# 8.0 is later this year, expected at the same time that .NET Core 3.0 ships. How can

Figure 1 **Applying a Filter Using the When Keyword**

```
Fruit fruit = new Apple { Color = Color.Green };
switch (fruit)
{
  case Apple apple when apple.Color == Color.Green:
    MakeApplePieFrom(apple);
    break;

  case Apple apple when apple.Color == Color.Brown:
    ThrowAway(apple);
    break;

  case Apple apple:
    Eat(apple);
    break;

  case Orange orange:
    orange.Peel();
    break;
}
```

we find new ways to express a constraint on the properties of a type? How can we make the expression of block patterns more intuitive and readable? In C# 8.0 the language takes another step forward to introduce a way to work with patterns that should be very familiar to those who've worked in languages like Kotlin. These are all wonderful additions that make the code readable and maintainable.

First, we now have an option to use something called a switch expression, instead of the traditional switch statement that developers have been using since C# 1.0. Here's an example of a switch expression in C# 8.0:

```
var whatFruit = fruit switch {
  Apple _ => "This is an apple",
  _ => "This is not an apple"
};
```

As you can see, instead of having to write case and break for each different match, I simply use a pattern and an expression. When I match for a fruit, the underscore (_) means that I don't care about the actual fruit that I matched on. In fact, it doesn't have to be an initialized type of fruit. The underscore will match on null, as well. Think of this as simply matching on the specific type. When I found this apple, I returned a string using an expression—much like the expression-bodied members that were introduced in C# 6.0.

This is about more than just saving characters. Imagine the possibilities here. For example, I could now introduce an expression-bodied member that includes one of these switch expressions, which also leverages the power of pattern matching, like so:

```
public Fruit Fruit { get; set; }
public string WhatFruit => Fruit switch
{
  Apple _ => "This is an apple",
  _ => "This is not an apple"
};
```

This can get really interesting and powerful. The following code shows how you would perform this pattern match in the traditional manner. Have a look and decide which one you would prefer:

```
public string WhatFruit
{
  get
  {
    if(Fruit is Apple)
    {
      return "This is an apple";
    }
    return "This is not an apple";
  }
}
```

Obviously, this is a very simple scenario. Imagine when I introduce constraints, multiple types to match against, and then use the casted type within the condition context. Sold on the idea yet? I thought so!

While this is a welcome addition to the language, please resist the urge to use switch expressions for every if/else if/else condition. For an example of what *not* to do, check out the following code:

```
bool? visible = false;
var visibility = visible switch
{
  true => "Visible",
  false => "Hidden",
  null, _ => "Blink"
};
```

This code indicates that you could have four cases for a nullable Boolean, which of course you can't. Just be mindful about how you use switch expressions and don't abuse the syntax, exactly as you would with any other language feature.

I've already covered the fact that switch expressions can cut down the amount of code you write, as well as make that code more readable. This is true also when adding constraints to your types. The changes to pattern matching in C# 8.0 really stand out when you look at the combination of tuples, deconstruction and what's known as recursive patterns.

As you can see, instead of having to write case and break for each different match, I simply use a pattern and an expression.

Expressing Patterns

A recursive pattern is when the output of one pattern-match expression becomes the input of another pattern-match expression. This means deconstructing the object and looking at how the type, its properties, their types, and so forth are all expressed, and then applying the match to all of these. It sounds complicated, but really it's not.

Let's look at a different type and its structure. In **Figure 2** you'll see a rectangle that inherits from Shape. The shape is just an abstract class that introduces the property point, a way for me to get the shape onto a surface so I know where it's supposed to go.

You might wonder what the Deconstruct method in **Figure 2** is all about. It allows me to "extract" the values of an instance into new variables outside of the class. It's commonly used together with pattern matching and tuples, as you'll discover in a moment.

So, I essentially have three new ways to express a pattern in C# 8.0, all of which have a specific use case. They are:

- Positional pattern
- Property pattern
- Tuple pattern

Don't worry, if you prefer the normal switch syntax, you can use these pattern-matching improvements with that, as well! These changes and additions to the language in terms of pattern matching are commonly referred to as recursive patterns.

The positional pattern leverages the deconstruct method that you have on your class. You can express a pattern that matches

Figure 2 **Example of Deconstruct**

```
abstract class Shape
{
  public Point Point { get; set; }
}

class Rectangle : Shape
{
  public int Width { get; set; }
  public int Height { get; set; }

  public void Deconstruct(out int width, out int height, out Point point)
  {
    width = Width;
    height = Height;
    point = Point;
  }
}
```

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

Figure 3 **Positional Pattern**

```
Shape shape = new Rectangle
{
  Width = 100,
  Height = 100,

  Point = new Point { X = 0, Y = 100 }
};

var result = shape switch
{
  Rectangle (100, 100, null) => "Found 100x100 rectangle without a point",
  Rectangle (100, 100, _) => "Found 100x100 rectangle",
  _ => "Different, or null shape"
};
```

given values that you get out of the deconstruction. Given the fact that you have a way defined to deconstruct the rectangle, you can express a pattern that leverages the position of the output like what you see in **Figure 3**.

First, let's match the type of shape. In this case I only want to match it against a rectangle. The second applied pattern, when matched to a rectangle, uses the deconstruct method together with the tuple syntax to express which values I require for each particular position.

I can specify that I explicitly want the point to be null, or I can use the underscore to express that I simply don't care. Keep in mind that the order matters very much here. If we'd have the version where we don't care on top, it would always match on that pattern even if the rectangle has a point or not. This is known as the positional pattern.

This is very handy if I have a deconstruct available, although if the deconstruct outputs a lot of values, it gets rather verbose. This is where the property pattern comes into play. So far I've matched on different types, but some scenarios require you to match on other things, such as state or just looking at the different property values, or the lack thereof.

As the following code describes, I don't care which type I get, as long as it matches a type containing a point, where this point has a value of 100 in the Y property, like so:

```
shape switch
{
  { Point: { Y : 100 } } => "Y is 100",
  { Point: null } => "Point not initialized",
};
```

Observe that the code doesn't in fact handle cases where the shape is null, or when the point is initialized but has a different Y value than 100. In those situations, this code will throw an exception. This could be solved by introducing the default case using the underscore.

Figure 4 **Tuple pattern**

```
var newState = (state, operation, key.IsValid) switch
{
  (State.Opened, Operation.Close, _)      => State.Closed,
  (State.Opened, Operation.Open, _)       => throw new Exception(
    "Can't open an opened door"),
  (State.Opened, Operation.Lock, true)    => State.Locked,
  (State.Locked, Operation.Open, true)    => State.Opened,
  (State.Closed, Operation.Open, false)   => State.Locked,
  (State.Closed, Operation.Lock, true)    => State.Locked,
  (State.Closed, Operation.Close, _)      => throw new Exception(
    "Can't close a closed door"),
  _ => state
};
```

I could also say that I require the point to be uninitialized, and only handle those uninitialized scenarios. This is a lot less verbose than using the positional patterns, and it works very well for situations where you can't add a deconstruct method to the type you're matching.

Finally, I have the tuple pattern, which leverages the positional pattern and allows me to compose a tuple on which to run my match. I can illustrate this with a scenario where I operate on different states, such as opening, closing and locking a door (see **Figure 4**). A particular situation may occur depending on the current state of the door, the operation I want to perform and the key I may possess. This example of using the Tuple pattern to introduce a state machine is one commonly used by C# Design Lead Mads Torgersen. Check out Torgersen's post, "Do More with Patterns in C# 8.0," at bit.ly/202SDqo.

The code in **Figure 4** first constructs a new tuple containing the current state, the desired operation, and a Boolean checking if the user has a valid key or not. It's a very simple scenario.

> While this is a welcome addition to the language, please resist the urge to use switch expressions for every if/else if/else condition.

Based on these different values, I can match on different situations by constructing more tuples, together with a positional pattern. This is the tuple pattern. If I try to open a door that's closed, but not locked, that will result in a new state telling me that the door is now open. If the door is locked and I try to unlock it with an invalid key, the door will remain locked. If I try to open a door that's opened, I get an exception. You get the idea. This is a very flexible and interesting way to approach a situation that previously was very verbose and produced code that was a lot less readable.

Final Words

The pattern-matching improvements in C# 8.0, together with the switch expression, will definitely change the way developers write applications. C# is nearly two decades old and has evolved to reflect the way that applications are built. Pattern matching is simply the latest expression of that evolution.

For developers, it's wise to be cautious about overusing these new principles and patterns. Be mindful about the code that you write, and make sure that it's readable, understandable and maintainable. That's all that your fellow developers ask for, and I reckon that these changes to the language will help improve the signal-to-noise ratio of the code you produce. ■

Filip Ekberg *is a public speaker, Pluralsight author, principal consultant and author of "C# Smorgasbord" (2012). Ekberg has worked all the way from Sydney to Gothenburg, and has more than a decade of experience with C#. You can contact him through Twitter: @fekberg or via filip@ekberg.dev.*

Thanks to the following Microsoft technical expert for reviewing this article: *Bill Wagner*

Visual Studio LIVE! EXPERT SOLUTIONS FOR ENTERPRISE DEVELOPERS | SQL Server LIVE! TRAINING FOR DBAs AND IT PROS | TECHMENTOR IN-DEPTH TRAINING FOR IT PROS | Office & SharePoint LIVE! ON-PREMISE, CLOUD, & CROSS-PLATFORM TRAINING | Artificial Intelligence LIVE! AI FOR DEVELOPERS AND DATA SCIENTISTS | Cloud & Containers LIVE! CLOUD-NATIVE, PAAS & SERVERLESS COMPUTING

The Ultimate Education Destination

LIVE! 360

TECH EVENTS WITH PERSPECTIVE

6 Great Events, 1 Low Price!

November 17-22, 2019 | ORLANDO
Royal Pacific Resort at Universal

Live! 360 brings the IT and developer community together for a unique conference, featuring 6 co-located conferences for (almost) every IT title. Customize your learning by choosing from hundreds of sessions, dozens of track topics, workshops and hands-on labs from hundreds of expert speakers.

Who Should Attend:

- Developers
- IT Professionals
- Database Administrators
- Business Intelligence Professionals
- SharePoint Professionals
- and more!

Save $500 when you register by August 16!

Promo Code MSDN

live360events.com/orlando

EVENT PARTNER: Microsoft

SILVER SPONSOR: aws

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

PRODUCED BY: Converge 360 — An 1105 Media company

Custom XAML Controls

Jerry Nixon

As an enterprise developer, you know your way around SQL Server. You know .NET Web services. And, for you, designing beautiful XAML interfaces (probably with Windows Presentation Foundation [WPF]) is child's play. Like thousands of other career developers, Microsoft technologies pad your resumé and you clip *MSDN Magazine* articles like this one and pin them to your Kanban board. Get your scissors, this article is table stakes.

It's time to boost your expertise with XAML controls. The XAML framework offers an extensive library of controls for UI development, but to accomplish what you want, you need more. In this article I'll show you how to get the results you want using XAML custom controls.

This article discusses:

- Logic encapsulation
- Component reusability
- Control templating
- Data binding

Technologies discussed:

Windows Presentation Foundation, Universal Windows Platform, XAML, C#

Code download available at:

github.com/JerryNixon/MsdnMgazineCustomControls

Custom Controls

There are two approaches to creating custom controls in XAML: user controls and templated controls. User controls are an easy, designer-friendly approach to creating a reusable layout. Templated controls offer a flexible layout with a customizable API for developers. As is the case in any language, sophisticated layouts can produce thousands of lines of XAML that can be difficult to navigate productively. Custom controls are an effective strategy for reducing layout code.

Choosing the correct approach will impact how successfully you can use and reuse the controls in your application. Here are some considerations to help you get started.

Simplicity. Easy is not always simple, but simple is always easy. User controls are simple and easy. Developers on any level can deliver them with little reach for documentation.

Design experience. Many developers love the XAML designer. Templated control layouts can be built in the designer, but it's the user control that embraces the design-time experience.

API surface. Building an intuitive API surface lets developers easily consume it. User controls support custom properties, events and methods, but templated controls are the most flexible.

Flexible visuals. Providing a great default experience lets developers consume controls with ease. But flexible controls support re-templated visuals. Only templated controls support re-templating.

To sum, user controls are optimal for simplicity and design experience, while templated controls give you the best API surface and the most flexible visuals.

Figure 1 **Prototyped UI**

If you decide to start with a user control and migrate to a templated control, you have some work ahead of you. But, it's not terminal. This article starts with a user control and moves to a templated control. It's important to recognize that many reusable layouts only require user controls. It's also reasonable to open a line-of-business solution and find both user controls and templated controls.

A New User Story

You need to get new users to consent to the end-user license agreement (EULA) in your new application. As you and I both know, no user wants to consent to a EULA. Still, legal needs to ensure users check "I agree" before continuing. So even if the EULA is confusing, you're going to make sure the XAML interface is clean and intuitive. You start prototyping; you add a TextBlock, a CheckBox, a Button, as shown in **Figure 1**, and then you start thinking.

Prototyping in the XAML designer is fast. And it's easy because you took the time to learn the tooling. But what about other forms in your application? You might need this functionality elsewhere. Encapsulation is a design pattern used to hide complex logic from the consumer. Don't repeat yourself (DRY) is another, focusing on code reuse. XAML offers both through user controls and custom controls. As a XAML developer you know custom controls are more powerful than user controls, but they're not as simple. You decide to start with a user control. Perhaps it can get the job done. Spoiler alert: It can't.

User Controls

User controls are easy. They provide consistent, reusable interfaces and a custom, encapsulated codebehind. To create a user control, select User Control from the Add New Item dialog, as shown in **Figure 2**.

User controls are usually a child of another control. However, their lifecycle is so similar to that of windows and pages that a user control can be the value set to the Window.Current.Content property. User controls are full-featured, supporting Visual State Management, internal resources, and every other staple of the XAML framework. Building them isn't a compromise of available functionality. My goal is to reuse them on a page, taking advantage of their support for Visual State Management, resources, styles and data binding. Their XAML implementation is simple and designer-friendly:

```
<UserControl>
  <StackPanel Padding="20">
    <TextBlock>Lorem ipsum.</TextBlock>
    <CheckBox>I agree!</CheckBox>
    <Button>Submit</Button>
  </StackPanel>
</UserControl>
```

This XAML renders my earlier prototype and shows just how simple a user control can be. Of course, there's no custom behavior yet, only the built-in behavior of the controls I declare.

The Text Fast Path EULAs are long, so let's address text performance. The TextBlock (and only the TextBlock) has been optimized to use the fast path, low memory and CPU rendering. It's built to be fast—but I can spoil that:

```
<TextBlock Text="Optimized" />
<TextBlock>Not optimized</TextBlock>
```

Using TextBlock inline controls like <Run/> and <LineBreak/> breaks optimization. Properties like CharacterSpacing, LineStackingStrategy and TextTrimming can do the same. Confused? There's an easy test:

```
Application.Current.DebugSettings
  .IsTextPerformanceVisualizationEnabled = true;
```

IsTextPerformanceVisualizationEnabled is a little-known debug setting that allows you to see what text in your application is optimized as you debug. If the text isn't green, it's time to investigate.

With every release of Windows, fewer properties impact the fast path; however, there are still several that negatively and unexpectedly impact performance. With a little intentional debugging, this isn't a problem.

Immutable Rules There are as many opinions as there are options for where business logic should reside. A general rule puts less-mutable rules closer to the control. This is generally easier and faster and it optimizes maintainability.

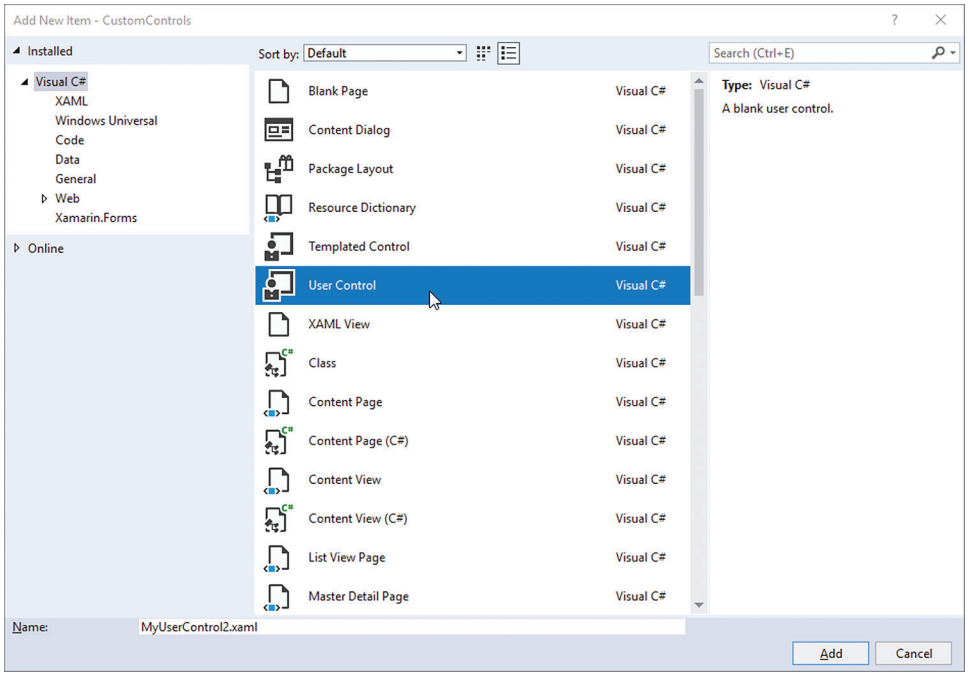


Figure 2 **Add New Item Dialog**

Business rules, by the way, are different from data validation. Responding to data entry and checking for text length and numeric ranges is simply validation. Rules govern types of user behavior.

For example, a bank has a business rule not to give a loan to customers with a credit score below a particular value. A plumber has a rule not to travel to a customer outside a certain ZIP code. Rules are about behavior. In some cases, rules change every single day, like what credit scores influence new loans. In other cases, rules never change, such as how a mechanic won't ever work on a Subaru older than 2014.

Now, consider this acceptance criteria: A user can't click the Button until the CheckBox is checked. This is a rule and it's as close to immutable as you can get. I'm going to implement it close to my controls:

```
<StackPanel Padding="20">
  <TextBlock>Lorem ipsum.</TextBlock>
  <CheckBox x:Name="AgreeCheckBox">I agree!</CheckBox>
  <Button IsEnabled="{Binding Path=IsChecked,
    ElementName=AgreeCheckBox}">Submit1</Button>
  <Button IsEnabled="{x:Bind Path=AgreeCheckBox.IsChecked.Value,
    Mode=OneWay}">Submit2</Button>
</StackPanel>
```

In this code, data binding perfectly satisfies my requirement. The Submit1 Button uses classic WPF (and UWP) data binding. The Submit2 Button uses modern UWP data binding.

Notice in **Figure 3** that Submit2 is enabled. Is this right? Well, in the Visual Studio Designer, classic data binding has the advantage of rendering at design time. For now, compiled data binding (x:Bind) only occurs at run time. Choosing between classic and compiled data binding is the most difficult easy decision you're going to make. On the one hand, compiled binding is fast. But, on the other, classic binding is simple. Compiled binding exists to address XAML's difficult performance problem: data binding. Because classic binding requires runtime reflection, it's inherently slower, struggling to scale.

Many new features have been added to classic binding, such as asynchronous binding; and several patterns have emerged to help developers. Yet, as UWP postured to succeed WPF, it suffered from the same dragging issue. Here's something to think about: The ability to use classic binding in an asynchronous mode was not ported to UWP from WPF. Read into that what you want, but it does encourage enterprise developers to invest in compiled binding. Compiled binding leverages the XAML code generator, creating the codebehind automatically and coupling the binding statements with real properties and datatypes expected at run time.

Because of this coupling, mismatched types can create errors, as can attempting to bind to anonymous objects or dynamic JSON objects. These edge cases aren't missed by many developers, but they're gone:

- Compiled binding resolves the performance issues of data binding while introducing certain constraints.
- Backward compatibility maintains classic binding support while giving UWP developers a better option.
- Innovation and improvements to data binding are invested into compiled binding, not classic binding.
- Features like function binding are available only with compiled binding where Microsoft's binding strategy is clearly focused.

Yet the simplicity and the design-time support of classic binding keeps the argument alive, pressing the Microsoft developer tooling team to continue to improve compiled binding and its developer experience. Note that in this article, choosing one or the other will have a near-immeasurable impact. Some of the samples will demonstrate classic binding while others show compiled binding. It's up to you to decide. The decision, of course, is most meaningful in large apps.

Custom Events Custom events can't be declared in XAML, so you handle them in your codebehind. For example, I can forward the click event of the submit button to a custom click event on my user control:

```
public event RoutedEventHandler Click;
public MyUserControl1()
{
  InitializeComponent();
  SubmitButton.Click += (s, e)
    => Click?.Invoke(this, e);
}
```

Here, the code raises the custom events, forwarding the RoutedEventArgs from the button. Consuming developers can handle these events declaratively, like every other event in XAML:

```
<controls:MyUserControl1 Click="MyUserControl1_Click" />
```

The value of this is that consuming developers don't need to learn a new paradigm; custom controls and out-of-the box first-party controls behave functionally the same.

Custom Properties To let consuming developers supply their own EULAs, I can set the x:FieldModifier attribute on the TextBlock. This modifies XAML compilation behavior from the default private value:

```
<TextBlock x:Name="EulaTextBlock" x:FieldModifier="public" />
```

But easy doesn't mean good. This method offers little abstraction and requires developers to understand the internal structure. It also requires codebehind. So, I'll avoid using the attribute approach in this case:

```
public string Text
{
  get => (string)GetValue(TextProperty);
  set => SetValue(TextProperty, value);
}
public static readonly DependencyProperty TextProperty =
  DependencyProperty.Register(nameof(Text), typeof(string),
    typeof(MyUserControl1), new PropertyMetadata(string.Empty));
```

Equally easy—and without the caveats—is a dependency property data-bound to the TextBlock's Text property. This allows the consuming developer to read, write or bind to the custom Text property:

```
<StackPanel Padding="20">
  <TextBlock Text="{x:Bind Text, Mode=OneWay}" />
  <CheckBox>I agree!</CheckBox>
  <Button>Submit</Button>
</StackPanel>
```

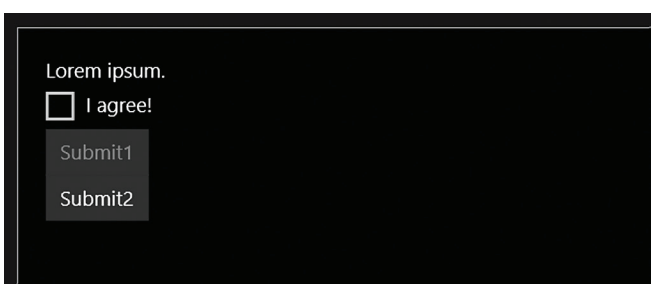


Figure 3 **Implementing a Business Rule with Data Binding**

The dependency property is necessary to support data binding. Robust controls support basic use cases like data binding. Plus, the dependency property adds only a single line to my code base:

```
<TextBox Text="{x:Bind Text, Mode=TwoWay,
  UpdateSourceTrigger=PropertyChanged}" />
```

Two-way data binding for custom properties in user controls is supported without INotifyPropertyChanged. This is because dependency properties raise internal changed events that the binding framework monitors. It's its own kind of INotifyPropertyChanged.

The preceding code reminds us that UpdateSourceTrigger determines when changes are registered. Possible values are Explicit, LostFocus and PropertyChanged. The latter occurs as changes are made.

Roadblocks The consuming developer may want to set the content property of the user control. This is an intuitive approach, but it's not supported by user controls. The property is already set to the XAML I declared:

```
<Controls:MyUserControl>
  Lorem Ipsum
</Controls:MyUserControl>
```

This syntax overwrites the content property: the TextBlock, CheckBox and Button. If I consider re-templating a basic XAML use case, my user control fails to deliver a complete, robust experience. User controls are easy, but offer little control or extensibility. An intuitive syntax and re-templating support are part of a common experience. It's time to consider a templated control.

> XAML has seen massive improvements in memory consumption, performance, accessibility and visual consistency.

Templated Controls

XAML has seen massive improvements in memory consumption, performance, accessibility and visual consistency. Developers love XAML because it's flexible. Templated controls are a case in point.

Templated controls can define something completely new, but are typically a composite of several existing controls. The example here of a TextBlock, CheckBox and Button together is a classic scenario.

By the way, don't confuse templated controls with custom controls. A templated control is a custom layout. A custom control is just a class inheriting an existing control without any custom styling. Sometimes, if all you need is an extra method or property on an existing control, custom controls are a great option; their visuals and logic are already in place and you're simply extending them.

Control Template A control's layout is defined by a ControlTemplate. This special resource is applied at run time. Every box and button resides in the ControlTemplate. A control's template can be easily accessed by the Template property. That Template property isn't read-only. Developers can set it to a custom ControlTemplate, transforming a control's visuals and behavior to meet their particular needs. That's the power of re-templating:

```
<ControlTemplate>
  <StackPanel Padding="20">
    <ContentControl Content="{TemplateBinding Content}" />
    <CheckBox>I agree!</CheckBox>
    <Button>Submit1</Button>
  </StackPanel>
</ControlTemplate>
```

ControlTemplate XAML looks like any other layout declaration. In the preceding code, notice the special TemplateBinding markup extension. This special binding is tuned for one-way template operations. Since Windows 10 version 1809, x:Bind syntax is supported in UWP ControlTemplate definitions. This allows for performant, compiled, two-way and function bindings in templates. TemplateBinding works great in most cases.

Generic.xaml To create a templated control, select Templated Control in the Add New Item dialog. This introduces three files: the XAML file; its codebehind; and themes/generic.xaml, which holds the ControlTemplate. The themes/generic.xaml file is identical to WPF. It's special. The framework merges it into your app's resources automatically. Resources defined here are scoped at the application level:

```
<Style TargetType="controls:MyControl">
  <Setter Property="Template">
    <Setter.Value>
      <ControlTemplate TargetType="controls:MyControl" />
    </Setter.Value>
  </Setter>
</Style>
```

ControlTemplates are applied using implicit styles: styles without a key. Explicit styles have a key used to apply them to controls; implicit styles are applied based on TargetType. So, you must set DefaultStyleKey:

```
public sealed class MyControl : Control
{
  public MyControl() => DefaultStyleKey = typeof(MyControl);
}
```

This code sets the DefaultStyleKey, determining which style is implicitly applied to your control. I'm setting it to the corresponding value of the TargetType in the ControlTemplate:

```
<ControlTemplate TargetType="controls:MyControl">
  <StackPanel Padding="20">
    <TextBlock Text="{TemplateBinding Text}" />
    <CheckBox Name="AgreeCheckbox">I agree!</CheckBox>
    <Button IsEnabled="{Binding IsChecked,
      ElementName=AgreeCheckbox}">Submit</Button>
  </StackPanel>
</ControlTemplate>
```

TemplateBinding binds the TextBlock's Text property to the custom dependency property copied from the user control to the templated control. TemplateBinding is one way, very efficient and generally the best option.

Figure 4 shows the result of my work in the designer. The custom layout declared in the ControlTemplate is applied to my custom

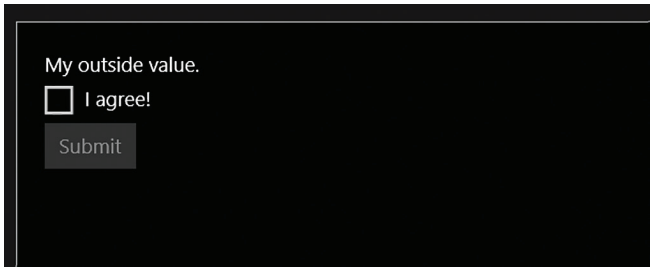


Figure 4 **Preview Transparently Using Internal Properties**

Figure 5 **Using the Class Attribute to Set the Content Property**

```
[ContentProperty(Name = "Text")]
public sealed class MyControl : Control
{
  public MyControl() => DefaultStyleKey = typeof(MyControl);
  public string Text
  {
    get => (string)GetValue(TextProperty);
    set => SetValue(TextProperty, value);
  }
  public static readonly DependencyProperty TextProperty =
    DependencyProperty.Register(nameof(Text), typeof(string),
      typeof(MyControl), new PropertyMetadata(default(string)));
}
```

control, and the binding is executed and rendered at design time:

```
<controls:MyControl Text="My outside value." />
```

The syntax to use my custom control is simple. I'll make it better by allowing the developer to use inline text. This is the most intuitive syntax to set an element's content. XAML provides a class attribute to help me do that, as **Figure 5** shows.

Notice the ContentProperty attribute, which comes from the Windows.UI.Xaml.Markup namespace. It indicates to which property direct, inline content declared in XAML should be written. So, now, I can declare my content like this:

```
<controls:MyControl>
  My inline value!
</controls:MyControl>
```

It's beautiful. Templated controls give you the flexibility to design control interaction and syntax in whatever way seems most intuitive to you. **Figure 6** shows the result of introducing ContentProperty to the control.

The ContentControl Where previously I had to create a custom property and map that property to the content of the control, XAML provides a control already built for that. It's known as ContentControl and its property is called Content. ContentControl also provides the ContentTemplate and ContentTransition properties to handle visualization and transitions. Button, CheckBox, Frame and many standard XAML controls inherit ContentControl. Mine could have, too; I'd just use Content instead of Text:

```
public sealed class MyControl2 : ContentControl
{
  // Empty
}
```

In this code, notice the terse syntax to create a custom control with a Content property. ContentControl auto-renders as a ContentPresenter when declared. It's a fast, easy solution. There's a caveat, however: ContentControl doesn't support literal strings in XAML. Because it violates my goal to make my control support literal strings, I'll stick to Control, considering ContentControl another time.

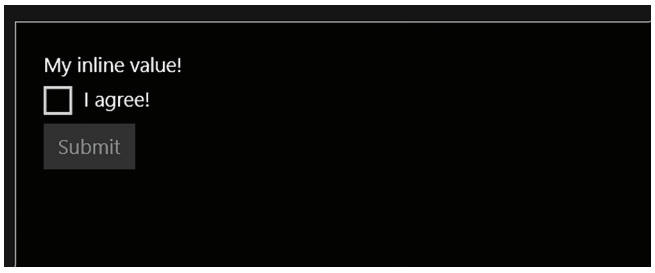


Figure 6 **Design Preview**

Accessing Internal Controls The x:Name directive declares the field name auto-generated in a codebehind. Give a checkbox an x:Name of MyCheckBox and the generator will create a field in your class called MyCheckBox.

By contrast, x:Key (for resources only) doesn't create a field. It adds resources to a dictionary of unresolved types until they're first used. For resources, x:Key offers performance improvements.

Because x:Name creates a backing field, it can't be used in a ControlTemplate; templates are decoupled from any backing class. Instead, you use the Name property.

Name is a dependency property on FrameworkElement, an ancestor of Control. You use it when you can't use x:Name. Name and x:Name are mutually exclusive in a given scope because of FindName.

FindName is a XAML method used to locate objects by name; it works with Name or x:Name. It's reliable in the codebehind, but not in templated controls where you must use GetTemplateChild:

```
protected override void OnApplyTemplate()
{
  if (GetTemplateChild("AgreeCheckbox") is CheckBox c)
  {
    c.Content = "I really agree!";
  }
}
```

GetTemplateChild is a helper method used in the OnApplyTemplate override to find controls created by a ControlTemplate. Use it to find references to internal controls.

Changing the Template Re-templating the control is simple, but I've built the class to expect controls with certain names. I must ensure the new template maintains this dependency. Let's build a new ControlTemplate:

```
<ControlTemplate
  TargetType="controls:MyControl"
  x:Key="MyNewStyle">
```

Small changes to a ControlTemplate are normal. You don't need to start from scratch. In the Visual Studio Document Outline pane, right-click any control and extract a copy of its current template (see **Figure 7**).

If I maintain its dependencies, my new ControlTemplate can totally change the visuals and behaviors of a control. Declaring an explicit style on the control tells the framework to ignore the default implicit style:

```
<controls:MyControl Style="{StaticResource MyControlNewStyle}">
```

But this rewriting of the ControlTemplate comes with a warning. Control designers and developers must be careful to support accessibility and localization capabilities. It's easy to remove these by mistake.

TemplatePartAttribute It would be handy if a custom control could communicate the named elements it expects. Some named

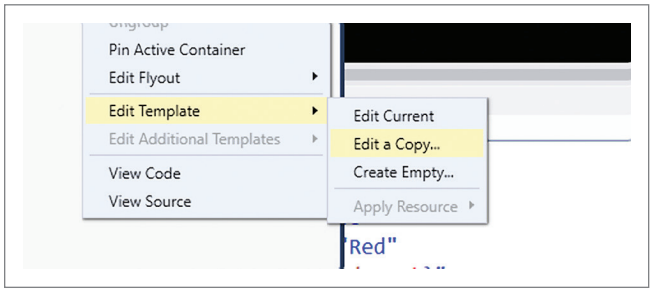


Figure 7 **Extracting a Control Template**

elements may be required only during edge cases. In WPF, you have TemplatePartAttribute:

```
[TemplatePart (
  Name = "EulaTextBlock",
  Type = typeof(TextBlock))]
public sealed class MyControl : Control { }
```

This syntax shows how the control can communicate internal dependencies to external developers. In this case, I expect a TextBlock with the name EulaTextBlock in my ControlTemplate. I can also specify the visual states I expect in my custom control:

```
[TemplateVisualState(
  GroupName = "Visual",
  Name = "Mouseover")]
public sealed class MyControl : Control { }
```

TemplatePart is used by Blend with TemplateVisualState to guide developers against expectations while creating custom templates. A ControlTemplate can be validated against these attributions. Since 10240, WinRT has included these attributes. UWP can use them, but Blend for Visual Studio doesn't. These remain a good, forward-looking practice, but documentation is still the best approach.

Accessibility First-party XAML controls are meticulously designed and tested to be beautiful, compatible and accessible. Accessibility requirements are now first-class citizens and are release requirements for every control.

> Developers around the world love XAML because it's so productive and flexible.

When you re-template a first-party control, you put at risk the accessibility features thoughtfully added by the development teams. These are difficult to get right, and simple to get wrong. When choosing to re-template a control, you should be well-versed in the accessibility capabilities of the framework and in the techniques to implement them. Otherwise, you lose a considerable portion of their value.

Adding accessibility as a release requirement helps not only those with permanent disabilities, but also those who are temporarily incapacitated. It also reduces risk when re-templating first-party controls.

You Did It!

After upgrading from a user control to a templated control, I introduced very little new code. But I've added a lot of functionality. Let's consider what's been accomplished overall.

Encapsulation. The control is a collection of several controls, bundled with custom visuals and behaviors that consuming developers can reuse with ease across an application.

Business logic. The control incorporates business rules that fulfill the acceptance criteria of the user story. I've put immutable rules close to the control and supported a rich design-time experience, too.

Custom events. The control exposes control-specific custom events like "click" that help developers interoperate with the control, without requiring them to understand the internal structure of the layout.

Custom properties. The control exposes properties to allow the consuming developer to influence the content of the layout. This has been done in a way to fully support XAML data binding.

API syntax. The control supports an intuitive approach that allows developers to declare their content with literal strings and in a straightforward way. I leveraged the ContentProperty attribute to do this.

Templates. The control ships with a default ControlTemplate that lays out an intuitive interface. However, XAML re-templating is supported to allow consumer developers to customize the visuals as needed.

There's More to Do

My control needs more, but not much more— just a little attention to layout (like the need to scroll large text) and a few properties (like the content of the checkbox). I'm amazingly close.

Controls can support Visual State Management, a native feature of XAML that allows properties to change based on sizing events or framework events (like mouseover). Mature controls have visual states.

Controls can support localization; the native capability in UWP uses the x:Uid directive associating controls with RESW strings that are filtered by the active locale. Mature controls support localization.

Controls can support external-style definitions to help update their visuals without requiring a new template; this can involve shared visuals and leverage themes and BasedOn styles. Mature controls share and reuse styles.

Wrapping Up

Prototyping a UI in XAML is fast. User controls create simple, reusable layouts easily. Templated controls require a bit more work to create simple, reusable layouts with more sophisticated capabilities. The right approach is up to the developer, based on a little knowledge and a lot of experience. Experiment. The more you learn the tooling, the more productive you'll become.

Windows Forms succeeded Visual Basic 6 in 2002, just as WPF succeeded Windows Forms in 2006. WPF brought with it XAML: a declarative language for UI. Microsoft developers had never seen anything like XAML. Today, Xamarin and UWP bring XAML to iOS, Android, HoloLens, Surface Hub, Xbox, IoT and the modern desktop. In fact, XAML is now the technology building the Windows OS itself.

Developers around the world love XAML because it's so productive and flexible. Microsoft engineers feel the same; we're building our own apps and even Windows 10 with XAML. The future is bright, the tooling is powerful and the technology is more approachable than ever. ■

Jerry Nixon *is a senior software engineer & lead architect in Commercial Software Engineering at Microsoft. He has developed and designed software for two decades. Speaker, organizer, teacher and author, Nixon is also the host of DevRadio. Most of his days are spent teaching his three daughters "Star Trek" backstories and episode plots.*

Thanks to the following Microsoft technical experts for reviewing this article: *Daniel Jacobson, Dmitry Lyalin, Daren May, Ricardo Minguez Pablos*

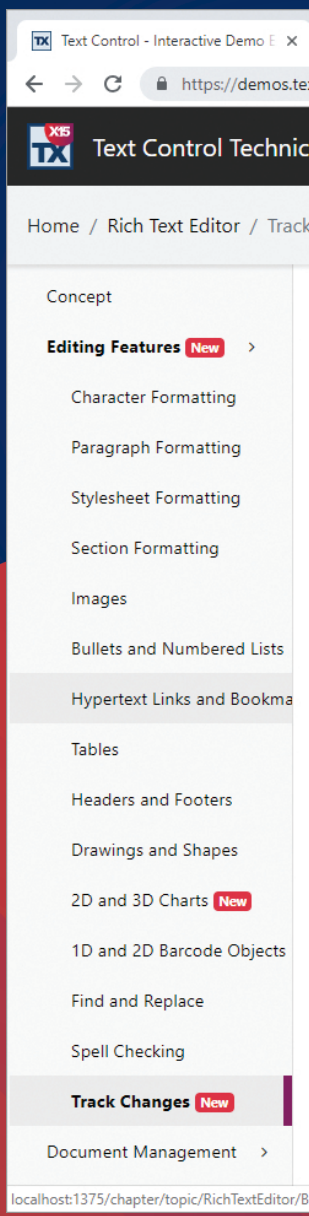
TEXT CONTROL

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

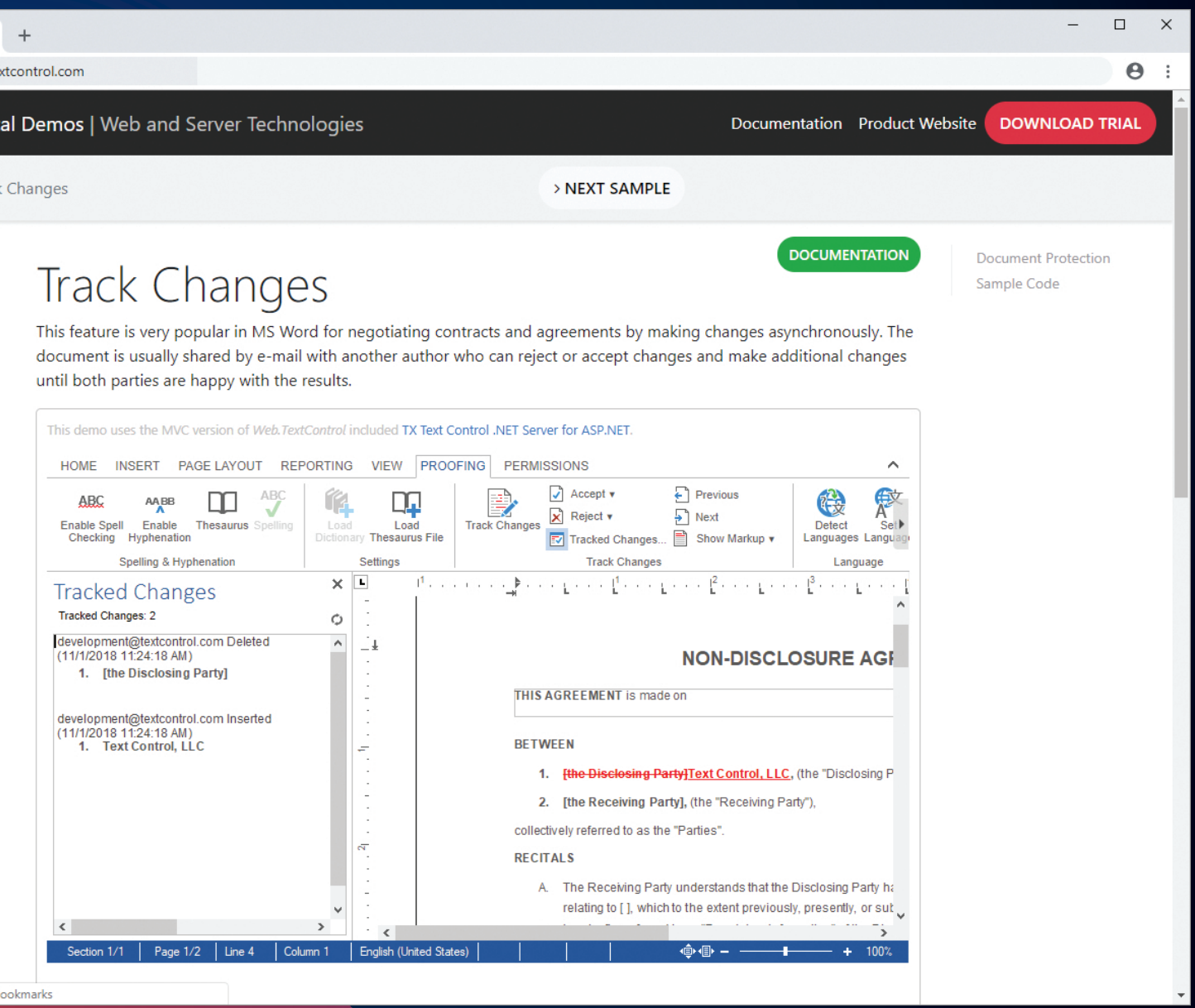


WE ARE CHANGING THE WAY YOU LOOK AT REPORTING

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!



Create a Centralized Pull Request Hub with WinForms in .NET Core 3.0

Eric Fleming

Windows Forms, or simply WinForms, has been used for years to develop powerful Windows-based applications with rich and interactive interfaces. The investments in these desktop applications across businesses of every type are profound, with some 2.4 million developers using Visual Studio to create desktop-style apps every month. The benefits of leveraging and extending existing WinForms code assets are compelling, but there are others, as well. The WinForms drag-and-drop designer experience empowers users to build fully functional UIs with no special knowledge or training. WinForms applications are easy to deploy and update, can work independent of Internet connectivity, and can offer improved security by running on a local machine that doesn't expose configurations to the Internet. Until recently, WinForms applications could only be built using the full .NET Framework, but the release of the .NET Core 3.0 preview changes all that.

The new features and benefits of .NET Core go beyond Web development. With .NET Core 3.0, WinForms adds capabilities like easier deployment, improved performance, support for .NET Core-specific NuGet packages, the .NET Core command-line interface (CLI) and more. Throughout this article, I'll touch on many of these benefits, why they matter and how to use them in WinForms applications.

This article discusses:

- Creating a WinForms project in .NET Core 3.0
- Creating a self-contained deployable package
- Benefits of using .NET Core 3.0 for WinForms applications

Technologies discussed:

.NET Core 3.0, Windows Forms

Let's jump right into building our first .NET Core 3.0 WinForms application. For this article, I'm going to build an application that retrieves and displays open pull requests for one of the open source Microsoft repositories hosted on GitHub. The first step is to install the latest versions of Visual Studio 2019 and the .NET Core 3.0 SDK, after which you'll have access to the .NET Core CLI commands to create a new WinForms application. This wasn't possible for WinForms applications before the addition of .NET Core support.

Coming soon is a new Visual Studio template that lets you create a WinForms project targeting .NET Core 3.0. It's not available just yet, so for now let's generate a new WinForms project named PullRequestHub by running the following command:

```
dotnet new winforms -o PullRequestHub
```

In order to ensure that the project was created successfully, navigate into the new directory created by the dotnet new command and use the CLI to build and run the project, like so:

```
cd .\PullRequestHub\
```

Because you have access to the .NET Core CLI, you also have access to the commands to restore, run and build. Before running, try out the restore and build commands, as follows:

```
dotnet restore
dotnet build
```

These commands work just as they would when run in the command line for a .NET Core Web application. And note that when you execute the dotnet run command, it actually performs both a restore and a build before executing the app (bit.ly/2UCkEaN). Now let's run the project to test it out by entering dotnet run at the command line.

Success! You've just created your first .NET Core WinForms application. When you run, you'll see a form appear on your screen with the text, "Hello .NET Core!".

Before I go further with adding logic to our application, let's take a moment to talk about the current state of the WinForms Designer view in Visual Studio.

Setting up the Designer for .NET Core WinForms Apps

When you open the CLI-generated project in Visual Studio, you may notice that some functionality is missing. Most notably, there's currently no designer view provided for .NET Core WinForms applications. While there are plans to make this functionality available, they've yet to be completed.

Fortunately, there's a workaround that can give you access to a designer, at least until native support is added. For now, you can create a .NET Framework project that contains your UI files. This way you can edit the UI files using the designer, and the .NET Core project will then reference the UI files from the .NET Framework project. This enables you to leverage the UI capabilities while still building the application in .NET Core. Here's how I do this for my project.

In addition to the PullRequestHub project you created, you'll want to add a new WinForms project running on a version of .NET Full-Framework. Name this project PullRequestHub.Designer. After the new project is created, remove the Form1 files from the .NET Core project, leaving only the Program.cs class.

Navigate into the PullRequestHub.Designer and rename the form files to PullRequestForm. Now you'll edit the .NET Core project file and add the following code to link the files together in both projects. This will take care of any additional forms or resources you create in the future, as well:

```
<ItemGroup>
  <Compile Include="..\PullRequestHub.Designer\**\*.cs" />
</ItemGroup>
```

Once you save the project file, you'll see the PullRequestForm files appear in the solution explorer and you'll be able to interact with them. When you want to use the UI editor, you'll need to make sure to close the PullRequestForm file from the .NET Core project and open the PullRequestForm file from the .NET Framework project. The changes will take place in both, but the editor is only available from the .NET Framework project.

Building the Application

Let's start adding some code to the application. In order to retrieve open pull requests from GitHub, I need to create an HttpClient. This is where .NET Core 3.0 comes in, because it provides access to the new HttpClientFactory. The HttpClient in the full-framework version had some issues, including one with creating the client with a using statement. The HttpClient object would be disposed of, but the underlying socket wouldn't be released for some time, which by default is 240 seconds. If the socket connection remains open for 240 seconds and you have a high throughput in your system, there's a chance your system saturates all the free sockets. When this happens, new requests must wait for a socket to free up, which can produce some rather drastic performance impacts.

The HttpClientFactory helps mitigate these issues. For one, it gives you an easier way to pre-configure client implementations in a more central location. It also manages the lifetime of the HttpClients for you, so you don't run into the previously mentioned issues. Let's look at how you can do this now in a WinForms application.

One of the best and easiest ways to use this new feature is through dependency injection. Dependency injection, or more generally inversion of control, is a technique for passing dependencies into classes. It's also a fantastic way to reduce the coupling of classes and to ease unit testing. For example, you'll see how you can create an instance of the IHttpClientFactory while the program is starting up, with that object able to be leveraged later in the form. This was not something very easily accomplished in WinForms on previous versions of .NET, and it's another advantage of using .NET Core.

In the Program.cs you're going to create a method named ConfigureServices. In this method, create a new ServiceCollection to make services available to you via dependency injection. You'll need to install the latest of these two NuGet packages first:

```
- 'Microsoft.Extensions.DependencyInjection'
- 'Microsoft.Extensions.Http'
```

Then add the code shown in **Figure 1**. This creates a new IHttpClientFactory to be used in your forms. The result is a client that you can explicitly use for requests involving the GitHub API.

Next, you need to register the actual form class, PullRequestForm, as a singleton. To the end of this method, add the following line:

```
services.AddSingleton<PullRequestForm>();
```

Then you need to create an instance of the ServiceProvider. At the top of the Program.cs class, create the following property:

```
private static IServiceProvider ServiceProvider { get; set; }
```

Now that you have a property for your ServiceProvider, at the end of the ConfigureServices method, add a line to build the ServiceProvider, like so:

```
ServiceProvider = services.BuildServiceProvider();
```

At the end of all of this, the full ConfigureServices method should look like the code in **Figure 2**.

Now you need to wire the form up with the container when it starts. When the application runs, this will invoke the PullRequestForm with the necessary services available to it. Change the Main method to be the following code:

```
[STAThread]
static void Main()
{
  Application.EnableVisualStyles();
  Application.SetCompatibleTextRenderingDefault(false);
  ConfigureServices();
  Application.Run((PullRequestForm)ServiceProvider.
GetService(typeof(PullRequestForm)));
}
```

Figure 1 **Create a New IHttpClientFactory**

```
private static void ConfigureServices()
{
  var services = new ServiceCollection();
  services.AddHttpClient();
  services.AddHttpClient("github", c =>
  {
    c.BaseAddress = new Uri("https://api.github.com/");
    c.DefaultRequestHeaders.Add("Accept", "application/vnd.github.v3+json");
    c.DefaultRequestHeaders.Add("User-Agent", "HttpClientFactory-Sample");
    c.DefaultRequestHeaders.Add("Accept", "application/json");
    ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls12;
  });
}
```

Great! Now you're all wired up. In the PullRequestForm constructor, you're going to inject the IHttpClientFactory you just wired up and assign it to a local variable, as shown in the code here:

```
private static HttpClient _httpClient;

public PullRequestForm(IHttpClientFactory httpClientFactory)
{
  InitializeComponent();
  _httpClient = httpClientFactory.CreateClient("github");
}
```

You now have an HttpClient you can use to make calls out to GitHub to retrieve pull requests, issues and the like. This also makes the next few steps a little tricky. The calls from the HttpClient are going to be async requests, and if you've been using WinForms, you know what's coming. You're going to have to handle the threading, and send dispatch updates to the UI thread.

In order to kick off retrieving all of the pull requests, let's add a button to the view. This way you can, in the future, add more repos or more groups of repos to check. Using the designer that you wired up, drag the button on to the form and rename the text to read "Microsoft." While you're at it, give your button a more meaningful name like RetrieveData_Button. You'll need to tie in to the RetrieveData_Button_Click event, but you need it to be async, using this code:

```
private async void RetrieveData_Button_Click(object sender, EventArgs e)
{

}
```

Here's where you'll want to call a method that retrieves open GitHub pull requests. But first, because you're dealing with async calls now, you must wire up the SynchronizationContext. You do this by adding a new property and updating the constructor with the following code:

```
private static HttpClient _httpClient;
private readonly SynchronizationContext synchronizationContext;

public PullRequestForm(IHttpClientFactory httpClientFactory)
{
  InitializeComponent();
  synchronizationContext = SynchronizationContext.Current;
  _httpClient = httpClientFactory.CreateClient("github");
}
```

Next, create a model named PullRequestData, so you can easily deserialize your request. Here's the code for that:

```
public class PullRequestData
{
  public string Url { get; set; }
  public string Title { get; set; }
}
```

Finally, create a method named GetPullRequestData. In this method, you're going to make your request to the GitHub API and retrieve all the open pull requests. You'll be deserializing a JSON request, so add the latest version of Newtonsoft.Json package to the project. Here's the code:

```
private async Task<List<PullRequestData>> GetPullRequestData()
{
  var gitHubResponse =
    await _httpClient.GetStringAsync(
    $"repos/dotnet/winforms/pulls?state=open");
  var gitHubData =
    JsonConvert.DeserializeObject<List<PullRequestData>>(gitHubResponse);
  return gitHubData;
}
```

This can now be invoked from the RetrieveData_Button_Click method. Once you have a list of the data you want, create a list of labels for each Title so you can display it on the form. After you have the list of labels, you can add them to the UI in the UpdateUI method. **Figure 3** shows this.

Your UpdateUI method will then use the synchronizationContext to update the UI, like so:

```
public void UpdateUI(List<Label> labels)
{
  synchronizationContext.Post(new SendOrPostCallback(o =>
  {
    foreach (var label in labels)
    {
      Controls.Add(label);
    }

  }), labels);
}
```

If you run the application and click the Microsoft button, the UI will be updated with the titles of all the open pull requests from the dotnet/winforms repository on GitHub.

Now it's your turn. To truly make this a centralized pull request hub, as the title of this article promises, let's update this example to read from multiple GitHub repositories. These repositories don't need to be from the Microsoft team, though it's fun to watch their progress. For example, microservices architectures are very common, and in them you may have numerous repositories that make up your system as a whole. Given that it's usually a good idea to not leave branches and pull requests out there, unmerged, for too long, a tool like this could boost your insight into open pull requests and improve the quality of your entire system.

You could certainly set up a Web app, but then you'd have to worry about deployments, where it's going to run, authentication

Figure 2 **The ConfigureServices Method**

```
private static void ConfigureServices()
{
  var services = new ServiceCollection();
  services.AddHttpClient();
  services.AddHttpClient("github", c =>
  {
    c.BaseAddress = new Uri("https://api.github.com/");
    c.DefaultRequestHeaders.Add("Accept", "application/vnd.github.v3+json");
    c.DefaultRequestHeaders.Add("User-Agent", "HttpClientFactory-Sample");
    c.DefaultRequestHeaders.Add("Accept", "application/json");
    ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls12;
  });
  services.AddSingleton<PullRequestForm>();
  ServiceProvider = services.BuildServiceProvider();
}
```

Figure 3 **Invoking from RetrieveData_Button_Click**

```
private async void RetrieveData_Button_Click(object sender, EventArgs e)
{
  var pullRequestData = await GetPullRequestData();

  await Task.Run(() =>
  {
    var labelsToAdd = new List<Label>();
    var verticalSpaceBetweenLabels = 20;
    var horizontalSpaceFromLeft = 10;
    for (int i = 0; i < pullRequestData.Count; i++)
    {

      Label label = new Label();
      label.Text = pullRequestData[i].Title;
      label.Left = horizontalSpaceFromLeft;
      label.Size = new Size(100, 10);
      label.AutoSize = true;
      label.Top = (i * verticalSpaceBetweenLabels);
      labelsToAdd.Add(label);
    }
    UpdateUI(labelsToAdd);
  });
}
```

Visual Studio LIVE!

EXPERT SOLUTIONS FOR ENTERPRISE DEVELOPERS

Developer Training Conferences and Events

In-depth Technical Content On:

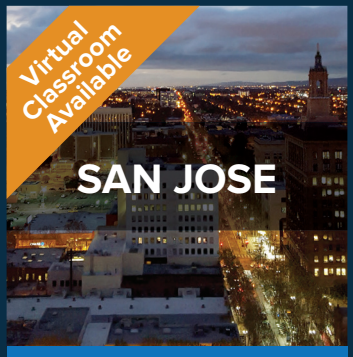
- AI, Data and Machine Learning
- Cloud, Containers and Microservices
- Delivery and Deployment
- Developing New Experiences
- DevOps in the Spotlight
- Full Stack Web Development
- .NET Core and More

April 22-26
Hyatt Regency

REGISTER NOW!
vslive.com/neworleans

June 9-13
Hyatt Regency Cambridge

REGISTER NOW!
vslive.com/boston



June 18-19
Microtek Training Center San Jose

REGISTER NOW!
vslive.com/sanjose

July 23
Developer's Guide to Getting Started with UX

REGISTER NOW!
vslive.com/virtual

August 12-16
Microsoft HQ

REGISTER NOW!
vslive.com/microsofthq

Sept. 29-Oct. 3
Westin Gaslamp Quarter

REGISTER NOW!
vslive.com/sandiego

October 6-10
Swissotel

REGISTER NOW!
vslive.com/chicago

November 17-22
Royal Pacific Resort at Universal

REGISTER NOW!
vslive.com/orlando

SUPPORTED BY

Microsoft | Visual Studio | msdn magazine | Visual Studio MAGAZINE

PRODUCED BY

CONVERGE 360 An 1105 MEDIA company

vslive.com #VSLIVE

and the like. With a WinForms application in .NET Core, you don't have to worry about any of this. Let's now take a look at one of the biggest advantages of building a WinForms app using .NET Core.

Packaging the Application

In the past, deploying a new or updated WinForms application could cause problems related to the version of the .NET Framework installed on the host machines. With .NET Core, apps can be deployed self-contained and run from a single folder, with no dependencies to the version of .NET Framework installed on the machine. This means the user doesn't need to install *anything*; they can simply run the application. It also enables you to update and deploy apps one at a time, because the packaged versions of .NET Core will not affect each other.

For the sample app in this article, you'll want to package it up as self-contained. Keep in mind that self-contained applications will be larger because they include the .NET Core libraries with them. If you're deploying to machines with the latest versions of .NET Core installed, you don't need to make the app self-contained. Instead, you can reduce the size of the deployed app by leveraging the installed version of .NET Core. Self-contained options are for when you don't want your application to be dependent on the environment in which it will be running.

To package the application locally, you need to make sure Developer Mode is enabled in your settings. Visual Studio will prompt you and give you a link to the settings when you try to run the packaging project, but to enable it directly go to your Windows settings, press the Windows key and search for Settings. In the search box type "For developers settings" and select it. You'll see an option to enable Developer Mode. Select and enable this option.

For the most part, the steps to create a self-contained package will seem familiar if you've previously packaged a WinForms application. First, start by creating a new Windows Application Packaging Project. Name the new project PullRequestHubPackaging. When prompted to select the target and minimum platform versions, use the defaults and click OK. Right-click on Applications and add a reference to the PullRequestHub project.

After the reference has been added, you need to set the PullRequestHub project as the Entry Point. Once that's done, the next time you build you'll very likely see the following error: "Project PullRequestHub must specify 'RuntimeIdentifiers' in the project file when 'SelfContained' is true."

To fix this error, edit the PullRequestHub.csproj file. When you open this project file, you'll notice yet another advantage of using .NET Core, because the project file is now using the new, lightweight format. In .NET Framework-based WinForms projects, the project file is much more verbose with explicit defaults and references, as well as NuGet references split out into a packages.config file. The new project file format brings package references into the project file, making it possible to manage all your dependencies in one place.

In this file, in the first PropertyGroup node, add the following line:

```
<RuntimeIdentifiers>win-x86</RuntimeIdentifiers>
```

A Runtime Identifier is used to identify target platforms where the application runs, and is used by .NET packages to represent platform-specific assets in NuGet packages. Once this is added, the build should succeed, and you can set the PullRequestHubPackaging project as the startup project in Visual Studio.

One thing to note in the PullRequestHubPackaging.wapproj file is the setting to indicate that the project is self-contained. The section of code in the file to pay attention to is the following:

```
<ItemGroup>
  <ProjectReference Include="..\PullRequestHub\PullRequestHub.csproj">
    <DesktopBridgeSelfContained>True</DesktopBridgeSelfContained>
    <DesktopBridgeIdentifier>$(DesktopBridgeRuntimeIdentifier)
    </DesktopBridgeIdentifier>
      <Properties>SelfContained=%(DesktopBridgeSelfContained);
        RuntimeIdentifier=%(DesktopBridgeIdentifier)
      </Properties>
    <SkipGetTargetFrameworkProperties>True
    </SkipGetTargetFrameworkProperties>
  </ProjectReference>
</ItemGroup>
```

Here you can see that the DesktopBridgeSelfContained option is set to true, which enables the WinForms application to be packaged with the .NET Core binaries. When you run the project, it dumps the files out to a folder named "win-x86" found in a path similar to this:

```
C:\Your-Path\PullRequestHub\PullRequestHub\bin\x86\Debug\netcoreapp3.0
```

Inside of the win-x86 folder you'll notice many DLLs, which include everything that the self-contained app needs to run.

More likely, you'll want to deploy the app as a side-loaded application or upload it to the Microsoft Store. Side-loading will make automatic updates possible using an appinstaller file. These updates are supported starting with Visual Studio 2017, Update 15.7. You can also create packages that support submission to the Microsoft Store for distribution. The Microsoft Store then handles all the code signing, distribution and updating of the app.

In addition to these options, there's ongoing work to make it possible for applications to be packaged up into a single executable, eliminating the need to populate an output directory with DLLs.

Additional Advantages

With .NET Core 3.0, you're also able to leverage features of C# 8.0, including nullable reference types, default implementations on interfaces, improvements to switch statements using patterns, and asynchronous streams. To enable C# 8.0, open the PullRequestHub.csproj file and add the following line to the first PropertyGroup:

```
<LangVersion>8.0</LangVersion>
```

Another advantage to using .NET Core and WinForms is that both projects are open source. This gives you access to the source code, and lets you file bugs, share feedback and become a contributor. Check out the WinForms project at github.com/dotnet/winforms.

.NET Core 3.0 promises to breathe new life into the investments that enterprises and businesses have made into WinForms applications, which continue to be productive, reliable, and easy to deploy and maintain. Developers can leverage new .NET Core-specific classes like HttpClientFactory, employ C# 8.0 features like nullable reference types, and package up self-contained applications. They also gain access to the .NET Core CLI and all the performance improvements that come with .NET Core. ■

Eric Fleming *is a senior software engineer with more than a decade of experience working with Microsoft tools and technologies. He blogs at ericflemingblog.com and co-hosts the Function Junction YouTube channel that explores Azure Functions. Follow him on Twitter: @efleming18.*

Thanks to the following technical experts who reviewed this article: *Olia Gavrysh (Microsoft), Simon Timms*

Visual Studio LIVE!
EXPERT SOLUTIONS FOR ENTERPRISE DEVELOPERS

CHICAGO

Navigate Today's Tech in the Windy City

OCTOBER 6-10, 2019
SWISSOTEL, CHICAGO, IL

#VSLive

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!

On-Demand Session Recordings Now Available

Get on-demand access for one full year to all keynotes and sessions from Visual Studio Live! Chicago, including everything Tuesday – Thursday at the conference.

SAVE $400! When You Register by August 2

Your Adventure Starts Here!

EVENT PARTNER: Microsoft

GOLD SPONSOR: CSC corptax.com

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

PRODUCED BY: CONVERGE 360 An 1105 MEDIA company

vslive.com/chicago

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

Save $400
When You Register By June 14!
Use Promo Code MSDN

vslive.com/microsofthq

GOLD SPONSOR
Progress

SUPPORTED BY
Microsoft | Visual Studio | msdn magazine | Visual Studio MAGAZINE

PRODUCED BY
CONVERGE360 | 1105 MEDIA

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, August 12, 2019 *(Separate entry fee required)*			
7:00 AM	8:30 AM	Pre-Conference Workshop Registration - Coffee and Morning Pastries			
8:30 AM	12:30 PM	**HOL01** Hands-On Lab: Hands-On With Cloud-Native .NET Development - *Rockford Lhotka and Richard Seroter*	**HOL02** Hands-On Lab: Leveling Up—Dependency Injection in .NET - *Jeremy Clark*	**HOL03** Full Day Hands-On Lab: Xamarin and Azure: Build the Mobile Apps of Tomorrow - *Laurent Bugnion*	
12:30 PM	2:00 PM	Lunch @ The Mixer - Visit the Microsoft Company Store & Visitor Center			
2:00 PM	5:30 PM	**HOL01** Hands-On Lab Continued - *Rockford Lhotka and Richard Seroter*	**HOL02** Hands-On Lab Continued - *Jeremy Clark*	**HOL03** Hands-On Lab Continued - *Laurent Bugnion*	
7:00 PM	9:00 PM	Dine-A-Round Dinner			
START TIME	**END TIME**	**Visual Studio Live! Day 1: Tuesday, August 13, 2019**			
7:00 AM	8:00 AM	Registration - Coffee and Morning Pastries			
8:00 AM	9:15 AM	**T01** Angular 101 - *Deborah Kurata*	**T02** Xamarin - *James Montemagno*	**T03** SQL Server 2019 Deep Dive - *Scott Klein*	**T04** C# and/or .NET Core 3.0 - *Dustin Campbell*
9:30 AM	10:45 AM	**T05** Moving to ASP.NET Core 2.X - *Philip Japikse*	**T06** Sharing C# Code Across Platforms - *Rockford Lhotka*	**T07** Application Modernization with Microsoft Azure - *Michael Crump*	**T08** DevOps Practices Can Make You A Better Developer - *Robert Green*
10:45 AM	11:15 AM	Sponsored Break - Visit Exhibitors - *Foyer*			
11:15 AM	12:15 PM	**KEYNOTE: To Be Announced** - *James Montemagno, Principal Program Manager - Mobile Developer Tools, Microsoft*			
12:15 PM	1:30 PM	Lunch - McKinley / Visit Exhibitors - *Foyer*			
1:30 PM	2:45 PM	**T09** Angular Best Practices - *Deborah Kurata*	**T10** To Be Announced	**T11** Data Pipelines with End-2-End Azure Analytics - *Scott Klein*	**T12** Get Func-y: Understanding Delegates in .NET - *Jeremy Clark*
3:00 PM	4:15 PM	**T13** Versioning ASP.NET Core APIs -*Philip Japikse*	**T14** Visual Studio for Mac: From Check-out to Publish - *Dominic Nahous and Cody Beyer*	**T15** Why, When, and How to Enhance Your App with Azure Functions - *Daria Grigoriu & Eamon O'Reilly*	**T16** CI / CD with Azure DevOps - *Tiago Pascoal*
4:15 PM	5:45 PM	Microsoft Ask the Experts & Welcome Reception			
START TIME	**END TIME**	**Visual Studio Live! Day 2: Wednesday, August 14, 2019**			
7:30 AM	8:00 AM	Registration - Coffee and Morning Pastries			
8:00 AM	9:15 AM	**W01** Getting Pushy with SignalR and Reactive Extensions - *Jim Wooley*	**W02** AI and Analytics with Apache Spark on Azure Databricks - *Andrew Brust*	**W03** Building Business Application Bots - *Michael Washington*	**W04** DevOps with ASP.NET Core, EF Core, & Azure DevOps - *Benjamin Day*
9:30 AM	10:45 AM	**W05** Building Reactive Client Experiences with RxJs and Angular Forms *Jim Wooley*	**W06** Power BI: What Have You Done for Me Lately - *Andrew Brust*	**W07** Windows Subsystem for Linux - *Rich Turner and Craig Loewen*	**W08** Real World Scrum with Azure DevOps - *Benjamin Day*
11:00 AM	12:00 PM	**GENERAL SESSION:** .NET Core 3 - *Scott Hunter, Director of Program Management, Microsoft*			
12:00 PM	1:30 PM	Birds-of-a-Feather Lunch - McKinley / Visit Exhibitors - Foyer			
1:30 PM	2:45 PM	**W09** C# in the Browser with Client-side Blazor and WebAssembly - *Rockford Lhotka*	**W10** UX Design Fundamentals: What do Your Users Really See? - *Billy Hollis*	**W11** Works On My Machine... Docker for Developers - *Chris Klug*	**W12** Bolting Security Into Your Development Process - *Tiago Pascoal*
2:45 PM	3:15 PM	Sponsored Break - Exhibitor Raffle @ 2:55 pm (Must be present to win) - Foyer			
3:15 PM	4:30 PM	**W13** Creating Business Applications Using Blazor (Razor Components) - *Michael Washington*	**W14** What's New with Azure App Service? - *Christina Compy*	**W15** Microservices with Azure Kubernetes Service (AKS) - *Vishwas Lele*	**W16** Database DevOps with SQL Server - *Brian Randell*
6:15 PM	9:00 PM	VSLive!'s Downtown Seattle Adventure			
START TIME	**END TIME**	**Visual Studio Live! Day 3: Thursday, August 15, 2019**			
7:30 AM	8:00 AM	Registration - Coffee and Morning Pastries			
8:00 AM	9:15 AM	**TH01** Angular Application Testing Outside the Church of TDD - *Chris Klug*	**TH02** Add Native to Cross Platform with Xamarin.Essentials - *Veronika Kolesnikova*	**TH03** Best Practices for Serverless DevOps: Inner Loop, Outer Loop, Observability - *Colby Tresness*	**TH04** Visual Studio Productivity Tips and Tricks - *Allison Buchholtz-Au*
9:30 AM	10:45 AM	**TH05** Any App, Any Language with Visual Studio Code and Azure DevOps - *Brian Randell*	**TH06** MVVM Made Easy for WPF Applications - *Paul Sheriff*	**TH07** Cross Platform Automation with PowerShell Core 6.0	**TH08** Exceptional Development: Dealing With Exceptions in .NET - *Jason Bock*
11:00 AM	12:15 PM	**TH09** To Be Announced	**TH10** Now, The Two Worlds Collided - *Veronika Kolesnikov*	**TH11** What Every Developer Needs to Know About Deep Learning - *Vishwas Lele*	**TH12** Use Async Internals in .NET - *Adam Furmanek*
12:15 PM	2:00 PM	Lunch @ The Mixer - Visit the Microsoft Company Store & Visitor Center			
2:00 PM	3:15 PM	**TH13** What's New in Bootstrap 4 - *Paul Sheriff*	**TH14** How to Interview a Developer - *Billy Hollis*	**TH15** Building Azure Cosmos DB Applications – Part I - *Leonard Lobel*	**TH16** Testability in .NET - *Jason Bock*
3:30 PM	4:45 PM	**TH17** To Be Announced	**TH18** To Be Announced	**TH19** Building Azure Cosmos DB Applications – Part II - *Leonard Lobel*	**TH20** Manual Memory Management in .NET Framework - *Adam Furmanek*
START TIME	**END TIME**	**Post-Conference Workshops: Friday, August 16, 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: ASP.NET Core with Azure DevOps - *Brian Randell*	**F02** Workshop: Web Development in 2019 - *Chris Klug*	**F03** Workshop: SQL Server for Developers: The Grand Expedition - *Andrew Brust and Leonard Lobel*	

Speakers and sessions subject to change

Denotes Microsoft Speaker/Session

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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

Using Survival Analysis for Predictive Maintenance

Zvi Topol

Some years ago, I introduced the basics of survival analysis and described how to implement a non-parametric algorithm called Kaplan-Meier in C# (msdn.com/magazine/dn630650). Now, I'm going to take another look at survival analysis, in particular at two more advanced methodologies that are readily available on two popular machine learning platforms, Spark Machine Learning Library (MLLib) and h2o.ai, which are both supported by Azure HDInsight. I'll use a predictive maintenance use case as the ongoing example.

Predictive Maintenance for the Industrial Internet of Things

The main idea behind the Industrial Internet of Things (IIoT) is to connect computers, devices, sensors, and industrial equipment and applications within an organization and to continually collect data, such as system errors and machine telemetry, from all of these with the aim of analyzing and acting on this data in order to optimize operational efficiencies.

The goal of predictive maintenance is to accurately predict when a machine or any of its components will fail. If you can do this, you can perform maintenance just before such failure is predicted to occur. This is more efficient than not performing any maintenance until a failure occurs, in which case the machine or component will be unavailable until the failure is fixed, if indeed it's reparable. Such unplanned downtime is likely to be very costly.

> The goal of predictive maintenance is to accurately predict when a machine or any of its components will fail.

Predictive maintenance is also more effective than performing preventive maintenance at frequent intervals, which could also be costlier because unnecessary maintenance may be applied.

The example and the data I'll use are an adapted version of the example at bit.ly/2J4WnbN. The example includes 100 manufacturing machines, with no interdependencies among the machines. Each machine is one of four possible models.

This article discusses:

- Predictive maintenance
- Industrial Internet of Things
- Survival analysis
- Cox proportional hazards regression
- The Weibull Accelerated Failure Time Regression model

Technologies discussed:

R, Spark MLLib, h2o.ai, Azure HDInsight

The data for the machines includes a history of failures, maintenance operations and sensor telemetry, as well as information about the model and age (in years) of the machines. This data is available in .csv files downloadable from the resource mentioned earlier. I'll also provide a transformed data file (comp1_df.csv) that's "survival analysis-ready" and will explain how to perform the transformations later on.

When building statistical models, you see covariates of three primary data types: categorical, ordinal and continuous.

Each machine in the original example has four different components, but I'm going to focus only on one component. The component can either be maintained proactively prior to a failure, or maintained after failure to repair it.

Survival Analysis

In my previous article about survival analysis, I introduced important basic concepts that I'll use and extend in this article. I encourage you to read that article to familiarize yourself with these concepts, including the survival and hazard functions, censoring and the non-parametric Kaplan-Meier (KM) estimator.

In this article, I'll show how to extend the concept of the KM estimator to include covariates or variables (also known as features) that can have effects on survival, or, in this case, on machine components' failure. In the example, I'll use machine model, machine age and machine telemetry as covariates and use survival regression models to estimate the effects of such covariates on machine failure.

The notion of estimating the effects of covariates on a target variable, in this case time to failure, hazard rate, or survival probabilities, isn't unique to survival analysis and is the basis for regression models in general.

When building statistical models, you see covariates of three primary data types: categorical, ordinal and continuous. Categorical data types are those types that fall into a few discrete categories. Here, a machine model is a categorical data type—there are four different machine models. Ordinal data types are categorical data types that have some meaningful order. For example, ratings of movies from one to 10, where 10 is the most entertaining and one the least. Finally, continuous data types are those that represent continuous numbers. Those would be the machine telemetry readings here, which are continuous numbers sampled at certain times (in this case, hourly).

After identifying the data types and the methodology to be used, you should encode the various data types into covariates. Typically, for regression models, continuous variables are naturally encoded as continuous covariates, while categorical data types will require some form of encoding. A popular option for such encoding, which I'll use in this article, is where, for categorical data types with N categories, N-1 covariates are created, and a category i is represented by setting its specific covariate to value one and all others to zero. The Nth category is represented by setting all covariates to zero. This is typically a good fit for regression models with an explicitly defined baseline, where all covariates can be equal to zero. This is also the format that the R programming language uses to encode categorical variables or factors.

This encoding for categoricals has a straightforward interpretation for what it means for some or all covariates to be set to zero. However, for continuous data types, setting a certain covariate to zero may not always be meaningful. For example, if a covariate represents machine height or width, setting that covariate to zero would be meaningless, because there are no such machines in reality.

One way around this problem is to use mean centered continuous covariates, where for a given covariate, its mean over the training dataset is subtracted from its value. Then, when you set that transformed covariate to zero, it's equivalent to setting the original covariate to its mean value. This technique is called "mean centering" and I'll use it here for the machine age and telemetry covariates.

It's important to remember, that following this transformation, you should always use mean centered covariates as an input to the model. This is also the case when applying the regression model to a new test dataset.

Once the data values are encoded as covariates, survival regression models then take those covariates and a certain form of survival target variables (which I'll talk about soon) and specify a model that ties the effects of such covariates on survival/time-to-event.

Transformation of the Data to Survival Format and Feature Engineering

In order to work with the survival regression models that I'll describe, your data needs to have at least two fields: the time stamp of the event of interest (here, machine failure) and a Boolean field indicating whether censoring occurred. (Here, censoring describes a situation in which no failure occurred at or before a specified time. In my example, maintenance happening in a preventive manner, rather than as a response to failure, is considered to be censoring.

The survival regression models I'll discuss have different assumptions made to simplify their mathematical derivation.

The survival regression models I'll discuss have different assumptions made to simplify their mathematical derivation. Some of these assumptions may not hold here, but it's still useful to apply survival modeling to this example.

The survival analysis literature is very rich and many advanced survival regression models and techniques have been developed to

address and relax some of these assumptions. You can read more about such models and techniques in the book, "The Statistical Analysis of Failure Time Data" by Kalbfleisch and Prentice (Wiley-Interscience, 2002), at bit.ly/2TACdLR.

> It's frequently desirable to perform additional transformations on the covariates, which is often called "feature engineering."

I'll make the assumption that each maintenance operation performed on a machine component completely resets that component and can therefore be treated independently. It's then possible to use survival regression on two types of intervals (depicted in **Figure 1**):

- The interval between a failure and the preceding maintenance operation (time to event).
- The interval between subsequent maintenance operations (censoring).

Each interval in **Figure 1** starts with a maintenance operation. The first type of interval ends with X, denoting a failure, while the second type ends with O, denoting another maintenance operation prior to a failure (this is essentially a proactive maintenance operation), which in this case means a censored observation.

Therefore, the original data needs to be transformed into this format with the two required fields. The "time_to_event" field represents the time in hours until either failure or the next maintenance occurs. The "event" field is set to one for a failure and to zero for a maintenance operation before failure.

It's frequently desirable to perform additional transformations on the covariates, which is often called "feature engineering." The purpose of this process is to generate covariates with better predictive power. For example, you can create another covariate that will calculate the mean of the pressure in the 10 hours prior to failure. There are many different options for functions and possible time windows to create such covariates, and there are a few tools you can use to help automate this process, such as the open source Python package tsfresh (tsfresh.readthedocs.io/en/latest).

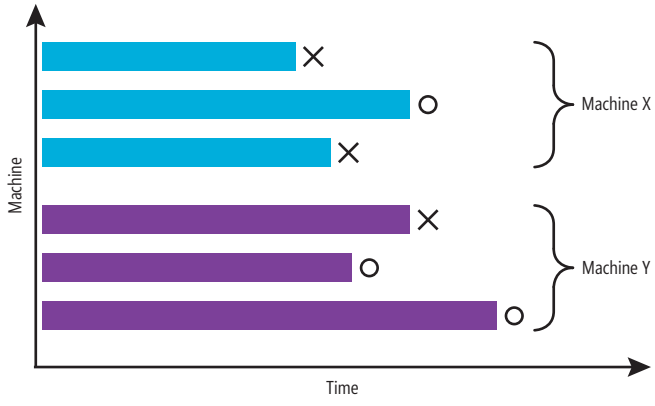


Figure 1 **Survival Representation of Machine Failures**

Now I'm going to discuss the two survival regression models: the Cox proportional hazard model (or Cox PH model) available in h2o.ai and the Weibull Accelerated Failure Time model available in Spark MLLib.

Cox Proportional Hazards Regression

Recall that a hazard function determines the event rate at time t for objects or individuals that are alive at time t. For the predictive maintenance example, it can be described as the probability of failing in the next hour, for a given time t and for all the machines where component 1 failure hasn't occurred since their last maintenance. Higher hazard rates imply higher risk of experiencing failure. The Cox PH regression estimates the effects of covariates on the hazard rate as specified by the following model:

$$h(t) = h_0(t)e^{\beta_1 X_1 + \ldots + \beta_p X_p}$$

Here, h(t) is the hazard function at time t, h0(t) is the baseline hazard at time t, the Xi variables are the different covariates and the corresponding betas are coefficients corresponding to the covariates (more on that a bit later). The baseline hazard is the hazard when all covariates are equal to zero. Note that this is closely related to the intercept in other regression models, such as linear or logistic regression.

According to this model, there's no direct relationship between the covariates and the survival time. This model is called semi-parametric because the hazard rate at time t is a function of both a baseline hazard rate that's estimated from the data and doesn't have a parametric closed form and a multiplicative component that's parameterized.

> The baseline hazard is the hazard when all covariates are equal to zero.

The reason this model is called a proportional hazard model is because it allows you to compare the ratio of two hazard functions. In this case, given an estimated model, the ratio between two different data points is:

$$\frac{h_1(t) = h_0(t)e^{\beta_1 X_1 + \ldots + \beta_p X_p}}{h_2(t) = h_0(t)e^{\beta_1 X_1 + \ldots + \beta_p X_p}} = e^{\beta_1 (X_1 - X'_1) + \ldots + \beta_p (X_p - X'_p)}$$

The baseline hazard rate cancels out and the resulting ratio between the hazards is only a function of coefficients and covariates and again doesn't depend on time. This is closely related to logistic regression where the log of the odds is estimated. Also, the Cox PH regression model doesn't directly specify the survival function, and the information it provides focuses on the ratio or proportion of hazard functions. Therefore, it's primarily used to understand the effects of covariates on survivability, rather than to directly estimate the survival function.

With the Cox PH model specified, the coefficients and the non-parametric baseline hazard can be estimated using various techniques. One popular technique is partial maximum likelihood estimation (also used in h2o.ai).

The following code snippet is an R script that runs an estimation of the Cox PH model using h2o.ai on the mean centered covariates (machine telemetry and age) and the categorical covariate machine model:

```
library(h2o)
localH2O <- h2o.init()
inputFileName<-'comp1_df.csv'
df<-read.csv(inputFileName, header=TRUE, stringsAsFactors=TRUE)
df.hex <- as.h2o(df, key = "df.hex")
model <- h2o.coxph(x = c("age_mean_centered", "model","volt_mean_centered",
  "rotate_mean_centered","pressure_mean_centered", "vibration_mean_centered"),
  event_column = "event", stop_column ="time_to_event" ,training_frame = df.hex)
summary(model)
```

At the time of this writing, the Cox PH model in h2o.ai isn't available to use from Python, so R code is provided. Installation instructions are available at bit.ly/2z2QweL, or, for h2o.ai with Azure HDInsight, at bit.ly/2J7nXp6.

Running the code snippet generates the output shown in **Figure 2**.

The first important thing to note is the estimated coefficients of the covariates. The machine model covariate is encoded as a categorical data type. The baseline for this category is model1, which is represented by setting the three covariates encoding the other three machine models (model.model2, model.model3 and model.model4) to zero. Each covariate gets its own coefficient. Understanding how to interpret the coefficients is important.

If you apply the exponential function to the coefficients for the machine model covariates (exp(coeff) in the output), you see that model.model2 has a value of 0.9352, while model.model4 has a value of 1.3619. This means that machines of model2 have a hazard rate that's 6.5 percent lower than the hazard rate of the baseline machine model (model 1), and that machines of model.model4 have a considerably higher hazard of 36.2 percent compared to machines of model.model1. In other words, machines of model.model4 have the highest risk of failure, while machines of model.model2 have the lowest risk of failure. Therefore, when prioritizing maintenance operations, the model of the machine should be an important factor to take into consideration.

All other covariates are mean centered continuous covariates. The interpretation of the coefficients affiliated with them is that now the hazard ratio is given by the exponential of the covariates around their means. Therefore, by increasing a covariate value by one unit (keeping all other covariates fixed), the hazard ratio increases (or decreases) by the exponential of the coefficient (in a similar way to that of the categorical variable). So, for example, by increasing the voltage by one unit, the risk for failure increases by 3.2 percent.

Another important point to mention here concerns model diagnostics techniques. Such techniques provide a basis to understand whether the model considered (in this case, the Cox PH model) is appropriate. Here, the Rsquare value (a value between zero and one, the higher the better) is relatively low (0.094) and most of the z-scores of the coefficients don't indicate that the coefficients are statistically significant (there isn't enough evidence to support that they're different from zero). Both of these indicators lead to the conclusion that there's room for improvement, for example through feature engineering. There are also other statistical tests that are specific to the Cox PH model that should be conducted. You can consult the survival analysis literature I mentioned earlier for more details.

Figure 2 **Output for the Cox PH Regression**

```
Surv(time_to_event, event) ~ model + volt_mean_centered + rotate_mean_centered +
    pressure_mean_centered + vibration_mean_centered + age_mean_centered

  n= 709, number of events= 192

                            coef      exp(coef) se(coef)    z      Pr(>|z|)
model.model2               -0.066955  0.935237  0.257424 -0.260    0.795
model.model3               -0.021837  0.978400  0.215614 -0.101    0.919
model.model4                0.308878  1.361896  0.227469  1.358    0.174
volt_mean_centered          0.031903  1.032418  0.003990  7.995    1.33e-15 ***
rotate_mean_centered        0.001632  1.001633  0.001362  1.199    0.231
pressure_mean_centered     -0.008164  0.991869  0.005768 -1.415    0.157
vibration_mean_centered     0.018220  1.018387  0.013866  1.314    0.189
age_mean_centered           0.004804  1.004815  0.013293  0.361    0.718
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

                           exp(coef) exp(-coef) lower .95 upper .95
model.model2                0.9352     1.0692    0.5647     1.549
model.model3                0.9784     1.0221    0.6412     1.493
model.model4                1.3619     0.7343    0.8720     2.127
volt_mean_centered          1.0324     0.9686    1.0244     1.041
rotate_mean_centered        1.0016     0.9984    0.9990     1.004
pressure_mean_centered      0.9919     1.0082    0.9807     1.003
vibration_mean_centered     1.0184     0.9819    0.9911     1.046
age_mean_centered           1.0048     0.9952    0.9790     1.031

Rsquare= 0.094   (max possible= 0.941 )
Likelihood ratio test= 70.1  on 8 df,   p=4.69e-12
Wald test            = 70.19  on 8 df,   p=4.514e-12
```

The Accelerated Failure Time Model

The survival regression model in Spark MLLib is the Accelerated Failure Time (AFT) model. This model directly specifies a survival function from a certain theoretical math distribution (Weibull) and has the accelerated failure time property.

The AFT model is defined as follows. Assume an object is characterized by using the (linear) covariates and coefficients:

$$\beta_1 X_1 + ... + \beta_p X_p$$

Also assume that the object has a parametric survival function s(t) and, denoted by s0(t), the survival function of a baseline object (with all covariates set to zero). The AFT model defines the relationship between s(t) and s0(t) as:

$$S(t) = S_0(te^{\beta_1 X_1 + ... + \beta_p X_p})$$

From this definition you can see why the model is called Accelerated Failure Time model. It's because the survival function includes an accelerator factor, which is the exponential function of the linear combinations of the covariates, which multiplies the survival time t.

This type of model is useful when there are certain covariates, such as age (in my dataset, machine age), that may cause monotonic acceleration or deceleration of survival/failure time.

The Weibull distribution is a generalization of the exponential distribution and is a continuous distribution popular in parametric survival models. There are a few variations on how to parameterize

it. Here, I'll use the following two-parameter Weibull distribution version for t>=0:

$$f(t) = \lambda k t^{k-1} e^{-\lambda t^k}$$

(There are also versions with three parameters.) The two parameters of the distribution are the shape that's determined by k and the scale that's determined by lambda. A rough analogy is the way a bell-shaped distribution has a characteristic mean and standard deviation.

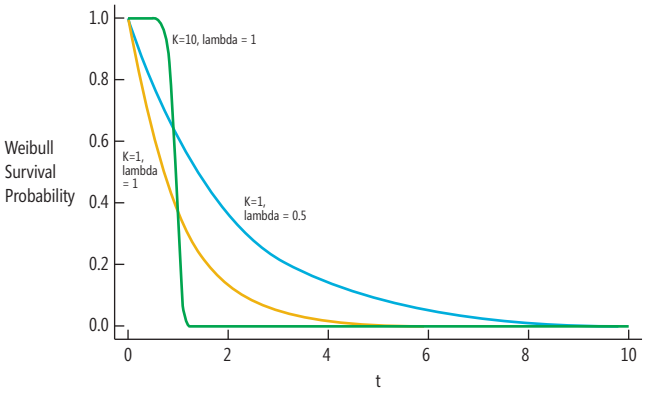


Figure 3 **Weibull Distribution Shape as a Function of Different Values of K and Lambda**

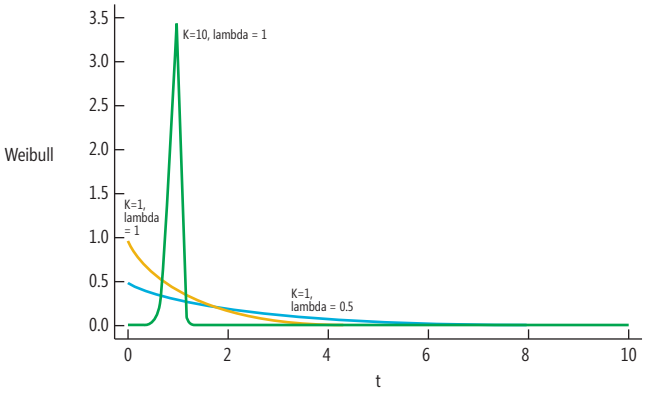


Figure 4 **Weibull Survival Function Shape for Different Values of K and Lambda**

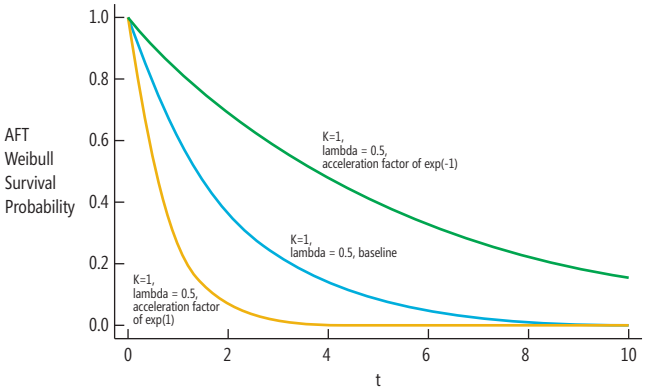


Figure 5 **Accelerated Failure Time for the Weibull Survival Probability Function**

Recall that the relationship between the distribution density function f(t), the hazard function h(t) and the survival function s(t) is given by f(t) = h(t)s(t).

The following are the Weibull hazard and survival functions:

$$h(t) = \lambda k t^{k-1}$$
$$s(t) = e^{-\lambda t^k}$$

Unlike the Cox PH model, both the survival and the hazard functions are fully specified and have parametric representations. Please refer to **Figure 3** and **Figure 4** for visualizations of the Weibull distribution and survival functions for different values of k and lambda.

Figure 5 illustrates the effects that AFT model covariates have on the shape of the Weibull survival function.

> The Weibull distribution is a generalization of the exponential distribution and is a continuous distribution popular in parametric survival models.

Estimation of the coefficients for the AFT Weibull model in Spark MLLib is done using the maximum likelihood estimation algorithm. You can learn more about how it's done at bit.ly/2XSauom, and find the implementation code at bit.ly/2HtJw0v.

Unlike the estimation of the Cox PH model, where only the coefficients of the covariates are reported (along with some diagnostics), the results obtained from estimating the Weibull AFT model report the coefficients of the covariates, as well as parameters specific for the Weibull distribution—an intercept and a scale parameter. I'll show how to convert those to k and lambda in a bit.

The results for the Weibull AFT implementation in Spark MLLib match the results for the Weibull AFT implementation using the survreg function from the popular R library "survival" (more details are available at bit.ly/2XSxkw8).

You can run the following R script for the AFT Weibull model estimation (the code runs on a locally installed Spark MLLi, but you can also use Spark on HDInsight at bit.ly/2u2U5Qf):

```
library(survival)
library(SparkR, lib.loc = c(file.path(Sys.getenv("SPARK_HOME"), "R", "lib")))
sparkR.session(master = "local[*]")
inputFileName<-'comp1_df.csv'
df<-read.csv(inputFileName, header=TRUE, stringsAsFactors=TRUE)
aftMachineDF<-suppressWarnings(createDataFrame(df))
aftMachineModel <- spark.survreg(aftMachineDF,Surv(time_to_event, event) ~ model +
  age_mean_centered +  volt_mean_centered + rotate_mean_centered +
  pressure_mean_centered + vibration_mean_centered)
summary(aftMachineModel)
```

The script generates only the estimated coefficients without additional information. It's possible to get such information by running survreg (because results match):

```
library(survival)
machineModel<-survreg(Surv(time_to_event, event) ~ model + age_mean_centered +
  volt_mean_centered+rotate_mean_centered + pressure_mean_centered +
  vibration_mean_centered, df, dist='weibull')
summary(machineModel)
```

In this case, the R script generates the more elaborate output shown in **Figure 6**.

Before moving on to describe the output, I should mention that the Weibull parameterization in Spark MLLib and in survreg is a bit different than the parameterization I discussed.

A transformation is required and can be done as follows. Denote the parameters reported—intercept by m and scale by s—then k = 1/s, lambda = exp(-m/s) and each coefficient should be multiplied by (-1/s). There's an R package called SurvRegCensCov that can do this conversion automatically, using ConvertWeibull on the model that survreg estimated:

```
$vars
                                Estimate        SE
lambda                      1.260459e-06  8.642772e-07
gamma                       1.662064e+00  8.636644e-02
modelmodel2                 -6.696297e-02 2.569595e-01
modelmodel3                 -4.524990e-02 2.155000e-01
modelmodel4                 2.723541e-01  2.268785e-01
age_mean_centered           1.251958e-03  1.322780e-02
volt_mean_centered          3.279500e-02  3.947495e-03
rotate_mean_centered        1.274045e-03  1.365339e-03
pressure_mean_centered      -8.598142e-03 5.807130e-03
vibration_mean_centered     .365213e-02   1.391255e-02
```

Here, gamma is equal to k from the previous Weibull parameterization. (For more information on SurvRegCensCov, see bit.ly/2CgcSMg.)

Given the estimated parameters, unlike with the Cox PH model, it's now possible to directly obtain the survival function (it's the Weibull AFT survival function) and use it to predict survival probabilities for any covariates. Assuming the first point in the dataset is a new data point, you can run the following:

```
predict(machineModel, newdata=df[1,], type='quantile')
```

This yields the time to event (in hours) for the quantiles 0.1 and 0.9 (the defaults), like so:

```
807.967 5168.231
```

> Estimation of the coefficients for the AFT Weibull model in Spark MLLib is done using the maximum likelihood estimation algorithm.

This means that given the covariates of the first data point (listed here), the probability of failure is 10 percent at or just before 807.967 hours following a maintenance operation, and the probability of failure is 90 percent at or just before 5168.231 hours following the maintenance operation:

model	age	volt_mean_centered	rotate_mean_centered
model3	18	3.322762	51.8113

pressure_mean_centered	vibration_mean_centered	age_mean_centered
10.10773	11.4267	6.488011

You can also use parameter "p" to get the survival time for any quantiles between zero and one; for example, adding the parameter "p=0.5" will give the median failure time, which, for the first data point, is 2509.814 hours after a maintenance operation.

Figure 6 **Output for the Weibull AFT Regression**

```
survreg(formula = Surv(time_to_event, event) ~ model + age_mean_centered
+
    volt_mean_centered + rotate_mean_centered + pressure_mean_centered +
    vibration_mean_centered, data = df, dist = "weibull")
                              Value        Std. Error    z           p
(Intercept)                 8.172991       0.119133    68.6040     0.00e+00
modelmodel2                 0.040289       0.154668    0.2605      7.94e-01
modelmodel3                 0.027225       0.129629    0.2100      8.34e-01
modelmodel4                  -0.163865      0.136382   -1.2015     2.30e-01
age_mean_centered           -0.000753      0.007960    -0.0946     9.25e-01
volt_mean_centered           -0.019731      0.002583   -7.6391     2.19e-14
rotate_mean_centered         -0.000767      0.000821   -0.9334     3.51e-01
pressure_mean_centered       0.005173       0.003496     .4795     1.39e-01
vibration_mean_centered      -0.008214      0.008391   -0.9789     3.28e-01
Log(scale)                   -0.508060      0.051963   -9.7773     1.41e-22

Scale= 0.602

Weibull distribution
Loglik(model)= -1710.3   Loglik(intercept only)= -1747.2
          Chisq= 73.73 on 8 degrees of freedom, p= 8.9e-13
Number of Newton-Raphson Iterations: 8
n= 709
```

As with the Cox PH model estimation, the p column in the output of survreg provides information about the statistical significance of the coefficients estimated, though in this case the figures are better (lower p-values). There's still room for feature engineering here as was described before for the Cox PH model.

It's also important to perform model diagnostics here, as was the case in the Cox PH regression, to make sure that the Weibull AFT model is a good fit for the data, compared, for example, to other parametric models. While I won't describe this process here, you can learn more about it by referring to the "Survival Analysis" book I mentioned earlier.

Wrapping Up

I've presented the use of predictive maintenance for the IIoT as a motivating example for the adoption of two survival regression models that are available in h2o.ai and Spark MLLib. I showed how to model a machine failure predictive maintenance problem in the survival analysis framework by encoding variables as covariates and transforming the time series data to survival format. I also described the two survival models, the differences between them and how to apply them to the data. Finally, I talked briefly about interpretation of the results and model diagnostics. It's important to note that I only scratched the surface of this fascinating and very rich topic, and I encourage you to explore more. A starting point for doing so is by referring to the literature I mentioned in the article. ■

Zvi Topol *has been working as a data scientist in various industry verticals, including marketing analytics, media and entertainment, and Industrial Internet of Things. He has delivered and lead multiple machine learning and analytics projects, including natural language and voice interfaces, cognitive search, video analysis, recommender systems and marketing decision support systems. Topol is currently with MuyVentive LLC, an advanced analytics R&D company, and can be reached at zvi.topol@muyventive.com.*

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

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

Save $300 When You Register by May 17!

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**	**Pre-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** UX Design Fundamentals: What Do Your Users Really See? - *Billy Hollis*	**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** How To Interview a Developer - *Billy Hollis*	**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** State of Mobile Development - *Sam Basu*	**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** Xamarin.Forms Takes You Places! - *Sam Basu*	**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** Add Native to Cross Platform with Xamarin.Essentials - *Veronika Kolesnikova*	**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** Now, The Two Worlds Collided - *Veronika Kolesnikova & Willy Ci*	**TH08** Data(base) DevOps in the Azure Cloud - *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!

DINO ESPOSITO

Routing and Route Templates in Blazor

A remarkable difference between the old days of ASP.NET Web Forms and the modern Web is the presence of a routing component at the gate of the Web server. In Web Forms, the vast majority of Web endpoints were physical file resources, invoked directly through their page path.

With ASP.NET MVC, a routing component kicks in whenever the requested URL can't be mapped to a physical server file. The router takes the requested URL as an instruction to execute that has a client response as its output, whether an HTML view, a JSON payload, a binary stream or other output. The URL can also include optional parameters that help the router determine the specific content to present.

All Web development frameworks today have a routing component, and Blazor is no exception. In this article, I'll explore the implementation and programming interface of the Blazor routing engine.

The Routing Engine

The Blazor routing engine is a component that runs on the client side. It's implementation, however, is made of C# code found in one of the assemblies downloaded in the browser and run through the WebAssembly processor. In Blazor applications, the router is currently configured in the app.cshtml file, like so:

```
<Router AppAssembly=typeof(Program).Assembly />
```

The following code shows the current content of the program.cs file. As you can see, at the moment it doesn't include anything related to the router engine, but something is expected to happen later. Or at least, this is what the comment found in the Visual Studio autogenerated app.cshtml files suggests:

```
public static void Main(string[] args)
{
  BrowserHttpMessageHandler.DefaultCredentials =
    FetchCredentialsOption.Include;
  CreateHostBuilder(args).Build().Run();
}

public static IWebAssemblyHostBuilder CreateHostBuilder(
  string[] args) =>
  BlazorWebAssemblyHost
    .CreateDefaultBuilder()
    .UseBlazorStartup<Startup>();
```

The router class gets the provided assembly name and searches it, along with all referenced assemblies, for Blazor components that match the URL of the ongoing request. Note that this particular aspect of the router class behavior may evolve in the future toward a model where you have to explicitly specify the assemblies that you want the router to consider. That way you don't accidentally end up with endpoints that you didn't expect.

Internally, the router builds up a table of routes and sorts them in a given order. The list of candidate routes results from the list of classes in the explored assemblies that implement the IComponent interface and, more importantly, are decorated with the Route attribute. All collected routes are stored in a dictionary sorted from the most specific to the least specific.

The evaluation algorithm is based on segments discovered in the URL and their position in the string. For example, a literal segment is more specific than a parameter segment, and in turn a parameter segment with more route constraints is considered more specific than others that present fewer constraints. Furthermore, as it happens in ASP.NET MVC, when it comes to resolving a URL, routes in the table are evaluated from most to least specific and the search stops at first match.

> Web development frameworks today have a routing component, whether they're plain server environments such as ASP.NET Core or single-page application (SPA) frameworks.

On the client-side, the router engages in a number of situations, most commonly when the user clicks a link, a submit button on a form, or an item in a dropdown list that triggers a server call. The router binds to an internal location-changed event and handles—from the client side—the whole process of navigating to the newly requested path. Needless to say, the router kicks in also when the location of the application is changed programmatically. Last, but not least, the router logs in the browser history any location change it's responsible for, so the back and forward buttons can work as users expect.

A War of Routers: Blazor vs. Angular

For a long time, the implementation of routing logic was buried in the folds of Web servers or server-side frameworks such as ASP.NET. It was with SPA frameworks—of which Angular is the

Visual Studio LIVE!
EXPERT SOLUTIONS FOR ENTERPRISE DEVELOPERS

Sept 29 - Oct 3, 2019 | Westin Gaslamp Quarter

EXPERIENCE TECH TRAINING IN THE WARM CALIFORNIA SUN

#VSLive

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!

On-Demand Session Recordings Now Available

Get on-demand access for one full year to all keynotes and sessions from Visual Studio Live! San Diego, including everything Tuesday – Thursday at the conference.

Save $400! When You Register by July 26

Your Adventure Starts Here!

vslive.com/sandiego

SUPPORTED BY

Microsoft | Visual Studio | msdn magazine | Visual Studio MAGAZINE

PRODUCED BY

CONVERGE360 An 1105 MEDIA company

king—that router implementations moved to the client. Let's take a moment to run a brief comparison of the features in the consolidated Angular router and the still in-the-works router of Blazor.

The bottom line is that the Blazor router at the moment only provides basic functionality as a client-side router. For instance, it lacks the ability to check authorizations on a route and create links that perform view transitions when the location changes. And unlike the Angular router, it cannot work asynchronously to run the resolve step after obtaining route parameters. Finally, the Blazor router lacks support for conditional redirection to alternate routes—something, again, that the Angular router can do.

It's reasonable to expect that part of this delta will be reduced by the time Blazor ships as version 1.0.

Route Templates

Routing is the process that binds together a URL with a list of known URL patterns. In Blazor, URL patterns, or route templates, are collected in a route table. The table is populated by looking at the components of the Blazor application decorated with the Route attribute. The path to each of those components becomes a supported route template.

At the moment, there's just one way for developers to control the route path of reachable components: the @page directive. In ASP.NET Core, for example, a developer can define routes explicitly by adding them to the table programmatically, letting the system figure out candidates using the default route conventions or using attributes on controller methods. If you use Razor pages in ASP.NET Core applications, then you go through exactly the same experience as a Blazor developer—the @page directive.

In summary, each and every Blazor component must specify its route template through the @page directive to be reachable. A Blazor component is made of a .cshtml file that's compiled into a C# class that implements the IComponent interface. The same dynamically compiled class is also decorated with the Route attribute if the Razor source contains the @page directive.

It's interesting to note that Blazor supports multiple route directives in the same view. In other words, the following code is well supported:

```
@page "/"
@page "/home"
<h1>My Home Page</h1>
```

All routes discovered are put in the same route table container and sorted according to the aforementioned rules. In the previous sample, both route directives are made of literals, so they both go in the top area of the final container and are sorted in order of (relative) appearance.

Routes do support parameters, and a parametric route is recognized at a lower priority in the final table than literal routes, as it's considered less specific. Here's an example of a parametric route:

```
@page "/user/view/{Id}"
```

The URL pattern-matching algorithm triggers for this route when the URL comprises the server name followed by /user/view/. Any content in the URL that trails /user/view/ is associated with the named parameter {Id}.

If you're familiar with ASP.NET MVC (and to a good extent even Web Forms), this model-binding pattern should be old hat. In ASP.NET, route parameters are assigned to the formal parameters of the matched controller method. In Blazor, things are slightly different but comparable.

In Blazor, router parameters are automatically assigned to the properties of the component annotated with the [Parameter] attribute. The matching occurs on the names of parameters and properties. Here's a quick example:

```
@page "/user/view/{Id}"

<h1>Hello @Id!</h1>

@functions {
  [Parameter]
  public string Id { get; set; }

  protected override void OnInit()
  {
    // Some code here
  }
}
```

At the moment, Blazor doesn't support optional parameters, so if {Id} in the sample URL is missing, the entire URL isn't matched. To avoid this, the best current workaround is to have two @page directives, with and without the parameter, as shown in the following code:

```
@page "/user/view/{Id}"
@page "/user/view/"

<h1>Hello @Id!</h1>

@functions {
  [Parameter]
  public string Id { get; set; } = "0";

  protected override void OnInit()
  {
    // Some code here
  }
}
```

At the same time, you might also want to give the bound page parameter a default that's overridden if a value is passed via the URL.

> It should be noted that while the Blazor router can be triggered by links, it can also be triggered programmatically.

Type matching is a common problem with parametric routes and automatic binding to variables. If the segment of the URL contains a literal string, but the bound variable is declared of type int, what happens? In normal conditions, without any precaution, it probably yields an exception, as a literal value is stuffed into an integer container. If you need to make sure that only values of a given type are specified where a parameter is expected, you should opt for route constraints.

A route constraint is nothing new if you're familiar with any flavor of ASP.NET MVC. It consists of adding a type attribute to each URL parameter, as shown here:

```
@page "/user/view/{Id:int}"
```

The name of the parameter is followed by a colon symbol and a literal that refers to a .NET type. Supported literals match one-to-one most of the .NET primitive types: int, bool, double, float, datetime, long and decimal. For routes with constraints, any parameter

values that can't be converted successfully to the specified type invalidate the match and the route isn't recognized.

Smarter Links and Programmatic URL Navigation

In a Blazor application, you're welcome to use anchor tags to create links to external content. However, when anchor tags are used to render a menu or a navigation bar, some additional work may be necessary to adjust CSS styles and to reflect the state of the link.

The built-in Blazor NavLink component can be used wherever an anchor element is required, but especially in a menu. The difference between a canonical HTML anchor element and the NavLink component is in the automatic assignment of the "active" style when the current address matches the link. The "active" CSS class is automatically added to the anchor tag rendered by the NavLink component if the current page URL matches the referenced URL. The implementation of the "active" CSS class remains the responsibility of the page developer. The component also includes a property that controls how matching is done. You can do a strict match or a prefix match.

The Blazor router can also be triggered programmatically. To navigate via code from within a Blazor page, you should first inject the configured dependency for the IUriHelper abstract type. The @inject directive does the job, as shown here:

```
@inject Microsoft.AspNetCore.Blazor.Services.IUriHelper Navigator
```

The injected object can be commanded via the method NavigateTo. The method takes the URL as an argument:

```
Navigator.NavigateTo("/user/view/1");
```

The method is conceptually equivalent to setting the href property of the DOM location object in pure JavaScript. In Blazor, though, the router makes the magic of navigating without leaving the client and without fully reloading the content from the server.

What's Missing

The Blazor framework is an attractive piece of software, but one that remains very much in the works. There are a number of missing routing features—for example, the ability to attach roles or user identities to a route—and authentication and authorization is still incomplete. Any consideration around security-related facilities in routes must wait until those APIs are finalized.

Another important piece of the routing puzzle that's missing: The ability to fully customize the logic of the router that decides about the target URL. This feature would help developers gain control over invalid link requests. While the Blazor router is far from finished, work continues toward a mature, shipping framework. You can view enhancements to the Blazor routing system tracked by the team at bit.ly/2TtYODP. ■

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

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

JAMES MCCAFFREY

Weighted k-NN Classification Using C#

The goal of a machine learning (ML) classification problem is to predict a discrete value. For example, you might want to predict the political leaning (conservative, moderate, liberal) of a person based on their age, annual health care expenses, sex, years of education and so on. There are many different classification techniques, including neural networks, decision trees and logistic regression. In this article I show how to implement the weighted k-nearest neighbors algorithm, using the C# language.

> When using k-NN, it's important to normalize the data so that large values, such as an annual expense of $30,000, don't overwhelm small values, such as an age of 25.

The best way to understand where this article is headed is to take a look at the demo run in **Figure 1** and a graph of the associated data in **Figure 2**. The demo program sets up 33 dummy data items. Each item represents a person's age, annual health care expenses and an arbitrary class to predict (0, 1, 2), which you can think of as political leaning, for concreteness. The data has only two predictor variables so it can be displayed in a graph, but k-NN works with any number of predictors.

The data has been normalized. When using k-NN, it's important to normalize the data so that large values, such as an annual expense of $30,000, don't overwhelm small values, such as an age of 25. The goal of the demo is to predict the class of a new person who has normalized age and expenses of (0.35, 0.38). The demo sets k = 6 and finds the six labeled data items that are closest to the item-to-classify.

After identifying the six closest labeled data items, the demo uses a weighted voting technique to reach a decision. The weighted voting values for classes (0, 1, 2) are (0.3879, 0.4911, 0.1209), so the item at (0.35, 0.38) would be classified as class 1.

This article assumes you have intermediate or better programming skills with C# or a C-family language such as Python or Java, but doesn't assume you know anything about the weighted k-NN algorithm. The complete demo code and the associated data are presented in this article. The source code and the data are also available in the accompanying download. All normal error checking has been removed to keep the main ideas as clear as possible.

Code download available at msdn.com/magazine/0519magcode.

Understanding the Weighted k-NN Algorithm

The weighted k-NN algorithm is one of the simplest of all ML techniques, but there are a few tricky implementation details. The first question most developers have is, "How do you determine the value of k?" The somewhat unsatisfying answer is that the value of k must be determined by trial and error.

When using k-NN, you must compute the distances from the item-to-classify to all the labeled data. There are many distance

```
C:\WeightedKNN\bin\Debug\WeightedKNN.exe

Begin weighted k-NN demo

Normalized age-expenses data looks like:
[id =  0, 0.25, 0.43, class = 0]
 . . .
[id = 32, 0.46, 0.22, class = 2]

Nearest neighbors (k=6) to (0.35, 0.38):

idx =   3  (0.37 0.31) class = 0  distance = 0.0728
idx =  12  (0.29 0.43) class = 1  distance = 0.0781
idx =   0  (0.25 0.43) class = 0  distance = 0.1118
idx =  15  (0.47 0.40) class = 1  distance = 0.1217
idx =  13  (0.35 0.51) class = 1  distance = 0.1300
idx =  30  (0.37 0.24) class = 2  distance = 0.1414

Weights (inverse distance technique):
0.2349  0.2190  0.1530  0.1406  0.1316  0.1209

Predicted class:
[0]  0.3879
[1]  0.4911
[2]  0.1209

End weighted k-NN demo
```

Figure 1 **Weighted k-NN Classification Demo Run**

functions, but using Euclidean distance is simple and effective. The Euclidean distance between the two items is the square root of the sum of the squared differences of coordinates. For example, if a labeled data item is (0.20, 0.60) and the item to classify is (0.35, 0.38) then:

```
dist = sqrt( (0.20 - 0.35)^2 + (0.60 - 0.38)^2 )
     = sqrt(0.0225 + 0.0484)
     = 0.2663
```

After computing all distances and finding the k-nearest distances, you must use a voting algorithm to determine the predicted class. There are many ways to compute a prediction from distances. The demo program uses the inverse weights technique, which is best explained by example. Suppose, as in the demo, the six closest distances are (0.0728, 0.0781, 0.1118, 0.1217, 0.1300, 0.1414) and the associated labeled classes are (0, 1, 0, 1, 1, 2). You compute the inverse of each distance, find the sum of the inverses, then divide each inverse by the sum. For the demo, the inverses are (13.7363, 12.8041, 8.9445, 8.2169, 7.6923, 7.0721). The sum of the inverses is 58.4663. Dividing each inverse distance by the sum gives the weights: (0.2349, 0.2190, 0.1530, 0.1405, 0.1316, 0.1210).

Each weight is a vote for its associated class. In the demo, the first and third closest items have class 0, so the vote for class 0 is the sum of the first and third weights: 0.2349 + 0.1530 = 0.3879. Similarly, the vote for class 1 is 0.2190 + 0.1405 + 0.1316 = 0.4911. And the vote for class 2 is just the sixth weight, 0.1210. The final prediction is the class that has the largest vote.

The Demo Program

The complete demo program, with a few minor edits to save space, is presented in **Figure 3**. To create the program, I launched Visual Studio and created a new console application named WeightedKNN. I used Visual Studio 2017 but the demo has no significant .NET Framework dependencies.

After the template code loaded, in the editor window I removed all unneeded namespace references, leaving just the reference to the top-level System namespace. In the Solution Explorer window, I right-clicked on file Program.cs, renamed it to the more descriptive WeightedKNNProgram.cs, and allowed Visual Studio to automatically rename class Program.

> After computing all distances and finding the k-nearest distances, you must use a voting algorithm to determine the predicted class.

For simplicity, the demo program uses a static code approach rather than an object-oriented programming approach. Most of the work is done in the Analyze method.

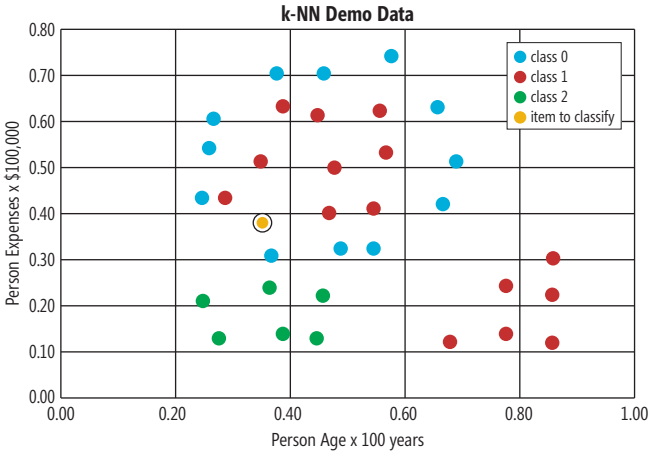


Figure 2 **Weighted k-NN Data**

Loading Data into Memory

The demo program hardcodes the dummy data and the item-to-classify:

```
double[][] data = new double[33][];
data[0] = new double[] { 0, 0.25, 0.43, 0 };
// Etc.
data[32] = new double[] { 32, 0.46, 0.22, 2 };
double[] item = new double[] { 0.35, 0.38 };
```

> I named the data as just "data" rather than something like trainData because data used by k-NN isn't really used to train a general model.

In a non-demo scenario, you'd likely store your data in a text file or SQL table and write a helper method to load the data into an array-of-arrays-style matrix. I named the data as just "data," rather than something like trainData because data used by k-NN isn't really used to train a general model.

The first value in each data item is an arbitrary ID. I used consecutive integers from zero to 32 but, in many cases, you'd have meaningful IDs, such as a person's employee ID number. The k-NN algorithm doesn't need data IDs, so the ID column could've been omitted. The second and third values are the normalized predictor values. The fourth value is the class ID. Class IDs for k-NN don't need to be zero-based integers, but using this scheme is programmatically convenient.

Computing Distances

The first step in the k-NN algorithm is to compute the distance from each labeled data item to the item-to-be classified. Based on my experience and a few research results, the choice of distance function to apply isn't as critical as you might think, and simple Euclidean distance usually works well.

The demo implementation of Euclidean distance is:

```
static double DistFunc(double[] item, double[] dataPoint)
{
  double sum = 0.0;
  for (int i = 0; i < 2; ++i) {
    double diff = item[i] - dataPoint[i+1];
    sum += diff * diff;
  }
  return Math.Sqrt(sum);
}
```

Notice that the for-loop indexing is hardcoded with 2 for the number of predictor values, and the difference between item[i] and

Figure 3 **The Weighted k-NN Demo Program**

```
using System;
namespace WeightedKNN
{
  class WeightedKNNProgram
  {
    static void Main(string[] args)
    {
      Console.WriteLine("Begin weighted k-NN demo ");
      Console.WriteLine("Normalized age-expenses data: ");
      Console.WriteLine("[id =  0, 0.25, 0.43, class = 0]");
      Console.WriteLine(" . . . ");
      Console.WriteLine("[id = 32, 0.46, 0.22, class = 2]");

      double[][] data = new double[33][];
      data[0] = new double[] { 0, 0.25, 0.43, 0 };
      data[1] = new double[] { 1, 0.26, 0.54, 0 };
      data[2] = new double[] { 2, 0.27, 0.60, 0 };
      data[3] = new double[] { 3, 0.37, 0.31, 0 };
      data[4] = new double[] { 4, 0.38, 0.70, 0 };
      data[5] = new double[] { 5, 0.49, 0.32, 0 };
      data[6] = new double[] { 6, 0.46, 0.70, 0 };
      data[7] = new double[] { 7, 0.55, 0.32, 0 };
      data[8] = new double[] { 8, 0.58, 0.74, 0 };
      data[9] = new double[] { 9, 0.67, 0.42, 0 };
      data[10] = new double[] { 10, 0.69, 0.51, 0 };
      data[11] = new double[] { 11, 0.66, 0.63, 0 };

      data[12] = new double[] { 12, 0.29, 0.43, 1 };
      data[13] = new double[] { 13, 0.35, 0.51, 1 };
      data[14] = new double[] { 14, 0.39, 0.63, 1 };
      data[15] = new double[] { 15, 0.47, 0.40, 1 };
      data[16] = new double[] { 16, 0.48, 0.50, 1 };
      data[17] = new double[] { 17, 0.45, 0.61, 1 };
      data[18] = new double[] { 18, 0.55, 0.41, 1 };
      data[19] = new double[] { 19, 0.57, 0.53, 1 };
      data[20] = new double[] { 20, 0.56, 0.62, 1 };
      data[21] = new double[] { 21, 0.68, 0.12, 1 };
      data[22] = new double[] { 22, 0.78, 0.24, 1 };
      data[23] = new double[] { 23, 0.86, 0.30, 1 };
      data[24] = new double[] { 24, 0.86, 0.22, 1 };
      data[25] = new double[] { 25, 0.86, 0.12, 1 };
      data[26] = new double[] { 26, 0.78, 0.14, 1 };

      data[27] = new double[] { 27, 0.28, 0.13, 2 };
      data[28] = new double[] { 28, 0.25, 0.21, 2 };
      data[29] = new double[] { 29, 0.39, 0.14, 2 };
      data[30] = new double[] { 30, 0.37, 0.24, 2 };
      data[31] = new double[] { 31, 0.45, 0.13, 2 };
      data[32] = new double[] { 32, 0.46, 0.22, 2 };

      double[] item = new double[] { 0.35, 0.38 };

      Console.WriteLine("Nearest (k=6) to (0.35, 0.38):");
      Analyze(item, data, 6, 3);  // 3 classes

      Console.WriteLine("End weighted k-NN demo ");
      Console.ReadLine();
    } // Main

    // ----------------------------------------------------

    static void Analyze(double[] item, double[][] data,
      int k, int c)
    {
      // 1. Compute all distances
      int N = data.Length;
      double[] distances = new double[N];
      for (int i = 0; i < N; ++i)
        distances[i] = DistFunc(item, data[i]);

      // 2. Get ordering
      int[] ordering = new int[N];
      for (int i = 0; i < N; ++i)
        ordering[i] = i;

      double[] distancesCopy = new double[N];
      Array.Copy(distances, distancesCopy, distances.Length);
      Array.Sort(distancesCopy, ordering);

      // 3. Show info for k nearest
      double[] kNearestDists = new double[k];
      for (int i = 0; i < k; ++i)
      {
        int idx = ordering[i];
        Show(data[idx]);
        Console.WriteLine("  distance = " +
          distances[idx].ToString("F4"));
        kNearestDists[i] = distances[idx];
      }

      // 4. Vote
      double[] votes = new double[c];  // one per class
      double[] wts = MakeWeights(k, kNearestDists);

      Console.WriteLine("Weights (inverse technique): ");
      for (int i = 0; i < wts.Length; ++i)
        Console.Write(wts[i].ToString("F4") + "  ");

      Console.WriteLine("\n\nPredicted class: ");
      for (int i = 0; i < k; ++i) {
        int idx = ordering[i];
        int predClass = (int)data[idx][3];
        votes[predClass] += wts[i] * 1.0;
      }

      for (int i = 0; i < c; ++i)
        Console.WriteLine("[" + i + "]  " +
        votes[i].ToString("F4"));
    } // Analyze

    static double[] MakeWeights(int k, double[] distances)
    {
      // Inverse technique
      double[] result = new double[k];  // one perneighbor
      double sum = 0.0;
      for (int i = 0; i < k; ++i)
      {
        result[i] = 1.0 / distances[i];
        sum += result[i];
      }
      for (int i = 0; i < k; ++i)
        result[i] /= sum;
      return result;
    }

    static double DistFunc(double[] item, double[] dataPoint)
    {
      double sum = 0.0;
      for (int i = 0; i < 2; ++i) {
        double diff = item[i] - dataPoint[i+1];
        sum += diff * diff;
      }
      return Math.Sqrt(sum);
    }

    static void Show(double[] v)
    {
      Console.Write("idx = " + v[0].ToString().PadLeft(3) +
        "  (" + v[1].ToString("F2") + " " +
        v[2].ToString("F2") + ") class = " + v[3]);
    }

  } // Program
} // ns
```

dataPoint[i+1] is offset to account for the data ID value at position [0]. The point here is that there's usually a tradeoff between coding an ML algorithm for a specific problem and coding the algorithm with a view toward general-purpose use. Because k-NN is so simple and easy to implement, I usually prefer the code-for-specific-problem approach.

At first thought, the requirement of computing a distance function seems to impose the major restriction on k-NN that all predictor variables must be strictly numeric. And until recently this idea was true, for the most part. However, one of the reasons for the increased interest in k-NN is that it's possible for k-NN to deal with mixed numeric and non-numeric data by using neural encoding of the predictor variables.

Briefly, the idea is to encode predictor variables using a neural autoencoder. An autoencoder can deal with non-numeric data using one-hot or 1-of-(N-1) encoding. An autoencoder predicts its output values to match its input values. The central hidden layer of an autoencoder is a completely numeric representation of the source numeric and non-numeric data.

Sorting or Ordering the Distances

After all distances have been computed, the k-NN algorithm must find the k-nearest (smallest) distances. One approach is to augment the entire labeled dataset with each distance value, then explicitly sort the augmented data. An alternative approach, used by the demo, is to use a neat overload of the .NET Array.Sort method to sort just the distances and the associated data indices in parallel. The key code is:

```
int[] ordering = new int[N];
for (int i = 0; i < N; ++i)
  ordering[i] = i;
double[] distancesCopy = new double[N];
Array.Copy(distances, distancesCopy, distances.Length);
Array.Sort(distancesCopy, ordering);
```

> One of the reasons for the increased interest in k-NN is that it's possible for k-NN to deal with mixed numeric and non-numeric data by using neural encoding of the predictor variables.

The array named ordering initially holds 0, 1, 2 . . 32. A copy of the 33 distances to the item-to-classify is created because you don't want to lose the distances in their original order. The statement Array.Sort(distancesCopy, ordering) sorts the values in the distancesCopy array from smallest to largest using the Quicksort algorithm, and simultaneously rearranges the index values in the ordering array. The result is that the first value in array ordering is the index to the data of the item with the smallest distance, the second value in ordering holds the index to the second-closest distance and so on. Nice!

Determining k-NN Weights and Voting

The most primitive form of using the k-nearest distances to predict a class is to use a simple majority rule approach. For example, if k = 4 and c = 3, and two of the closest distances are associated with class 2, and one closest distance is associated with class 0, and one closest distance is associated with class 1, then a majority rule approach predicts class 2. Note that weighted k-NN using uniform weights, each with value 1/k, is equivalent to the majority rule approach.

> Compared to many other classification algorithms, the results of weighted k-NN are relatively easy to interpret.

The majority rule approach has two significant problems: First, ties are possible. Second, all distances are equally weighted. The most common weighting scheme for weighted k-NN is to apply the inverse weights approach used by the demo program. But there are many other techniques. For example, the reverse distances technique sums all distances, divides all distances by the sum, then reverses the order.

Another approach is to use the rank of the k-nearest distances (1, 2, . . 6) instead of the distances themselves. For k = 6, the rank order centroid weights would be calculated as (1 + 1/2 + 1/3 + 1/4 + 1/5 + 1/6) / 6 = 0.4083, (1/2 + 1/3 + 1/4 + 1/5 + 1/6) / 6 = 0.2417, and so on.

For the demo program, the inverse, uniform, reverse and centroids weights are:

```
Inverse   Uniform   Reverse   Centroids
-------------------------------------
0.2349    0.1667    0.2156    0.4083
0.2190    0.1667    0.1982    0.2417
0.1530    0.1667    0.1856    0.1583
0.1405    0.1667    0.1705    0.1028
0.1316    0.1667    0.1191    0.0611
0.1210    0.1667    0.1110    0.0278
```

Wrapping Up

The weighted k-NN classification algorithm has received increased attention recently for two reasons. First, by using neural autoencoding, k-NN can deal with mixed numeric and non-numeric predictor values. Second, compared to many other classification algorithms, the results of weighted k-NN are relatively easy to interpret. Interpretability has become increasingly important due to legal requirements of ML techniques from regulations such as the European Union's GDPR. ■

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*

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
- Office 365 for the IT Pro
- PowerShell and DevOps
- Security
- Soft Skills for IT Pros

SAVE $400
WHEN YOU REGISTER BY JUNE 7
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	TechMentor's Downtown Seattle Adventure				
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

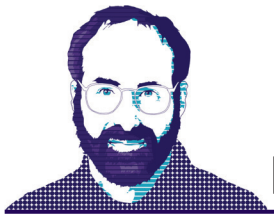
CONNECT WITH TECHMENTOR

Twitter
@TechMentorEvent

Facebook
Search "TechMentor"

LinkedIn
Search "TechMentor"

TechMentorEvents.com/MicrosoftHQ



DAVID PLATT

Calc or Stats?

My daughter Annabelle is now in the spring of her senior year of high school. Most seniors slack off, but that's not her way. She's taking both AP Calculus and AP Statistics courses, and is now trying to finish them off.

She absolutely loved calculus, saying, "Derivatives are the meaning of life." But statistics bore the living crap out of her. (I can't repeat her exact language in this column, and I can't imagine where she learned those words.) And yet, I can't help wondering.

I've trained as an engineer, worked my entire career in technical fields, taken calculus through multivariable. But never, not once, have I fired it in anger against a target that I needed to kill in order to eat. The concepts of differentiating and integrating, yes; the actual operations, never. When was the last time you took the cross product of a vector field, or even a common garden variety derivative? On the other hand, I see percentages and probabilities many times per day, and have never studied them in detail. I'm probably misusing them and making mistakes. As are you.

> I say to Zak, you're a doctor, and you want other doctors to take their medicine. How about a spoonful of sugar to help it go down?

The results of this misprioritization aren't benign. Here's an example from Zak Kohane, whom you may remember from my earlier articles (December 2017, msdn.com/magazine/mt814422). He and his colleagues published a research letter in "JAMA Internal Medicine," entitled "Medicine's Uncomfortable Relationship with Math: Calculating Positive Predictive Value" (bit.ly/2eil1A6). They asked 61 doctors the following question: "If a test to detect a disease whose prevalence is 1/1,000 has a false positive rate of 5 percent, what is the chance that a person found to have a positive result actually has the disease, assuming you know nothing about the person's symptoms or signs?"

Figure it out yourself. I'll bet you get it wrong. So did three-quarters of the doctors they asked. Would you say 95 percent? 90 percent? I'll stop writing to give you time to work it out.

The correct answer is 2 percent. Remember, the disease is rare, 1 in 1,000 people. A false positive test is much more common, 5 percent, or 50 out of 1,000 tests. The odds are 50-to-1 that this positive value indicates a testing error. Not that this comforts the patient, or the doctor.

Zak and his co-authors wrote: "With wider availability of medical technology and diagnostic testing, sound clinical management will increasingly depend on statistical skills …. Specifically, we favor revising premedical education standards to incorporate training in statistics in favor of calculus, which is seldom used in clinical practice."

Why don't technical people study statistics? Partly because it's not on the advanced math track in high school, so students who take it suffer the perception of being misfits who couldn't hack the calculus track. But watching Annabelle's virtual high school video, I'm convinced that the main reason is that current presentations of it are painfully boring. I tried one of their lessons, and couldn't stay awake.

Why are things this way? I've written for years in this space of the enormous educational opportunities, once basic classes are taught online (see msdn.com/magazine/hh708761 and onward). Why isn't the best statistics teacher in the universe making this video? Or even one in the top half?

I could do far better myself. If there's one thing I've learned in the teaching and presentation business, it's this: Know your audience, because they're not you. What do high schoolers think about? From mentoring the robotics class, I know: sex, money and misbehavior. I could design the most killer stats class ever for high school and early college. I'd tap their natural proclivities by using Charles Wheelan's "Naked Statistics" (Norton, 2014) as the textbook. His chapter on live beer tasting commercials during the Super Bowl would fit right in. I'd work in supplemental readings from Darrell Huff's 1954 classic, "How to Lie with Statistics."

Instead of showing one dry equation, then another, and then another, I'd start with a real-life situation, explain what we want to know about it and why, then develop the math from there. For example, "Here's how the lottery and casinos are taking you for suckers. And here's how you turn the tables on them."

I say to Zak, you're a doctor, and you want other doctors to take their medicine. How about a spoonful of sugar to help it go down? Who's got the money, and the nerve, to pay me to do this? ■

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!

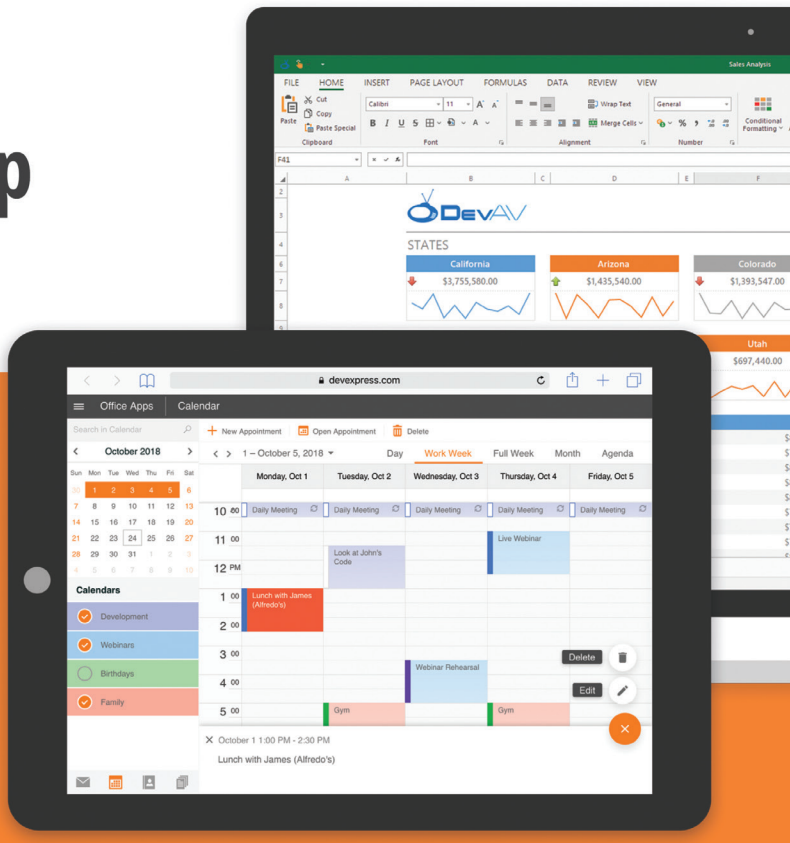
melissa

www.Melissa.com | 1-800-MELISSA

Free Trials, Free Data Quality Audit & Professional Services.

DevExpress®

Your Next Great App Starts Here



UI Components
Reporting & Analytics

WIN ASP MVC WPF UWP JS

What's in this Issue?

/n software	/n software Red Carpet Subscription	6
Aspose	Aspose.Total for .NET	8
	Aspose.PDF for .NET	10
CData Software	CData ADO.NET Subscription	12
DevExpress	DevExpress Universal	14
	DevExtreme	16
GrapeCity	ActiveReports	18
	ComponentOne Studio Enterprise	20
	SpreadJS	22
Infragistics	Infragistics Ultimate	24
	Indigo.Design	26
InstallAware	InstallAware Studio Admin	28
	InstallAware Studio	30
Syncfusion	Syncfusion Essential Studio Enterprise	32
	Syncfusion Essential Studio for JavaScript	34
Telerik	Kendo UI	36
	Telerik DevCraft Complete	38
Text Control	TX Text Control .NET Server for ASP.NET	40
	TX Text Control ReportingCloud	42
Actipro Software	Actipro WPF Studio	44
LEADTOOLS	LEADTOOLS Document Imaging Suite SDK	46
Nevron	Nevron Vision for .NET	48
Syncro Soft	Oxygen XML Editor Enterprise	50

Official Supplier

As official and authorized distributors, we supply you with legitimate licenses directly from 300+ software publishers.

24/5 Customer Service

Need help to find the right software license, upgrade or renewal? Call, Email or Live Chat with our experts.

Trusted for Over 20 Years

Over 850,000 licenses delivered to Developers, SysAdmins, Corporations, Governments & Resellers, worldwide.

Open 24 hours a day, Monday to Friday

International Customer Service Centers in the US, UK, Ireland and Japan.

United States	888 850 9911
US Gov't Sales	888 850 9966
United Kingdom	0800 581111
Japan	0120 343 550
Argentina	0800 666 0185
Australia	1 800 0 15292
Austria	0800 281 750
Belgium	0800 7 5603
Brazil	0800 891 6478
Bulgaria	00800 115 4445
Canada	(888) 850 9911
Chile	1230 020 6857
China (North)	10800 481 1661
China (South)	10800 813 1594
Colombia	01800 710 2043
Croatia	0800 222550
Czech Republic	800 143 313
Denmark	80 88 17 20
Ecuador	1 800 010 562

Finland	0800 1 18002
France	0800 90 92 62
Germany	0800 186 07 06
Greece	00800 44121891
Hong Kong	800 908 581
Hungary	06800 16674
India	000 800 44 01 455
Indonesia	001 803 00811 351
Ireland	1 800 535 661
Israel	180 924 2003
Italy	800 790046
Japan	0120 343 550
Korea, Rep. of	00798 14 800 6332
Malaysia	1 800 81 7261
Mexico	01 800 681 1559
Netherlands	0800 022 8832
New Zealand	800 449181
Norway	800 1 3685
Panama	00 1 800 203 1587

Peru	0800 53288
Philippines	1800 1816 0315
Poland	00800 442 1092
Portugal	800 844 125
Russia	810 800 2251 1044
Singapore	800 810 2136
Slovenia	0800 80297
South Africa	0800 99 1088
Spain	900 93 8926
Sweden	020 794 989
Switzerland	0800 83 5305
Taiwan ROC	00 801 814313
Thailand	001 800 81 4 5257
Turkey	00 800 4491 3617
United Kingdom	0800 581111
United States	888 850 9911
Uruguay	000 401 902 38
Venezuela	0 800 100 9103
Viet Nam	120 81 863

USA

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

Tel: +1 770 250 6100
Fax: +1 770 250 6199

Europe (UK)

ComponentSource
The White Building
33 King's Road
Reading
Berkshire
RG1 3AR
United Kingdom

Tel: +44 118 958 1111
Fax: +44 118 958 9999

Europe (Ireland)

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

Tel: +353 1 685 6704
Fax: +353 1 685 6725

Asia-Pacific

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

Tel: +81 3 3237 0281
Fax: +81 3 3237 0282

/n software Red Carpet Subscription

Top 50 Product | Top 25 Publisher | Enterprise Partner

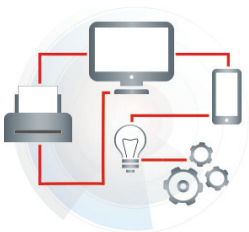
A comprehensive suite of Internet components. Any protocol, any platform, any IDE.

Publisher: /n software | Category: Communication & Messaging | ★★★★★

Components for every major protocol from FTP to IMAP to SNMP, SSL and SSH security, S/MIME encryption, Digital Certificates, Credit Card Processing, Bluetooth Low Energy, Internet of Things (IoT), and e-business (EDI) transactions.

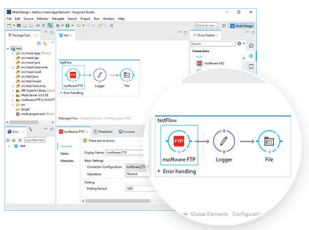
Bluetooth Low Energy Components

A suite of Bluetooth Low Energy (BLE) components that provide straightforward access to BLE operations. The IP*Works! BLE components provide simple service discovery and access to BLE-enabled devices.



IP*Works! Internet of Things (IoT)

Easily implement Internet of Things (IoT) communications protocols in your applications on any platform with this lightweight set of components. It helps you easily communicate between networked devices.



Mule Connectors

The /n software Connectors for MuleSoft are pure Java Connectors and Modules that seamlessly extend MuleSoft's functionality - complete with built-in, enterprise-ready web connectivity, secure messaging, and file transfer.

/n software

Prices from $ 1,535.04

www.componentsource.com/nsoftware-red-carpet-subscription

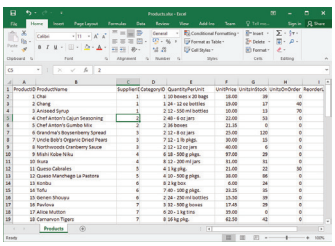
Aspose.Total for .NET

#5 Product | #3 Publisher | Enterprise Partner

Open, create, convert, print & save files from within your own applications.

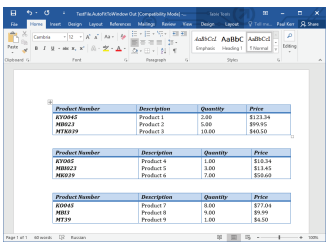
Publisher: Aspose | Category: Document & Text Processing | ★★★★★

Aspose.Total for .NET is a compilation of every .NET component offered by Aspose. It allows you to create a wide range of applications that work with file formats from Excel, Word, PowerPoint, Project, OneNote, Outlook, Visio and PDF.



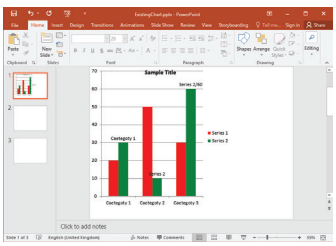
Aspose.Cells for .NET

Create, read, edit, convert, protect or print Excel spreadsheets including more than 10 file formats without installing Microsoft Excel. Includes a combination of APIs and GUI controls.



Aspose.Words for .NET

An advanced class library that lets you perform a wide range of document processing tasks directly within your applications. Supports DOC, DOCX, OOXML, RTF, HTML, OpenDocument, PDF, XPS, EPUB and many other formats.



Aspose.Slides for .NET

Enables any .NET application to read, write, modify and convert PowerPoint documents, without the need for Microsoft PowerPoint. It can render presentation slides to PDF, XPS, HTML, Images and PDF Notes.

ASPOSE File Format APIs

Prices from $ 2,939.02

www.componentsource.com/aspose-total-net

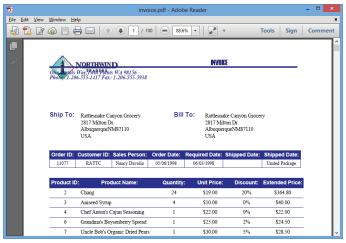
Aspose.PDF for .NET

Top 25 Product | #3 Publisher | Enterprise Partner

Create and manipulate PDF documents without using Adobe Acrobat.

Publisher: Aspose | Category: PDF | ★★★★★

Aspose.PDF for .NET is a set of PDF APIs that enables your .NET applications to read, write and manipulate existing PDF documents. It also allows you to create forms and manage form fields embedded in a PDF document.



Convert PDFs

Convert between PDF and other popular document formats including HTML, PCL, DOCX, XLS, SVG, JPG, PNG, TIFF and XPS. You can also transform XML and XSL-FO documents into PDFs.

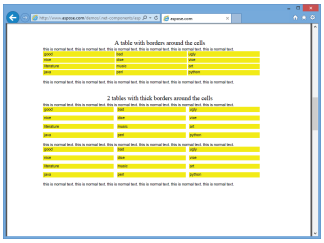
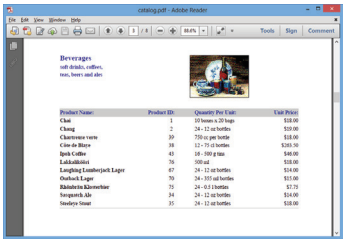


Table Creation

Access, create and modify tables, rows and cells. You can also add headings and Table of Contents during PDF creation and programmatically format document elements.



Document Manipulation

Concatenate or merge two or more PDF documents, append new pages to an existing PDF file and extract or insert pages. Document formatting is supported using an advanced document object model.

ASPOSE
File Format APIs

Prices from $ 979.02
www.componentsource.com/aspose-pdf-net

CData ADO.NET Subscription

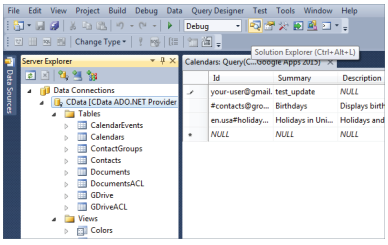
Comprehensive access to data from ADO.NET.

Publisher: CData Software | Category: Data Access

Real-time data integration from 125+ SaaS, NoSQL, and Big Data sources. Includes a single SQL interface to data that insulates users from the challenges and complexities of integrating with individual APIs, SDKs, and services.

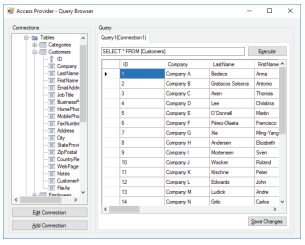
Connect .NET Apps to Data

Comprehensive access to application, database, and web API data through easy-to-use tools. Whether you are building desktop, web, or mobile apps you can use the drivers for fast and secure access to data.



Visual Studio Integration

The ADO.NET Data Providers can be used to access and explore data directly from the Visual Studio Server Explorer. These constructs return live data that you can work with directly from within Visual Studio.



Access Data from any .NET App

Use objects to connect to data just as you would any traditional database, through the Visual Studio Server Explorer, in code, and in data controls like DataGridView, GridView and DataSet.

Prices from $ 979.02

www.componentsource.com/cdata-ado-net-subscription

Know SQL?

Connect to 125+ data sources (NoSQL, Big Data, SaaS, etc.) through standard SQL queries.

ODBC JDBC ADO.NET SSIS CLOUD BIZTALK EXCEL MOBILE

SQL is the language of data, and our state-of-the-art Drivers let you leverage the full power of SQL to connect with hundreds of applications, databases, and APIs. Through standards-compliant Driver technologies like ODBC, JDBC, ADO.NET, & OData, you can read, write, and update live data, from any data source, using standard SQL queries.

Learn more: www.componentsource.com/cdata

cdata

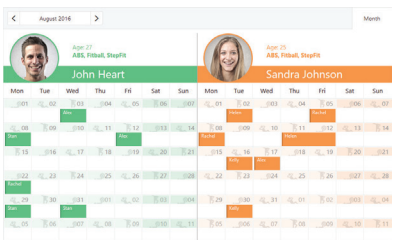
DevExpress Universal

#2 Product | #1 Publisher | Enterprise Partner

All DevExpress Visual Studio products in one integrated subscription.

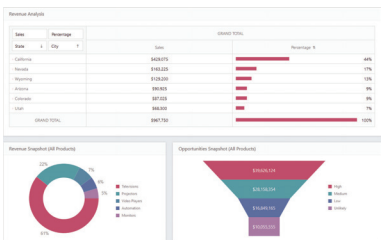
Publisher: DevExpress | Category: Presentation Layer | ★★★★½

Includes charts, data grids, spreadsheets, calendars, schedulers, diagrams, navigation, text editors, PDF, maps and gauges. Supports ASP.NET Web Forms, MVC & Core, WinForms, WPF, UWP, JavaScript, jQuery, Angular and React.



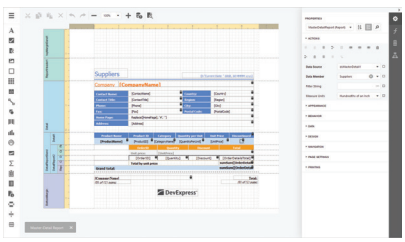
ASP.NET Core Controls

DevExpress ASP.NET Core client-side controls are complemented with server-side web API data processing extensions. As a result, regardless of dataset size, users can quickly shape and analyze mission-critical information.



Data Visualization Widgets

JavaScript data visualization widgets allow you to transform data to its most concise and readable visual representation. All chart, gauge and range selector widgets can be composed into beautiful, informative dashboards that effectively convey intelligence at a single glance.



DevExpress Reports

A feature-complete reporting platform which ships with an easy-to-use Visual Studio report designer and a rich set of report controls, including pivot tables and charts so you can construct reports of unmatched elegance and informational clarity.

DevExpress™

Prices from $ 2,111.99

www.componentsource.com/devexpress-universal

DevExpress®

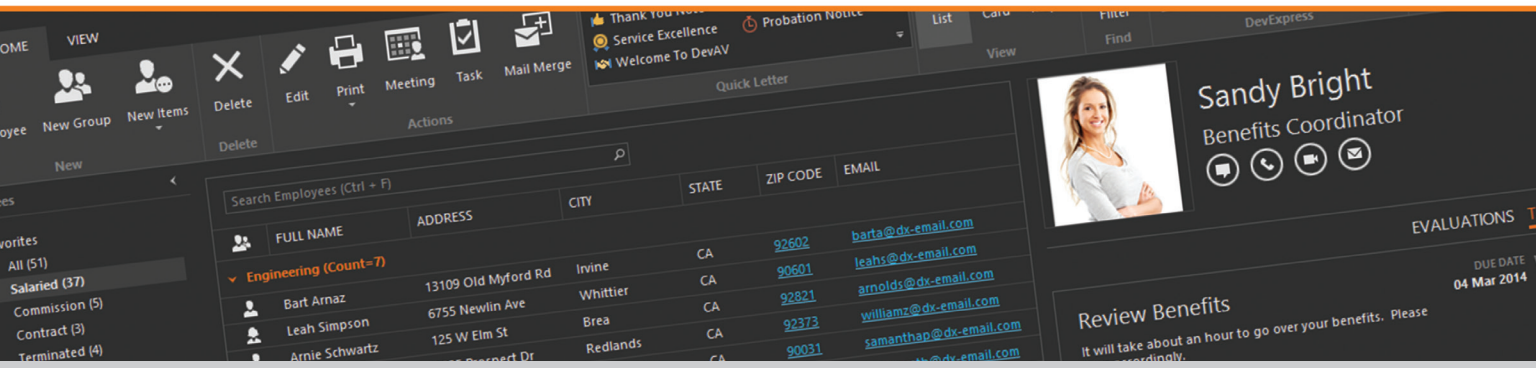
High-Performance Office-Inspired User Interface Controls

DATA GRID · SPREADSHEET · WORD PROCESSING · RIBBON · SCHEDULER · DIAGRAM · AND MORE

Your Next Great Office-Inspired App Starts Here

Learn how you can create high-impact and fully customizable user experiences with our fully integrated suite of UI components and data visualization libraries for Visual Studio.

componentsource.com/devexpress-universal



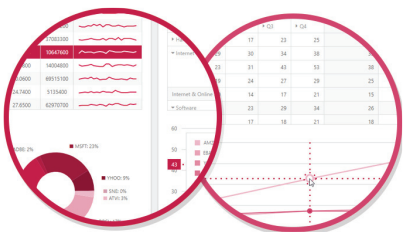
DevExtreme

Top 50 Product | #1 Publisher | Enterprise Partner

HTML5 JavaScript component suite for responsive web development.

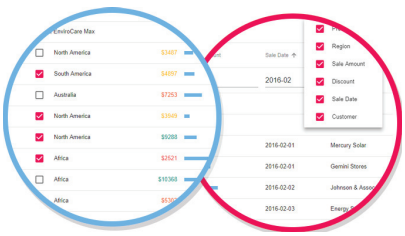
Publisher: DevExpress | Category: Presentation Layer | ★★★★★

Create responsive web apps for touch devices and traditional desktops. DevExtreme includes data grids, interactive charts, data editors, navigation and multi-purpose widgets that are designed to look great and provide powerful functionality in any browser.



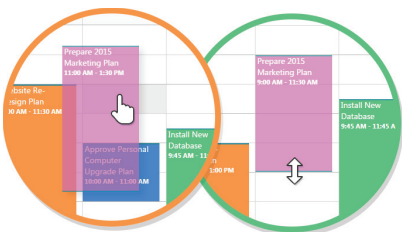
Charting and Pivot Grid

Transform data into its most concise and readable visual representation. DevExtreme Analytics widgets ship with a fully optimized client side data engine so you can build solutions that meet and exceeds end-user performance expectations.



Grids, Trees and Editors

The blazing-fast DevExtreme Data Grid is a feature-rich data shaping and editing client-side widget which allows your end users to easily manage information and display it on-screen as business requirements dictate.



Scheduling and Navigation

The DevExtreme Scheduling widget ships with four pre-built views (Day, Week, Work Week and Month). Your users can quickly switch between each view via its integrated calendar view selector.

DevExpress™

Prices from $ 479.99

www.componentsource.com/devextreme

DevExpress

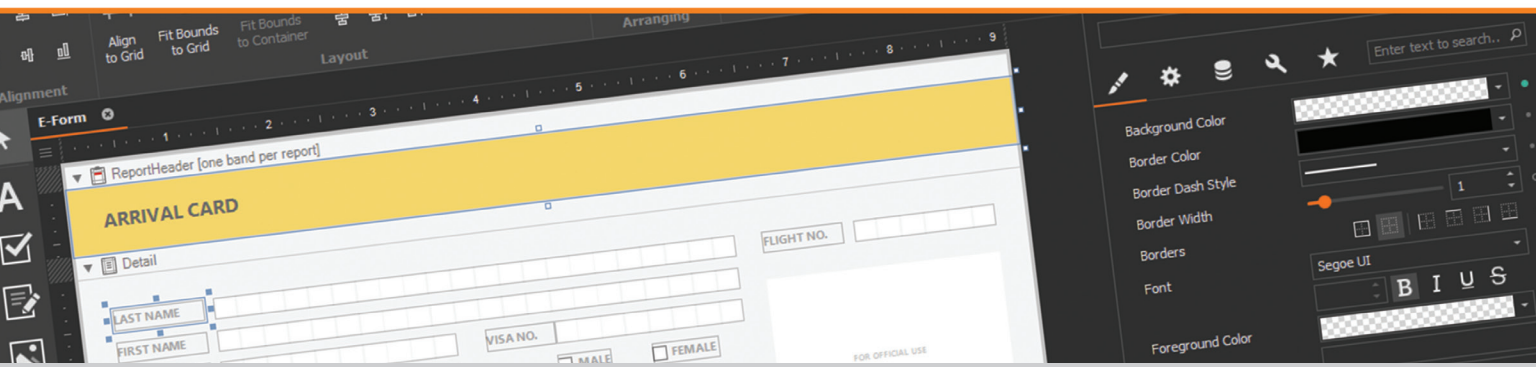
Enterprise-Ready Reporting
for WinForms, WPF, ASP.NET, MVC and .NET Core

WIN WPF ASP MVC .NET CORE

Your Next Great App Starts Here

DevExpress Reporting ships with a fully integrated and easy-to-use End-User Report Designer for all supported platforms.

componentsource.com/devexpress-universal



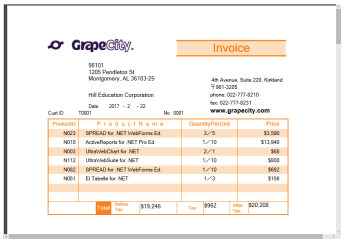
ActiveReports

A flexible and extensible platform for every kind of reporting.

#3 Product | #2 Publisher | Enterprise Partner

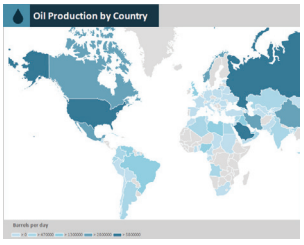
Publisher: GrapeCity | Category: Reporting | ★★★★½

Design, customize, publish, and view data in your business applications. Create everything from simple invoices to complex statistical data visualizations, and you can even enable your end users to design reports for themselves.



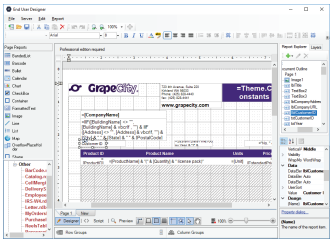
Dynamic Reports

You can control report behavior by using events or by designing dynamic reports entirely in code. You can also use property expressions to change the run-time behavior of controls in RDL and Page reports.



.NET Reporting Components

ActiveReports includes a complete set of reporting controls and code libraries. Use built-in controls such as barcodes, tablix, charts, data bars, sparklines, and maps to visualize data.



Reporting for End Users

End users always ask for tweaks in standard reports. Give them the power to make the changes themselves with a fully customizable set of designer controls that includes a drag and drop report designer.

GrapeCity

Prices from $ 1,575.02

www.componentsource.com/activereports-developer

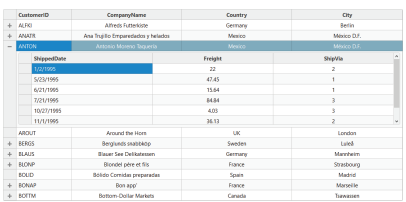
ComponentOne Studio Enterprise

Top 10 Product | #2 Publisher | Enterprise Partner

Award-winning .NET controls for business applications.

Publisher: GrapeCity | Category: Presentation Layer | ★★★★½

A complete collection of award-winning performant, extensible .NET UI controls for mobile, web, and desktop. Includes controls for WinForms, WPF, UWP, ASP.NET MVC, Xamarin, JavaScript and server-side APIs for processing images, Excel files and PDFs.

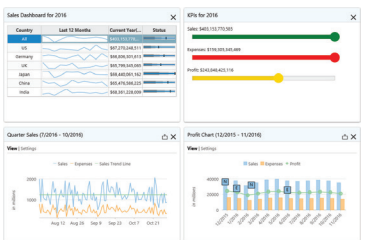


Industry-Leading FlexGrid

Get power, flexibility, and speed in a single lightweight grid control. Sort, group, filter, merge cells, freeze columns, and import or export to Excel with FlexGrid, an industry-leading flexible data grid.

Engaging User Interactions

The UI and navigational control set offers intuitive displays of information with several forms of interaction. Edit, sort, delete, build menus and toolbars, and easily apply themes when crafting tailored UIs.



Powerful Data Visualization

Whether you need 60+ charts, Gantt views, pivot tables, gauges, maps, or sparklines, ComponentOne Studio's data visualization controls handle large data sets and impress your users with professional design.

GrapeCity

Prices from $ 1,472.58

www.componentsource.com/componentone-studio

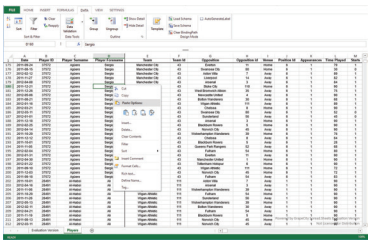
SpreadJS

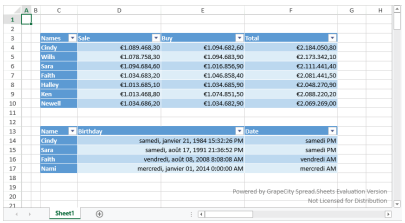
Top 50 Product | #2 Publisher | Enterprise Partner

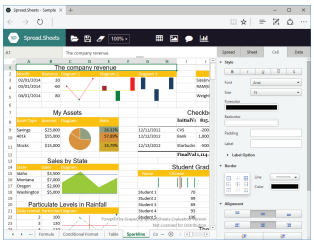
High-speed Excel-like JavaScript spreadsheet components for enterprise apps.

Publisher: GrapeCity | Category: Spreadsheet

Surpass the limits of a traditional spreadsheet with these Excel-like JavaScript spreadsheet components. Create spreadsheets, grids, dashboards, and forms with the comprehensive API.







Excel-like JavaScript Spreadsheets

SpreadJS helps you deliver user-friendly Excel spreadsheet experiences. Users can instantly design spreadsheets with the SpreadJS Designers for desktop and web. They can import, edit, and export Excel spreadsheets, use the built-in templates or create their own.

Built-in Globalization

Select from available cultures, or create your own custom culture and adjust date masks, languages, number separators, currencies, and more. You can modify the culture string and create a custom culture using the CultureInfo class.

Data Display and Visualization

With full Excel chart support, interactive shapes, sparklines, and a free-form grid layout, you can build extensive data visualization dashboards that include performance indicators, data bars, color scales, and conditional formatting.

GrapeCity

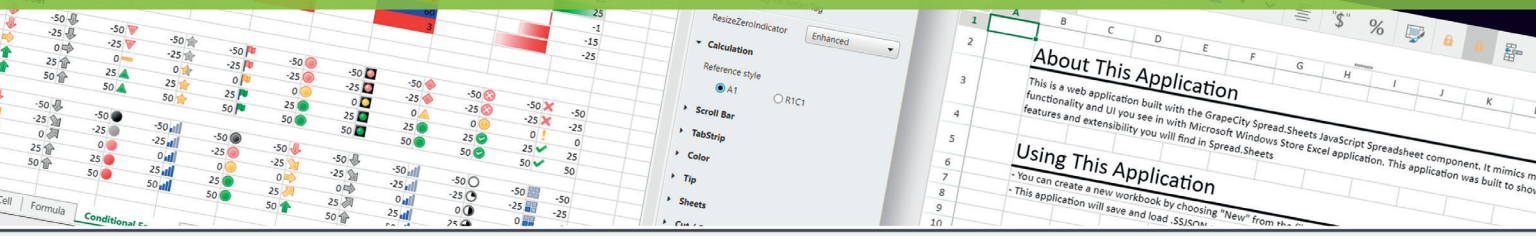
Prices from $ 1,476.52

www.componentsource.com/spreadjs

GrapeCity | Developer Solutions

SpreadJS

Deliver high-performance, high-speed JavaScript spreadsheet apps



Design **spreadsheets, charts,** and **input forms** code-free with desktop **designer apps**

Generate **Excel documents** with the **fastest** spreadsheet API available

Pure JavaScript supports **Angular** and **TypeScript**, with full support for **React** and **Vue**

Build apps for **risk analysis, financial models, budgeting,** and **business performance**

Work with **462 Excel functions,** the most available

Learn more at **componentsource.com/spreadjs**

Infragistics Ultimate

Top 50 Product | Top 10 Publisher | Enterprise Partner

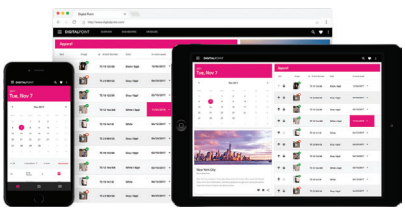
UX design & enterprise app development for web, desktop and mobile.

Publisher: Infragistics | Category: Presentation Layer

Includes 100+ beautifully styled, high-performance grids, charts, & other UI controls, plus visual configuration tooling, rapid prototyping, and usability testing. Includes components for Angular, JavaScript, ASP.NET, Windows Forms, WPF & Xamarin.

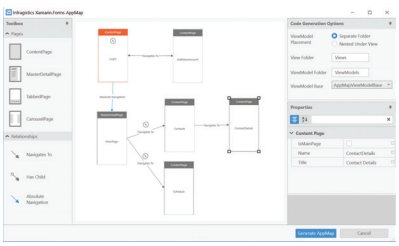
Ultimate UI for WPF

Create engaging modern and Office-inspired apps for desktop and touch devices with lightning-fast grids and charts, dynamic data visualizations, and versatile controls.



Angular Data Grid

The Angular Data Grid can handle unlimited rows and columns of data, while providing access to custom templates and real-time data updates. Featuring an intuitive API for easy theming and branding, you can quickly bind to data with minimal code.



Ultimate UI for Xamarin

Easily lay out your entire Xamarin.Forms application using a familiar Visio-style diagram interface. Design your app in minutes by dragging and dropping pages, child relationships, and navigation instructions onto the design surface.

INFRAGISTICS

Prices from $ 1,955.10

www.componentsource.com/infragistics-ultimate

Indigo.Design

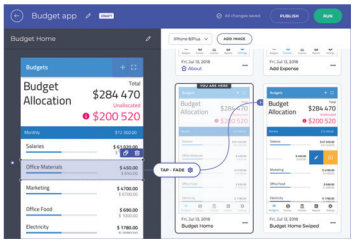
Top 10 Publisher

Enterprise Partner

Unparalleled productivity for teams. Pixel-perfect Angular code from Sketch designs.

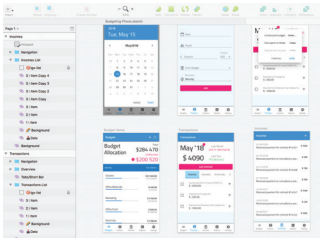
Publisher: Infragistics | Category: Software Architecture & Design

Designer meets developer with Indigo.Design, a unified platform for visual design, UX prototyping, code generation, and app development.



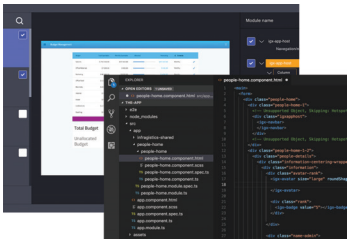
Share, Collaborate & Test

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



Design to Match Your Brand

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



Get Runnable Code with 1-Click

Generate high-quality HTML, CSS, and Angular code from your design with the click of a button. 50+ UI components in Sketch map to the Ignite UI toolset.

INFRAGISTICS

Prices from $ 975.10

www.componentsource.com/indigo-design

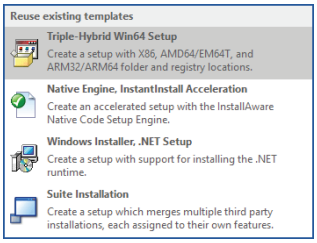
InstallAware Studio Admin

Top 100 Product | Top 50 Publisher | Enterprise Partner

An installer with full-stack ARM64 Platform (Snapdragon 835/850 Cellular PC) support.

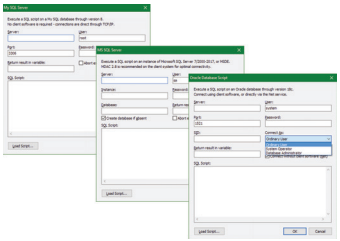
Publisher: InstallAware | Category: Development & Deployment

Design, build, deploy, and even repackage native installers on always-connected, desktop Windows 10 ARM64 platforms.



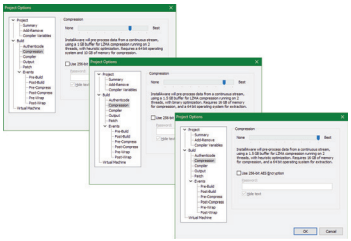
Repackage ARM64 Applications

InstallAware builds setups based on changes detected in native file system and registry locations on ARM64 platforms.



Universal Client-Free Database Support

Run SQL scripts on MS SQL Server versions through 2017, MySQL versions through 8.x, and Oracle versions through 18c.



Advanced 32-bit Compatible Compression

Compress .NET 4.7.2 13% smaller, .NET 4.5.2 17% smaller; SQL Server 2017 11% smaller and SQL Server 2016 with SP1 38% smaller.

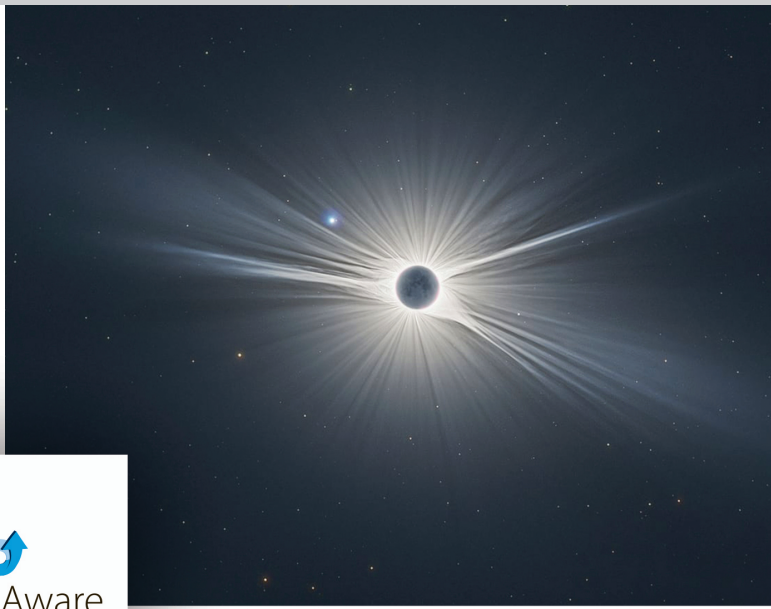
InstallAware

Prices from $ 3,919.02

www.componentsource.com/installaware-studio-admin

NEW!
InstallAware X9

InstallAware X9 is the first and only installer that you may use to build triple-hybrid installers with X64 (AMD64/EM64T), ARM64, and X86 payloads all delivered from a single setup file!



InstallAware

ARM64 (QUALCOMM SNAPDRAGON 835/50) CELLULAR PC
First and only with full-stack support to build, deploy, and repackage apps.

NEXT GENERATION MSIX PACKAGING FORMAT
Build MSIX packages from the Visual Studio Toolbar, InstallAware IDE, or the command line!

UNIVERSAL CLIENT-FREE DATABASE REFRESH
MS SQL versions through 2017, MySQL versions through 8.x, Oracle versions through 18c.

FULL WINDOWS 10 OCTOBER 2018 ECO-SYSTEM SUPPORT
Including .NET Framework 4.7.2, Visual C++ Runtime 15.9, and SQL Server Express 2017.

ADVANCED 32-BIT COMPATIBLE COMPRESSION
Reduce the already-compressed sizes of .NET 4.52 by 17%, MS SQL Express 2016 with SP1 by 38%!

InstallAware is the most flexible platform for traditional and agile development teams creating Windows and Azure software installers, as well as MSIX Universal Windows Platform, Microsoft App-V Virtualization, and agentless/royalty-free InstallAware Virtualization packages.
Bridge all of your Win32, Win64, and .NET apps through Microsoft's Desktop Bridge to the Microsoft Windows Store using InstallAware's new MSIX Builder!

Visual Studio Partner

www.componentsource.com/installaware

InstallAware Studio

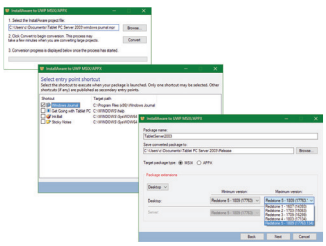
Top 50 Publisher

Enterprise Partner

An installer with next generation MSIX Packaging Format support.

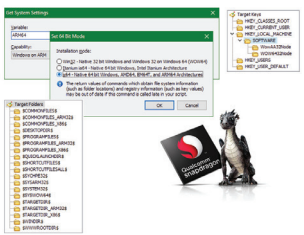
Publisher: InstallAware | Category: Development & Deployment

Build Win32, Win64, and .NET apps as MSIX packages, crossing the Microsoft Desktop Bridge to the Microsoft Windows Store.



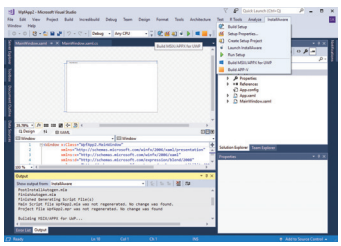
Build MSIX/APPX packages from Visual Studio

A single click on the InstallAware toolbar for Visual Studio builds your active solution/project as an MSIX package.



Build Triple-Hybrid AMD64/ ARM64/X86 Installers

InstallAware lets you build a single setup targeting three platforms from one file: AMD64/ EM64T, ARM64, and X86.



Windows 10 October 2018 Refresh

Full eco-system support for the latest Windows 10 version, including .NET 4.7.2, Visual C++ 15.9, and SQL Server 2017 runtimes.

InstallAware

Prices from $ 1,959.02

www.componentsource.com/installaware-studio

About InstallAware

InstallAware Software, founded in 2003, is the leading Cloud Infrastructure Company with its laser sharp focus on bullet-proof enterprise software deployment. InstallAware has been recognized by multiple awards from Microsoft, SDTimes "Leader of the Software Development Industry", Visual Studio Magazine Reader's Choice, ComponentSource, WindowsITPro, among others.

InstallAware X9 is available in a complimentary edition for all Visual Studio users and paid editions. For more information, visit https://www.componentsource.com/installaware

- A SINGLE SETUP FOR X64, ARM64, AND X86 PAYLOADS
- MSIX BUILDER FOR NEXT-GENERATION PACKAGING
- UNIVERSAL, CLIENT-FREE DATABASE SUPPORT
- WINDOWS 10 OCTOBER 2018 PLATFORM REFRESH
- ADVANCED 32-BIT COMPATIBLE COMPRESSION

www.componentsource.com/installaware

Syncfusion Essential Studio Enterprise

Top 100 Product | Top 25 Publisher | Enterprise Partner

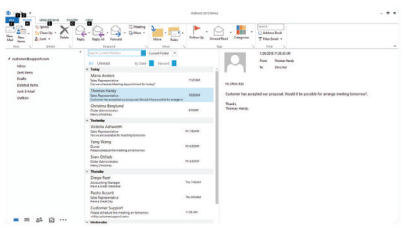
High-performance toolkits for your development projects.

Publisher: Syncfusion | Category: Presentation Layer | ★★★★★

Includes more than 1,000 components and frameworks for WinForms, WPF, ASP.NET (Web Forms, MVC, Core), UWP, Xamarin, JavaScript, Angular, Vue, and React.

ASP.NET Web Forms

A comprehensive suite of ASP.NET Web Forms components for enterprise web development. It includes several complex widgets such as data grid, chart, gantt, diagram, spreadsheet, schedule, pivot grid and much more.



Outlook-style Scheduling

Add a customizable Outlook-style scheduler to your applications. Users can plan, create and manage appointments/events and also organize various other activities like meetings, deadlines, tasks, etc.

Mobile Development

Essential Studio Enterprise comes with over 130 controls, including data grids, charts, and Excel, Word and PDF manipulation controls, for developing Xamarin.Forms, Xamarin.iOS and Xamarin.Android applications.

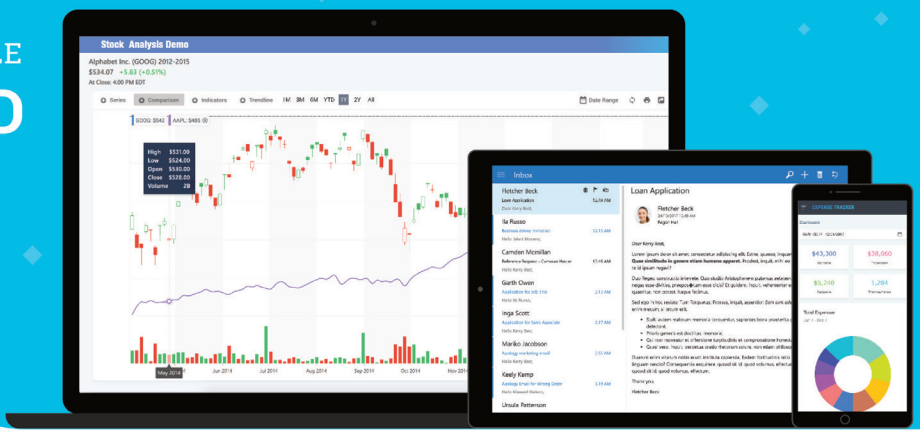
Syncfusion

Prices from $ 2,245.50

www.componentsource.com/syncfusion-essential-studio-enterprise

MORE THAN **1000** CUSTOMIZABLE

CONTROLS AND FRAMEWORKS



ESSENTIAL STUDIO ENTERPRISE EDITION

WEB

- ASP.NET MVC
- ASP.NET Web Forms
- ASP.NET Core
- JavaScript
- Angular
- React
- Vue

MOBILE

- JavaScript
- Xamarin
- Universal Windows Platform

DESKTOP

- Windows Forms
- WPF
- Universal Windows Platform

FILE FORMATS

- Excel
- PDF
- Word
- PowerPoint

DATA SCIENCE

- Predictive Analytics

CHOOSE WITH CONFIDENCE

5

120+
FIVE STAR REVIEWS

Syncfusion

NOW AVAILABLE / Multi-license bundles!

FOR MORE INFORMATION VISIT

componentsource.com/syncfusion

Syncfusion Essential Studio for JavaScript

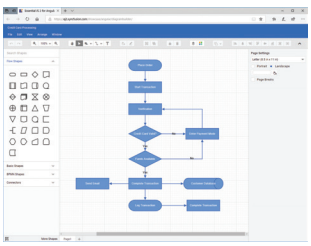
Top 25 Publisher

Enterprise Partner

A complete JavaScript UI controls library.

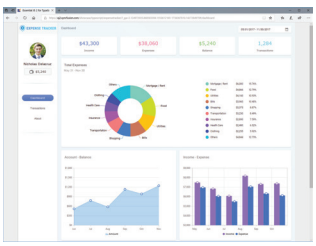
Publisher: Syncfusion | Category: Presentation Layer | ★★★★☆

Includes high-performance, lightweight, modular, and responsive UI components. All the controls are touch friendly and render adaptively based on the device, providing optimal user experience on phones, tablets, and desktops.



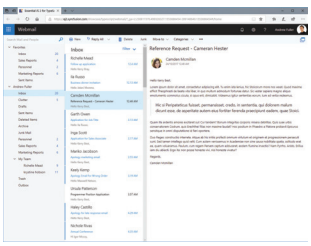
Create & Edit Interactive Diagrams

Create flowcharts, organizational charts, mind maps, and BPMN charts either through code or a visual interface. The library provides all standard flowchart shapes as ready-made objects, making it is easy to add them to a diagram surface in a single call.



Interactive Charts & Graphs

Visualize and analyze data with 30+ charts and graphs ranging from line to financial charts. Users can interact and explore with features such as zoom and pan, selection, tooltip and trackball.



Rich Text with Markdown Editor

Edit WYSIWYG HTML and markdown content with a rich set of tools for modern web and mobile applications. Developers can easily incorporate the controls into a blog editor, HTML email composer, or discussion forum-like applications.

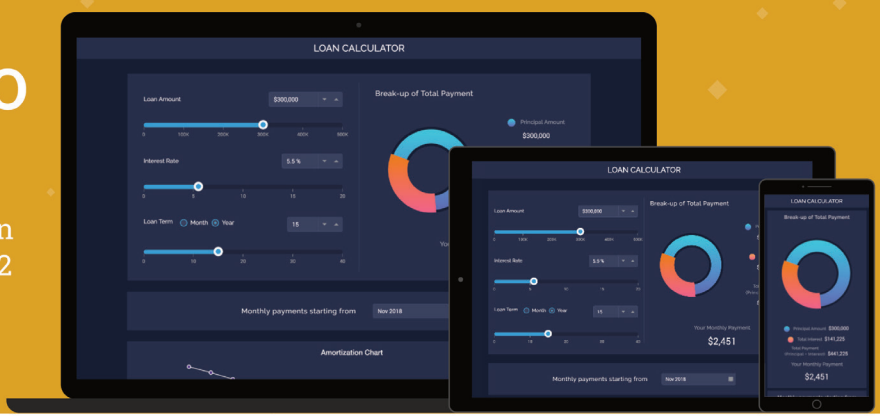
Syncfusion

Prices from $ 975.10

www.componentsource.com/essential-studio-for-javascript

SYNCFUSION ESSENTIAL STUDIO FOR JAVASCRIPT

Featuring Syncfusion's next-generation pure JavaScript package, Essential JS 2



COMPREHENSIVE. MODERN. FREE OF EXTERNAL DEPENDENCIES.

- Lightweight
- Modular
- Responsive
- Customizable with built-in themes
- Built for performance
- Usable in multiple languages

INTEGRATES WITH TOP FRAMEWORKS

- Angular
- React
- Vue

Syncfusion

FOR MORE INFORMATION VISIT

www.componentsource.com/essential-studio-for-javascript

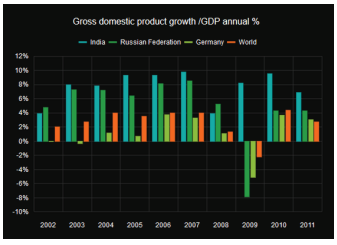
Kendo UI

Top 100 Product | Top 10 Publisher | Enterprise Partner

JavaScript, HTML5 UI widgets for responsive web and data visualization.

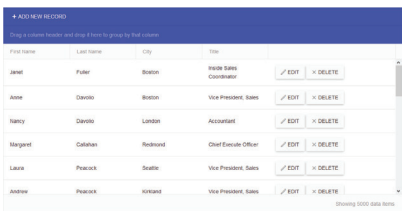
Publisher: Telerik | Category: Presentation Layer | ★★★★☆

A complete JavaScript UI component library that includes 70+ customizable UI components and allows you to quickly build high quality, high performance responsive web apps using your framework of choice - jQuery, Angular, React or Vue.



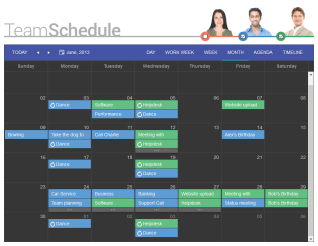
Eye-catching JS Charts

The Kendo UI Chart includes all commonly used chart types. Charts are completely rendered through JavaScript, so are server platform agnostic, and boost the performance of your app.



HTML5/JS Grid Widget

Kendo UI's responsive and adaptive HTML5 grid widget offers over 100 features from filtering and sorting data, to advanced features like pagination and hierarchical data grouping.



Powerful HTML5 Scheduler

The Kendo UI Scheduler allows you to easily schedule, display and edit appointments. You can display day, week, month, timeline and agenda views.

Progress® Telerik®

Prices from $ 881.02

www.componentsource.com/kendo-ui

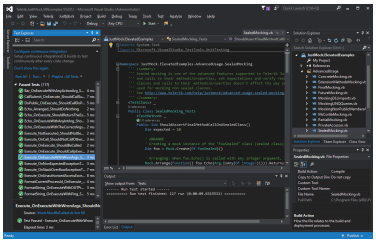
Telerik DevCraft Complete

Top 25 Product | Top 10 Publisher | Enterprise Partner

Quickly build modern, high-performance apps for any platform.

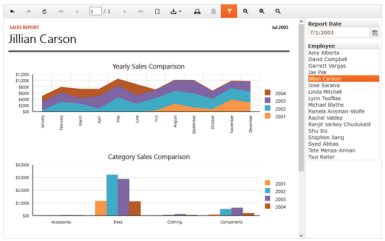
Publisher: Telerik | Category: Presentation Layer | ★★★★☆

Build .NET and JavaScript apps with a sleek, fast and consistent UI across all web, desktop and mobile platforms. Includes mocking and reporting functionality, and Priority Support with unlimited support tickets and 24 hour response time.



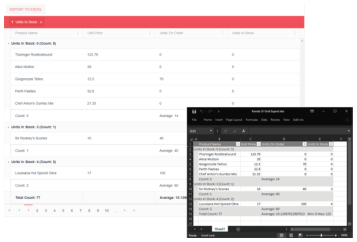
Flexible Mocking Framework

Telerik JustMock makes it easier for you to create mock objects and set expectations independently of external dependencies like databases, web service calls, or proprietary code.



.NET Reporting Made Easy

Telerik Reporting lets you create, view and export rich, beautiful, interactive and reusable reports – everything a lightweight and feature-complete reporting solution should do.



Grid Export to Excel

The Kendo UI grid provides server-agnostic client Excel export functionality. You have the option to customize the rows/columns and cells of the exported file by intercepting the excelExport event.

Progress® Telerik®

Prices from $ 1,469.02

www.componentsource.com/telerik-devcraft-complete

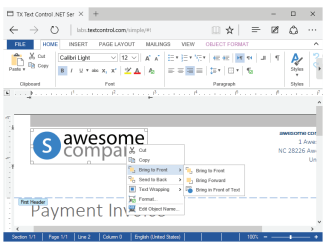
TX Text Control .NET Server for ASP.NET

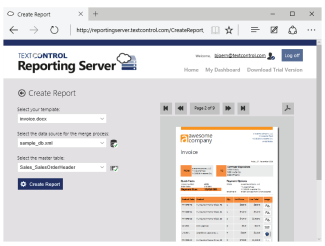
Top 50 Publisher | Enterprise Partner

Powerful word processing & reporting for your Web Forms & MVC web applications.

Publisher: Text Control | Category: Word Processing

Specifically designed to run in server side applications, TX Text Control .NET Server for ASP.NET is ideal for batch processing or printing large volumes of documents.





Cross-Browser Document Editing

Includes a true WYSIWYG, HTML5-based web editor and reporting template designer. Give your users an MS Word compatible editor to create powerful reporting templates anywhere - in any browser on any device.

Automate MS Word Documents

Replacing MS Office Automation in applications is a typical use case for document APIs such as TX Text Control. Automate, edit and create documents using UI and non-UI components.

Document Conversion

Convert and modify different document types. Supports a wide range of word processing formats (RTF, DOC, DOCX, HTML, XML, PDF) and image file formats (GIF, PNG, JPG, BMP, WMF, EMF, TIF).

TEXT CONTROL

Prices from $ 2,938.04

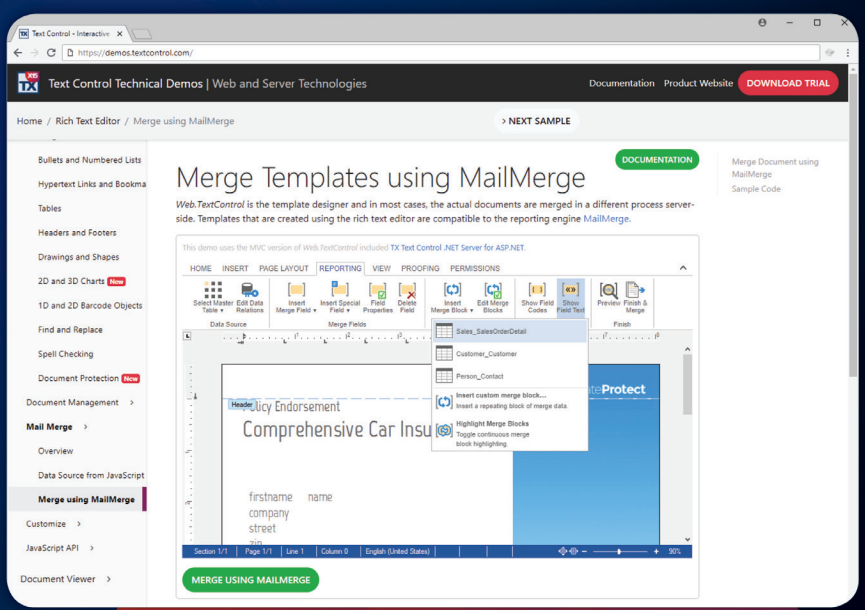
www.componentsource.com/tx-text-control-net-server

TEXT CONTROL

Visual Studio Partner

Integrate Documents and Reporting in Any Platform

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



Learn more:

componentsource.com/textcontrol

WE ARE CHANGING THE WAY YOU LOOK AT REPORTING

Text Control ReportingCloud

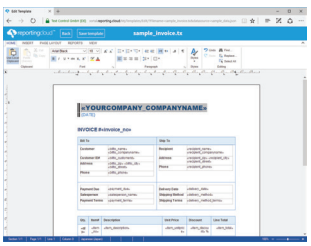
Top 50 Publisher

Enterprise Partner

Web API reporting platform to create MS Word compatible reports in the Cloud.

Publisher: Text Control | Category: Cloud Services

Use the Text Control ReportingCloud web API to merge MS Word compatible templates with JSON data from all clients such as .NET, Javascript, PHP, Node.JS, jQuery, Python, Android, Java, iOS and many more.

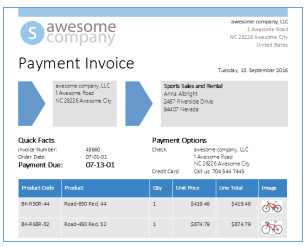


Design your Template

Create customized invoices, contracts and quotes directly using the WYSIWYG online template designer.

Merge with Data

Using simple HTTP requests, merge fields and repeating blocks in templates can be merged with hierarchical JSON data from within your application.



Retrieve Created Document

The merged templates are returned as Adobe PDF, MS Word and HTML format and you can use them immediately in your application.

TEXT CONTROL

Prices from $ 341.04

www.componentsource.com/text-control-reportingcloud

reportingcloud

TEXT CONTROL

Create Documents in the Cloud from Any Platform

RESTful Web API powered reporting platform to create MS Word compatible reports.

Visual Studio Partner

php

JS

WE ARE CHANGING THE WAY YOU LOOK AT REPORTING

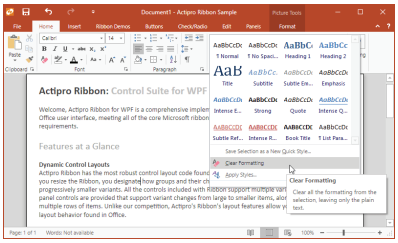
Actipro WPF Studio

Top 100 Product | Top 50 Publisher | Corporate Partner

Everything you need to add rich functionality to your WPF apps.

Publisher: Actipro Software | Category: Presentation Layer | ★★★★★

Over 100 WPF controls and components including barcodes, charts, datagrid, docking & MDI, editors, gauges, micro charts, navigation, property grid, ribbon, syntax editor, themes, tree controls, views, wizards, and a shared library of controls.

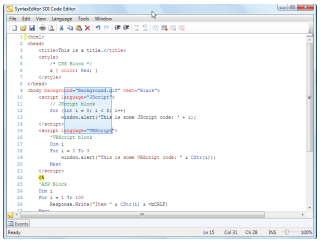


Ribbon Control

Use the Ribbon control to reproduce an Office-like user interface with split buttons, galleries and mini-toolbars. Various layout controls govern where items are placed within a ribbon as it decreases in width.

Easily Generate Rich Charts

Actipro Charts supports many chart types from basic line & bar charts to complex stacked area charts. It includes useful features such as multiple series, labels, legends, stacking, and customizable palettes.



SyntaxEditor for WPF

Add advanced code editing functionality to your WPF apps. Over 20 sample languages are included to get you started (such as C#, HTML, JavaScript, and more), and optional premium add-ons are also available.

actipro

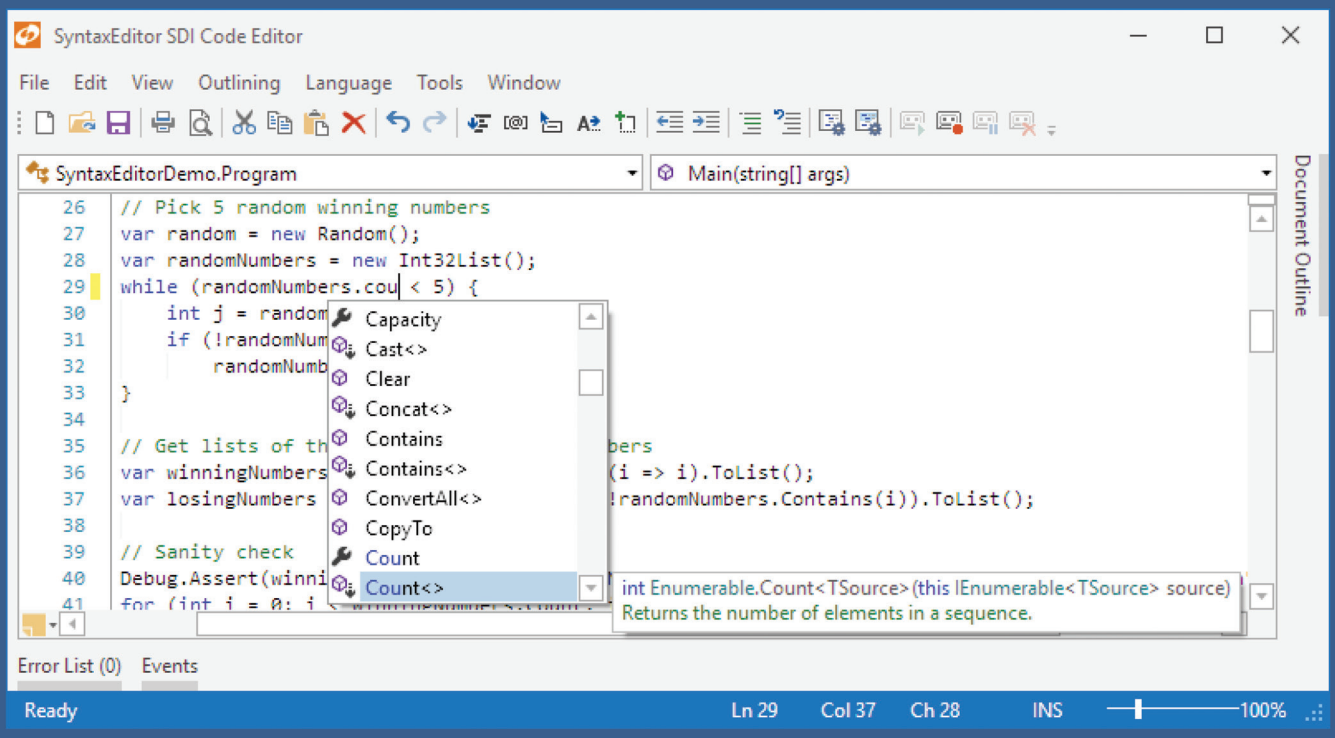
Prices from $ 636.02

www.componentsource.com/actipro-wpf-studio

Actipro SyntaxEditor

for WPF, Universal Windows, Silverlight, WinForms

The ultimate syntax-highlighting code editor control



Discover the possibilities...

componentsource.com/actipro

actipro

LEADTOOLS Document Imaging Suite SDK

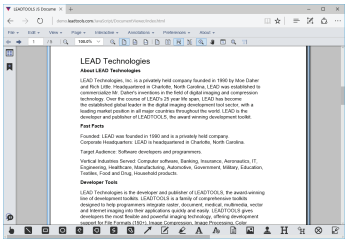
Top 25 Publisher

Corporate Partner

Develop powerful document imaging applications.

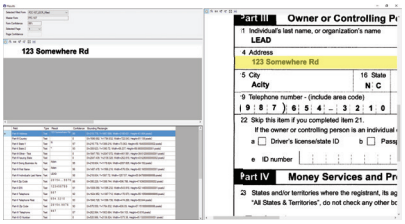
Publisher: LEADTOOLS | Category: Image Editing & Processing | ★★★★★

Build end-to-end document imaging solutions that include OCR, OMR, barcode, forms recognition and processing, PDF, conversion, annotation, HTML5 Zero-footprint viewing, print capture and image viewing functionality.



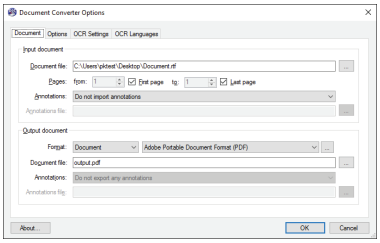
Document Viewer

With only a few lines of code you can create robust, fully featured applications with rich document viewing features, including text search, annotation, inertial-scrolling, paging, and vector display.



Forms Recognition

In addition to basic forms recognition with static field locations, LEADTOOLS is able to detect and process unstructured and semi-structured documents such as invoices, driver's licenses, and passports.



Document Converter

Create powerful, automated document conversion, archival and delivery systems using .NET (C# & VB) for Windows, Android and Linux.

LEADTOOLS®

Prices from $ 4,895.10

www.componentsource.com/leadtools-document-imaging-suite

Nevron Vision for .NET

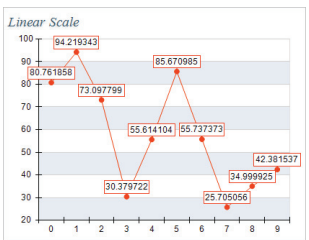
Top 100 Publisher

Corporate Partner

Data visualization components for desktop and web development.

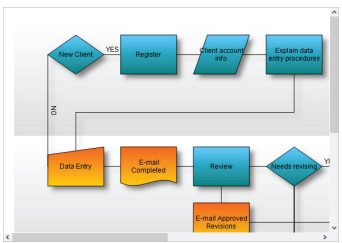
Publisher: Nevron | Category: Presentation Layer

Nevron Vision for .NET offers everything you need to create advanced digital dashboards, reports, diagrams and MMI interfaces. It includes charts, gauges, diagrams, maps and user interface controls.



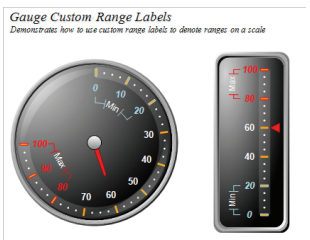
Nevron Chart for .NET

Display virtually any 2D, 3D chart or gauge with superior quality. Packed with hundreds of intuitive examples and many advanced features, Nevron Chart allows you to get started quickly using no code at all.



Nevron Diagram for .NET

A complete diagramming solution packed with interactive features, shapes, automatic layouts, stunning visual effects and comes equipped with ready to use controls to boost your application development.



Nevron Gauge for .NET

Suitable for any application that needs to visualize KPIs or Scorecards. It features a complete set of Radial and Linear gauges, LED displays, State Indicators, Advanced Axes and visual effects.

NEVRON

Prices from $ 1,034.55

www.componentsource.com/nevron-net-vision-ultimate

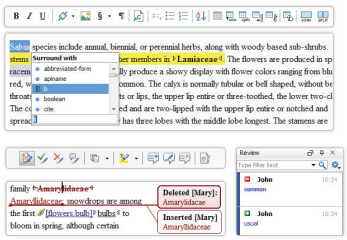
Oxygen XML Editor Enterprise

Top 25 Product | #4 Publisher | Corporate Partner

A complete solution for XML development and authoring.

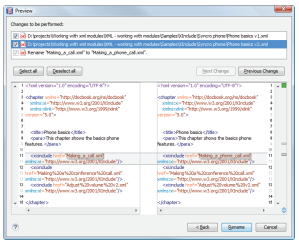
Publisher: Syncro Soft | Category: Structured Document Authoring

A collection of must have tools for XML editing, with support for all types of XML documents and other file types, including XML Schemas, CSS, XSLT, WSDL, RelaxNG, Schematron, Ant, XQuery, and many more.



XML Publishing Frameworks

Oxygen offers CSS-based, visual editing support for a number of important XML documentation frameworks (DITA, DocBook, TEI, XHTML). Also, if you are planning to use other types of XML documents, an API is available for customizing Oxygen.



Update References

When an XML, XSL, XSD, or WSDL resource is renamed or moved in the Project view, Oxygen XML Editor gives you the option to update the references for that resource.

Structured XML Editing

To see the occurrences of an ID in an XML document while in Text mode, you can simply place the cursor inside the ID declaration or reference. The occurrences are marked in the vertical side bar at the right side of the editor.

syncro soft

Prices from $ 945.45

www.componentsource.com/oxygen-xml-editor-enterprise