THE MICROSOFT JOURNAL FOR DEVELOPERS

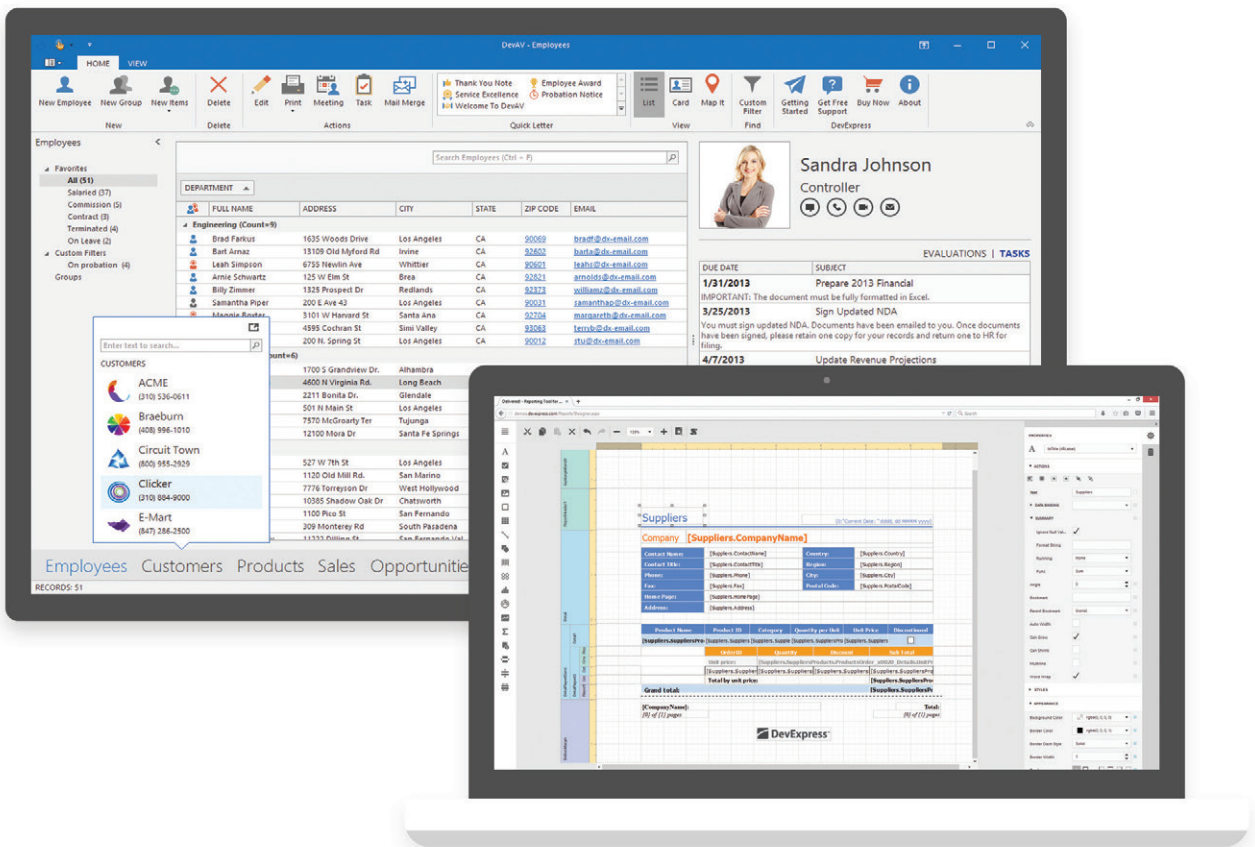
NOVEMBER 2018 VOL 33 NO 11

msdn magazine

DevExpress®

Your Next Great App Starts Here

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



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

devexpress.com/trial

THE MICROSOFT JOURNAL FOR DEVELOPERS

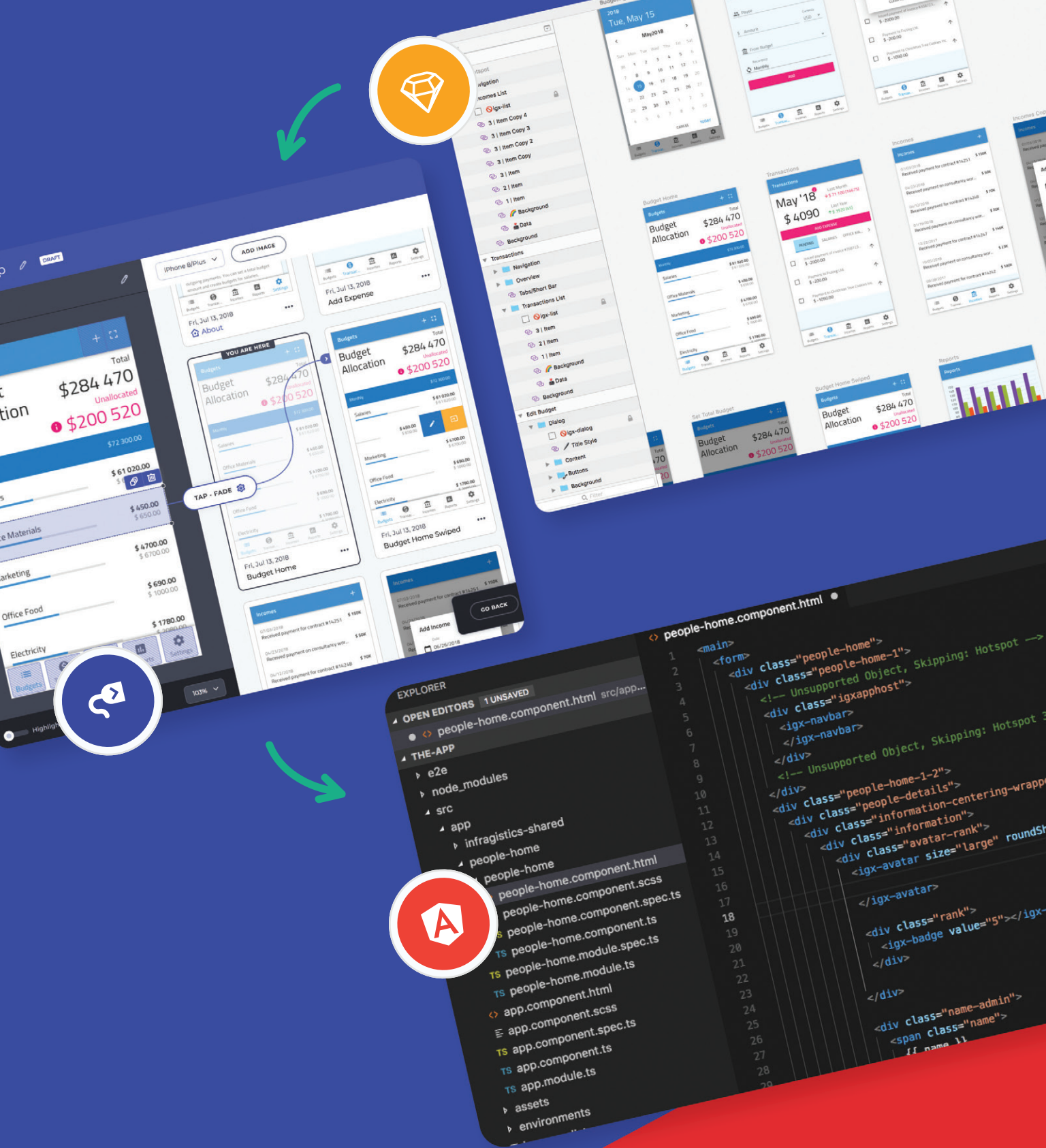
NOVEMBER 2018 VOL 33 NO 11

msdn magazine

New Publishing Options for .NET Core.....18

COLUMNS

Microsoft



indigo.design

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

msdn magazine

NOVEMBER 2018 VOLUME 33 NUMBER 11

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

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

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

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

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

Reaching the Staff

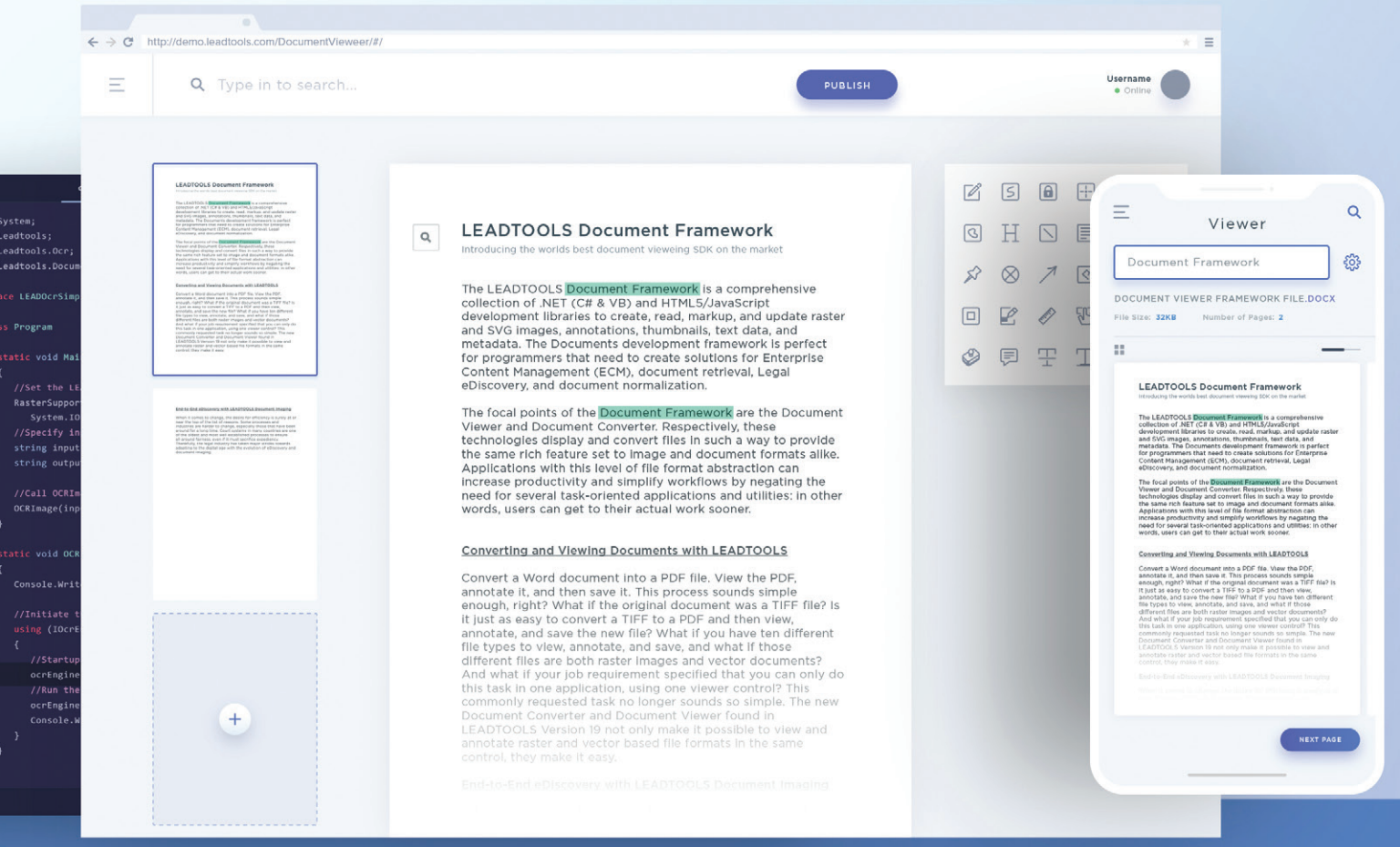
Staff may be reached via e-mail, telephone, fax, or mail.
E-mail: To e-mail any member of the staff, please use the following form: FirstinitialLastname@1105media.com
Irvine Office (weekdays, 9:00 a.m. – 5:00 p.m. PT)
Telephone 949-265-1520; Fax 949-265-1528
4 Venture, Suite 150, Irvine, CA 92618
Corporate Office (weekdays, 8:30 a.m. – 5:30 p.m. PT)
Telephone 818-814-5200; Fax 818-734-1522
9201 Oakdale Avenue, Suite 101, Chatsworth, CA 91311
The opinions expressed within the articles and other contentsherein do not necessarily express those of the publisher.

Microsoft

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

LEADTOOLS®

Multi-platform Document Viewer SDK



With only a few lines of code, developers can use the LEADTOOLS Document Viewer SDK to add rich document viewing features to any project, including text search, annotation, memory-efficient paging, inertial scrolling, and vector display.

- VIEW, CREATE, ORGANIZE & EXPORT A DOCUMENT FROM MULTIPLE FILES
- LOAD, SAVE, CONVERT PDF, DOC, DOCX, RTF, HTML, SVG, AND XPS
- VIEW AND EDIT ANNOTATIONS & MARKUP, INCLUDING PDF AND IBM FILENET
- DRAG AND DROP INTERFACE FOR ADDING AND REARRANGING PAGES
- INTELLIGENT TEXT EXTRACTION USES OCR ONLY WHEN NEEDED

Microsoft .NET · iOS · macOS

Get Started Today

DOWNLOAD OUR FREE EVALUATION

LEADTOOLS.COM

MICHAEL DESMOND

Sensors in Sports: A Dive into Applied Machine Learning

Back in April, Microsoft Senior Architect Kevin Ashley led and co-authored a feature article in the magazine titled "Sensors in Sports: Analyzing Human Movement with AI" (msdn.com/magazine/mt846466). The article explored the effort by the U.S. Olympic Ski Team to improve training techniques and race results through the application of remote sensor telemetry and machine learning (ML). It was a fascinating look at how ML can be used to improve outcomes in a physically dynamic environment.

This month, Ashley returns to our pages, exploring how sensor data and visual artificial intelligence (AI) are being used to improve the performance of athletes on the U.S. Olympic Diving Team. "Sensors in Sports: Analyzing Olympic Diving with Sensors and Vision AI," details how Microsoft worked with coaches, athletes, and program leaders to deploy a sensor platform and smartphone app that lets coaches capture and analyze data related to athletes' dives. At the time of this writing, the Microsoft team had just completed a live exercise at Stanford University with U.S. Olympic Diving coaches. The session showed how the combination of detailed telemetry and high-resolution video allows coaches to identify flaws in technique, refine training regimes, and improve outcomes in competition.

The effort got its start as part of a program at Microsoft to encourage organizations to integrate Internet of Things (IoT) sensors, data ingestion, ML and AI, says Ashley.

"We started Sensors in Sports as a collaborative effort at Microsoft to help our partners improve their integration of connected devices, technology and sports," Ashley explains. "Understanding human movement is key to many industries. The models we build to analyze sports are useful to individual athletes, as well as our partners in many areas related to sports."

Tracking sports movement comes with its challenges. Fast action means that remote sensors must operate at high frequencies, often at 100 samples per second, to produce useful data. Likewise, IoT algorithms must be autonomous and intelligent enough to keep pace. The physical sensors themselves must be hardened against environmental factors, such as water, ice and snow, as well as physical impacts. They must also be small, light and able to operate at low power, so their presence in no way disrupts training sessions.

> At the time of this writing, the Microsoft team had just completed a live exercise at Stanford University with U.S. Olympic Diving coaches.

"Over the last two years working with different sports, I've built about a hundred sensor prototypes, and now with the help of Microsoft Garage we can create even more custom designs for sports," Ashley says, referring to the internal Microsoft program that supports what the company calls the "hack culture at Microsoft."

Sports telemetry is a young and fast-moving discipline, and Ashley says that figuring out what data to capture, measure and analyze is a real challenge.

"This is precisely why we work with coaches and data scientists, training the best athletes in the world," says Ashley. "Each sport is different, and at the same time, fundamentally, many activities are strikingly similar in how the human body reacts to the challenges. Our goal as sports data scientists is finding patterns, using ML to classify and analyze movements and find reusable methods in each activity."

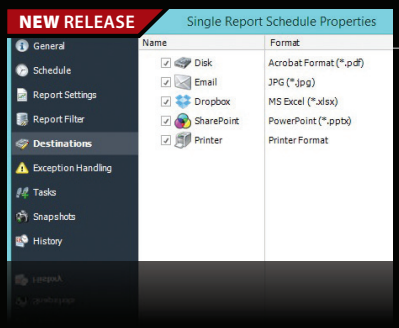
Visit us at msdn.microsoft.com/magazine. Questions, comments or suggestions for *MSDN Magazine*? Send them to the editor: mmeditor@microsoft.com.

DevExpress DXperience 18.1 | from $1,439.99

DevExpress

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

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



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

Christiansteven

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

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

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

LEADTOOLS
THE WORLD LEADER IN IMAGING SDKS

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

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

Help & Manual Professional | from $586.04

ec software

Help and documentation for .NET and mobile applications.

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

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

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

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

TED NEWARD

How To Be MEAN: Testing Angularly

Welcome back again, MEANers.

With any luck, the "debate" around unit testing of code no longer requires discussion—as a developer, you should be looking for ways to automate the testing code you write, without question. Arguments may continue as to whether those tests should come before or after the code in question, perhaps, but it's pretty clear that tests are not an optional part of a modern software project anymore. Which then, of course, raises the question: How do you test an Angular application? I touched briefly on the test files, back in the May 2017 issue when I first started building out some Angular components, but it was hardly expansive (msdn.com/magazine/mt808505). Now it's time to take a deeper look.

Back to Beginnings

Let's take a step back to the beginning of the application. When "ng new" is run to scaffold out the initial stages of the application, it creates all the necessary tools and hooks and files in place to ensure that the application can be tested. In fact, immediately after "ng new," without making even a single change to any file, you can run "ng test" to run the test runner, which in turn executes the code written against the test framework code scaffolded out.

> As a developer, you should be looking for ways to automate the testing code you write, without question.

When run, "npm test" fires up Karma, the test runner, which then fires up a browser instance and runs the tests inside that instance. Karma continues to run, keeping an open WebSocket to the browser so that any changes to the source files can be immediately tested, removing some of the overhead of testing and bringing us closer to a no-wait code-and-test cycle. The project scaffolding provides three tests, encoded in the "app.component.spec.ts" file, which tests the corresponding "app.component.ts" code. As a general rule, each component in the Angular application should have a corresponding ".spec.ts" file to hold all the tests. If each component is created by the Angular CLI, it will usually generate a corresponding test file, with a few exceptions (such as "class") that will require a "--spec" argument to the "generate" command to create the spec (short for "specification") file.

As a matter of fact, let's generate a quick class, Speaker (of course), and then write some tests for it. As usual, create the component with the Angular CLI ("ng generate class Speaker --spec"), and edit the "speaker.ts" file to contain a simple class with five constructor public properties:

```
export class Speaker {
  constructor(public id: number,
    public firstName: string,
    public lastName: string,
    public age: number,
    public bio?: string,
    )
  { }
}
```

The corresponding tests should exercise the various properties to ensure they work as expected:

```
import { Speaker } from './speaker';

describe('Speaker', () => {
  it('should create an instance', () => {
    expect(new Speaker(1, "Ted", "Neward", 47, "Ted is a big geek")).toBeTruthy();
  });
  it('should keep data handy', () => {
    var s = new Speaker(1, "Ted", "Neward", 47, "Ted is a big geek");
    expect(s.firstName).toBe("Ted");
    expect(s.firstName).toEqual("Neward");
    expect(s.age).toBeGreaterThan(40);
  })
});
```

As the class gets more complicated, more complex tests should be added, as well. Much of the power of the testing lies in the "expect" framework, which provides a large number of "toBe" methods to test various scenarios. Here you see several flavors of this, including "toEqual," which does an equality test, and "toBeGreaterThan," which does exactly as its name implies.

But those of you who are following along at home, however, will see something is wrong—after the spec file is saved, the open browser instance turns an ugly shade of red, pointing out that "Ted" is not the same as "Neward"! Sure enough, there's a bug in the second "expect" statement, looking to compare "firstName" when it should be "lastName." This is good, because while test code is testing for bugs in code, it's also the case that sometimes the bug is in the test, and getting that feedback as soon as you write the test helps avoid test bugs.

More Tests

Of course, an Angular app is made up of more than simple classes; it also may include services, which are generally pretty simple to

DevExpress®

From Desktops to Web and Mobile

Your Next Great App Starts Here

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

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

WIN ASP MVC WPF UWP JS

Download your free 30-day trial today.

devexpress.com/try

Figure 1 **The SpeakerService Class That Didn't Make Any HTTP Requests**

```
@Injectable()
export class SpeakerService {

  private static speakersList : Speaker[] = [
    new Speaker(1, "Ted", "Neward", 47,
      "Ted is a big geek living in Redmond, WA"),
    new Speaker(2, "Brian", "Randell", 47,
      "Brian is a high-profile speaker and developer of 20-plus years.
      He lives in Southern California."),
    new Speaker(3, "Rachel", "Appel", 39,
      "Rachel is known for shenanigans the world over. She works for Microsoft."),
    new Speaker(4, "Deborah", "Kurata", 39,
      "Deborah is a Microsoft MVP and Google Developer Expert in Angular,
      and works for the Google Angular team."),
    new Speaker(5, "Beth", "Massi", 39,
      "Beth single-handedly rescued FoxPro from utter obscurity
      and currently works for Microsoft on the .NET Foundation.")
  ]

  public getSpeakers() : Speaker[] { return SpeakerService.speakersList; }
  public getSpeakerById(id : number) : Speaker {
    return SpeakerService.speakersList.find( (s) => s.id == id);
  }
}
```

Figure 2 **Testing the SpeakerService Class**

```
describe('SpeakerService', () => {
  beforeEach(() => {
    TestBed.configureTestingModule({
      providers: [SpeakerService]
    });
  });

  it('should be able to inject the SpeakerService', inject([SpeakerService],
    (service: SpeakerService) => {
    expect(service).toBeTruthy();
  }));

  it('should be able to get a list of Speakers', inject([SpeakerService],
    (service: SpeakerService) => {
    expect(service.getSpeakers()).toBeDefined();
    expect(service.getSpeakers().length).toBeGreaterThan(0);
  }));
});
```

test, because they tend to provide behavior and very little state. In the case of services that provide some behavior on their own, such as formatting or some simple data transformation, testing is easy and reminiscent of testing a class like Speaker. But services also frequently have to interact with the world around them in some fashion (such as making HTTP requests, as the SpeakerService did a few columns back), which means testing them becomes trickier if you don't want to have to include the dependencies. Actually sending requests out over HTTP, for example, would make the tests subject to the vagaries of network communication or server outages, which could yield some false-negative failures and make the tests less deterministic. That would be bad.

It's for these situations that Angular makes such heavy use of dependency injection.

For example, let's start with the version of SpeakerService that didn't make any HTTP requests, as shown in **Figure 1**.

This version is trivial to test, because it's synchronous and requires no outside dependencies, as shown in **Figure 2**.

Notice the "inject" calls in each test? This is basically how Angular manages the dependency injection in the testing environment; it's what allows you to supply any sort of compatible service back end (real or mocked) to the environment.

Typically, the service does a little bit more than what's in the simple SpeakerService, and it's a pain to have to comment out and/or replace the "real" one with a fake one that does nothing, so this is where using a "mock" service works better. Angular has a useful construct called the "Spy" that can insert itself into a regular service and override certain methods to provide a mocked result:

```
it('should be able to get a list of Speakers',
    inject([SpeakerService], (service: SpeakerService) => {
  spy = spyOn(service, 'getSpeakers').and.returnValues([]);
  var speakers: Speaker[] = service.getSpeakers();
  expect(speakers).toBeDefined();
  expect(speakers.length).toBe(0);
}));
```

By using the Spy, you can "override" the method being invoked in the test in order to provide whatever values you want to get back.

Component Tests

A large part of building an Angular application, however, is building components that can appear on the page, and it's important to be able to test those, too. To get an even better understanding of testing a visual component, let's start with a simple on/off toggle-switch type component:

```
@Component({
  selector: 'app-toggle',
  template: `<button (click)="clicked()">
    I am {{isOn ? "on" : "off" }} -- Click me!
  </button>`,
  styleUrls: ['./toggle.component.css']
})
export class ToggleComponent {
  public isOn = false;
  public clicked() { this.isOn = !this.isOn; }
}
```

To test this, you can literally ignore the DOM entirely and just examine the state of the component when various actions are invoked:

```
it('should toggle off to on and off again', () => {
  const comp = new ToggleComponent();
  expect(comp.isOn).toBeFalsy();
  comp.clicked();
  expect(comp.isOn).toBeTruthy();
  comp.clicked();
  expect(comp.isOn).toBeFalsy();
});
```

> Angular has a useful construct called the "Spy" that can insert itself into a regular service and override certain methods to provide a mocked result.

This doesn't check the DOM at all, however, which could hide some critical bugs. Just because the "isOn" property has changed doesn't mean the template has rendered the property correctly, for example. To check for this, you can get hold of the component instance created by the fixture, and examine the DOM rendered for it, like so:

```
it ('should render HTML correctly when clicked', () => {
  expect(fixture.componentInstance.isOn).toBeFalsy();
  const b = fixture.nativeElement.querySelector("button");
  expect(b.textContent.trim()).toEqual("Click me! I am OFF");

  fixture.componentInstance.clicked();
  fixture.detectChanges();

  expect(fixture.componentInstance.isOn).toBeTruthy();
  const b2 = fixture.nativeElement.querySelector("button");
  expect(b2.textContent.trim()).toEqual("Click me! I am ON");
});
```

The "nativeElement" here obtains the DOM node for the component, and I use "querySelector" to do a jQuery-style query to find the relevant DOM node inside—in this case, the button the Toggle creates. From there, I grab its text content (and trim it, because the preceding demo code line breaks in two places that would be awkward to replicate in the test) and compare it against the expected results. However, notice that after I "click" the component, there's a call to "detectChanges"; this is because Angular needs to be told to go process the DOM-relative changes that the event handler might have caused, such as updating the interpolated strings in the template. Without this, the tests will fail despite the component working flawlessly in the browser. (I made this exact mistake while writing the article, in fact, so don't feel bad if you forget to detect the changes.) Note that if the component does any significant initialization inside of its onInit method, the test will need to detectChanges before doing any meaningful work, for the same reason.

Keep in mind, by the way, that all of this test code is against the client-side of the application, not the server-side.

Wrapping Up

Keep in mind, by the way, that all of this test code is against the client-side of the application, not the server-side. Remember all the Express code I wrote to provide APIs to store data into the database and so on? All of that is basically "outside" this framework, and therefore needs to be maintained and run separately. You can use some of the "build" tools I've discussed to run both the server-side and client-side tests as part of a larger test cycle, and thus make sure they get triggered as part of any changes to either client or server. Angular also supports "E2E" (short for "end-to-end") testing, which is outside the scope of what's being talked about here, but is intended to support exactly this situation.

Happy coding! ■

Ted Neward *is a Seattle-based polytechnology consultant, speaker and mentor, currently working as the director of Engineering and Developer Relations at Smartsheet.com. 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: *Garvice Eakins (Smartsheet.com)*

FRANK LA VIGNE

A Closer Look at Reinforcement Learning

In last month's column, I explored a few basic concepts of reinforcement learning (RL), trying both a strictly random approach to navigating a simple environment and then implementing a Q-Table to remember both past actions and which actions led to which rewards. In the demo, an agent working randomly was able to reach the goal state approximately 1 percent of the time and roughly half the time when using a Q-Table to remember previous actions. However, this experiment only scratched the surface of the promising and expanding field of RL.

Recall that in the previous column (msdn.com/magazine/mt830356), an RL problem space consists of an environment, an agent, actions, states and rewards. An agent examines the state of an environment and takes an action. The action then changes the state of the agent and/or environment. The agent receives a reward and examines the updated state of its environment. The cycle then restarts and runs for a number of iterations until the agent succeeds or fails at a predefined goal. When an agent succeeds or fails, the simulation ends. With a Q-table, an agent remembers which actions yielded positive rewards and references it when making decisions in subsequent simulations.

Multi-Armed Bandit Problem

One of the classical problems in RL is the tension between exploration and exploitation. Slot machines, often referred to as "one-armed bandits" are the inspiration for this problem. A bank of slot machines then creates "multi-armed bandit." Each of these slot machine has a probability of paying out a jackpot or not. The probability of each turn resulting in a jackpot may be represented as P, and the probability of not paying out is 1 – P. If a machine has a jackpot probability (JP) of .5, then each pull of the lever has an equal chance of winning or losing. Conversely, a machine with a JP of 0.1 would yield a losing result 90 percent of the time.

Now, imagine a bank of five slot machines and the player (or agent) has a goal to maximize winnings and minimize losses. With no foreknowledge of any of the machines' jackpot probability (JP), the agent must take some risks at first. With the first pull of the lever, the agent wins and receives a payout. However, subsequent tries reveal that this machine pays out about half of the time, a JP of .54. As slot machines go, this is quite generous. The agent must decide if it should exploit the current known resource or explore a new machine. If the probability of the first slot machine paying out is this generous, is it worth trying any of the machines in the bank to see if their payout chances are better?

Code download available at bit.ly/2IvCFIK.

The best way to further explore this problem space is with some Python code in a Jupyter notebook. Create a Python 3 notebook on your preferred platform. I covered Jupyter notebooks in a previous article (msdn.com/magazine/mt829269). Create an empty cell and enter the following code and execute the cell.

```
import numpy as np
import matplotlib.pyplot as plt
number_of_slot_machines = 5
np.random.seed(100)
JPs =  np.random.uniform(0,1, number_of_slot_machines)
print(JPs)
plt.bar(np.arange(len(JPs)),JPs)
plt.show()
```

The output should read and show a plot of the values, as shown in **Figure 1**.

```
[0.54340494 0.27836939 0.42451759 0.84477613 0.00471886]
```

The code creates an array of JP values for a series of five slot machines ranging from 0.004 to 0.844. However, the first machine the agent tried, while generous, is not the best. Clearly, the fourth slot machine with an 84.4 percent payout rate is the best paying machine in the environment. It is also worth noting that the final slot machine has the worst odds of paying out a jackpot. Remember that the agent has no prior knowledge of the payout rates and it must discover them on its own. Had the agent stayed on the first machine, choosing exploitation over exploration, the agent would never have found the best paying slot machine.

To represent what the agent knows at the start of a simulation, add the following code to a new cell:

```
known_JPs = np.zeros(number_of_slot_machines)
```

This creates an array of zeros, meaning that the agent assumes that the JP of each slot machine is zero. While this may not be the best initial value in all cases, it will suffice for our purposes here. To create a simulation of a slot machine, add the following code to a new cell and execute it:

```
def play_machine(slot_machine):
  x = np.random.uniform(0, 1)
  if (x <= JPs[slot_machine]):
    return(10)
  else:
    return(-1)
```

This code snippet simulates a slot machine paying out a reward of 10 if the machine pays out and a negative reward of -1 if the machine does not. Odds of a payout are based on the likelihood defined in the JPs numpy array. To test the code, enter the following Python code into a new cell and execute:

```
# Test Slot Machine 4
for machines in range(10):
  print(play_machine(3))
print ("------")
# Test Slot Machine 5
for machines in range(10):
  print(play_machine(4))
```

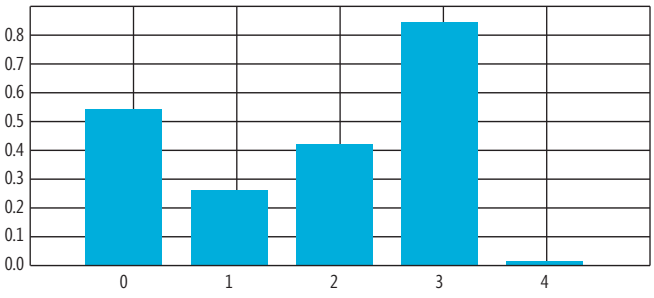


Figure 1 **Jackpot Probabilities of the Five Slot Machines**

This code pits the best performing machine against the worst performing machine. As this is all based on chance, there's no guarantee of the output results. The results should reflect that, with a majority of 10 values for machine 4 and nearly all -1 values for machine 5. With the simulated slot machine code behaving as expected, it's now time to examine a common algorithm in RL: Epsilon Greedy.

The Epsilon Greedy Algorithm

The core dilemma the agent faces here is whether to prioritize greed, the desire to exploit a known resource, or curiosity, the desire to explore other slot machines in the hopes of a better chance of rewards. One of the simplest algorithms for solving this dilemma is known as the Epsilon Greedy algorithm, where the agent chooses at random between using the slot machine with the best odds of payout observed thus far, or trying out another machine in the hopes that it may provide a better payout. With a low value of Epsilon, this algorithm follows the greedy algorithm, but will occasionally try another slot machine. For instance, if the Epsilon value is .1, the algorithm will opt to exploit 90 percent of the time and explore only 10 percent of the time. Typically, default values of Epsilon tend to fall between .05 and .1. In short, the agent will primarily play the best slot machine discovered that it knows of and sometimes try a new machine. Remember that each pull of the lever comes at a cost and the agent doesn't know what we know: that slot 4 pays out the best.

This underscores the notion of RL. The agent knows nothing about the environment initially, so it needs to first explore, then exploit later.

Figure 2 **Reinforcement Learning Code**

```
def multi_armed_bandit(arms, iterations, epsilon):
  total_reward, optimal_action = [], []
  estimated_payout_odds = np.zeros(arms)
  count = np.zeros(arms)
  for i in range(0, iterations):
    epsilon_random = np.random.uniform(0, 1)
    if epsilon_random > epsilon :
      # exploit
      action = np.argmax(estimated_payout_odds)
    else :
      # explore
      action = np.random.choice(np.arange(arms))

    reward = play_machine(action)
    estimated_payout_odds[action] = estimated_payout_odds[action] +
      (1/(count[action]+1)) *
      (reward - estimated_payout_odds[action])

    total_reward.append(reward)
    optimal_action.append(action == np.argmax(estimated_payout_odds))
    count[action] += 1

  return(estimated_payout_odds, total_reward)
```

Learning continues throughout the entire process. Essentially, this is the notion of delayed gratification, and it's in the agent's best interest not to be totally greedy so it leaves some room for exploration.

Testing the Epsilon Greedy Hypothesis

To test this hypothesis, add the code in **Figure 2** to a new cell and execute it. This code creates the multi_armed_bandit function, which simulates a series of runs against a collection of slot machines. The function stores the observed odds of a jackpot payout. At each iteration, the agent will randomly play the slot machine with the best payout it has observed thus far, or arbitrarily try another machine. The argmax function returns the highest value in the numpy array. Here, that means the slot machine with the best odds of hitting a jackpot. The function's parameters allow for control over the number of slot machines, the amount of iterations to run and the value of epsilon.

With the RL code in place, now it's time to test the Epsilon Greedy algorithm. Enter the code from **Figure 3** into an empty cell and execute it. The results show the chart from **Figure 1** for easy reference, followed by the odds that the RL code observed.

As you can see in **Figure 4**, the algorithm did an excellent job, not only of determining the slot machine with the most favorable odds, but also producing fairly accurate payout probabilities for the other four slot machines. The graphs line up rather well. The exception being the fifth slot machine, which has such low odds of a payout that it scored negatively in the agent's observations.

Now, with the baseline established, it's time to experiment some more. What would happen if epsilon were set to zero, meaning that the algorithm will never explore? Enter the following code in a new cell and execute it to run that experiment:

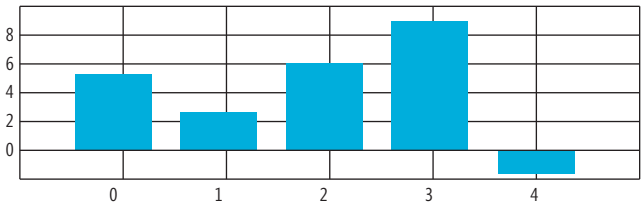
```
print("\n----------------------------------")
print ("Learned Odds with epsilon of 0")
print("----------------------------------")
learned_payout_odds, reward =
  multi_armed_bandit(number_of_slot_machines, iterations, 0)
plt.bar(np.arange(len(learned_payout_odds)),learned_payout_odds)
plt.show()
print (learned_payout_odds)
print ("Reward: ", sum(reward))
```

The resulting chart shows with one value higher than zero. One machine dominates the others, making it quite clear that the agent found one machine and stuck with it. However, run the code several times and you may notice that occasionally an interesting pattern develops. There will be one or more machines with

Figure 3 **Code to Compare the Actual Slot Machine Odds with the Agent's Observations**

```
print ("Actual Odds")
plt.bar(np.arange(len(JPs)),JPs)
plt.show()
print (JPs)
print("----------------------------------")

iterations = 1000
print("\n----------------------------------")
print ("Learned Odds with epsilon of .1")
print("----------------------------------")
learned_payout_odds, reward = multi_armed_bandit(number_of_slot_machines,
iterations, .1)
plt.bar(np.arange(len(learned_payout_odds)),learned_payout_odds)
plt.show()
print (learned_payout_odds)
print ("Reward: ", sum(reward))
```




```
[ 5.38709677  2.66666667  5.72222222  8.34395604  -1. ]
Reward: 7899
```

Figure 4 **Results with an Epsilon Value of .1**

negative values, with one machine with a higher than zero value. In these cases, the agent lost on a given machine and then won on another machine. Once the agent discovers a winning machine, it will stick with that machine, as it will be the machine that the argmax function will choose. If epsilon is set to zero, the agent may still explore, but it will not be intentional. As such, the observed slot machine odds are nowhere near the actual odds. It is also worth noting that the "greedy" method produces a lower reward score than when epsilon was set to .1. Greed, at least absolute greed, would appear to be counterproductive.

What if epsilon were set to 1, making the agent explore every time and not exploit at all? Enter the following code into a new cell and execute it:

```
print("\n----------------------------------")
print ("Learned Odds with epsilon of 1")
print("----------------------------------")
learned_payout_odds, reward  = multi_armed_bandit(number_of_slot_machines,
iterations, 1)
plt.bar(np.arange(len(learned_payout_odds)),learned_payout_odds)
plt.show()
print (learned_payout_odds)
print ("Reward: ", sum(reward))
```

The results will show that the agent did an excellent job of observing odds similar to those of the true odds, and the chart lines up very closely with **Figure 1**. In fact, the results of setting epsilon to 1 look very similar to when the value was .1. Take note of the Reward value, however, and there is a stark difference. The reward value when epsilon was set to .1 will nearly always be higher than when it's set to 1. When the agent is set to only explore, it will try a machine at random at every iteration. While it may be learning from observation, it is not acting on those observations.

Wrapping Up

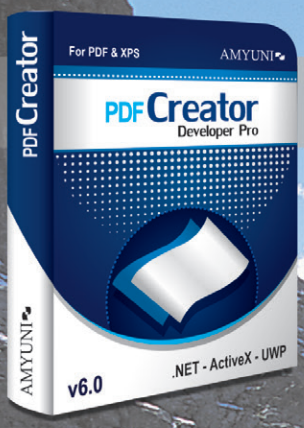
RL remains one of the most exciting spaces in artificial intelligence. In this article, I explored the Epsilon Greedy algorithm with the classic "Multi-Armed Bandit" problem, specifically drilling into the explore-or-exploit dilemma that agents face. I encourage you to further explore the trade offs by experimenting with different values of epsilon and larger amount of slot machines. ■

Frank La Vigne *works at Microsoft as an AI Technology Solutions professional where he helps companies achieve more by getting the most out of their data with analytics and AI. He also co-hosts the DataDriven podcast. He blogs regularly at FranksWorld.com and you can watch him on his YouTube channel, "Frank's World TV" (FranksWorld.TV).*

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

Switch to Amyuni® PDF

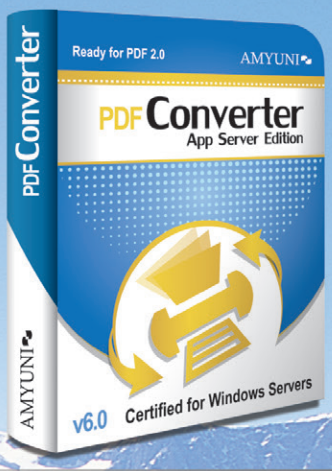
AMYUNI



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

NEW v6.0

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



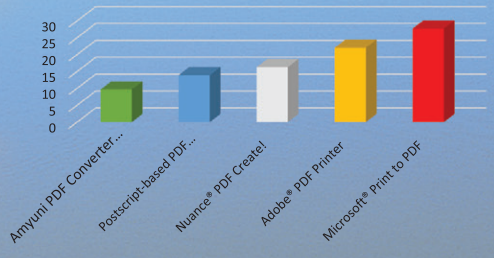
High Performance PDF Printer for Desktops and Servers

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

Complete Suite of PDF, XPS and DOCX Components

- All your PDF processing, conversion and editing in a single package
- Combines Amyuni PDF Converter and PDF Creator for easy licensing, integration and deployment
- Includes our Microsoft® WHQL certified PDF and DOCX printer driver
- Export PDF documents into other formats such as images, HTML5 or editable DOCX
- Import/Export XPS files using our native libraries

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



Other Developer Components from Amyuni®

- Postscript to PDF Library: For document workflow applications that require processing of Postscript documents
- OCR Module: Free add-on to PDF Creator uses the Tesseract 3 engine for accurate character recognition
- Javascript Engine: Integrate a full JS interpreter into your PDF processing applications
- USB Mobile Monitor: Mirror the display of your Windows or Linux system onto your Android device

Windows Server 2016 Certified

Windows Server 2012 Certified

Windows

All development tools available at
www.amyuni.com

AMYUNI Technologies

USA and Canada
Toll Free: 1866 926 9864
Support: 514 868 9227
sales@amyuni.com

Europe
UK: 0800-015-4682
Germany: 0800-183-0923
France: 0800-911-248

LIVE! 360
TECH EVENTS WITH PERSPECTIVE

JOIN US AT

The Ultimate Education Destination

2018 Orlando

ROYAL PACIFIC RESORT AT UNIVERSAL ORLANDO

DECEMBER 2-7, 2018

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

- 6 FULL Days of Training Including Hands-On Labs & Workshops
- 6 Co-Located Conferences for 1 Low Price
- Customize Your Own Agenda from Hundreds of Sessions
- Expert Education and Training
- Knowledge Share and Networking

CONNECT WITH LIVE! 360

twitter.com/live360
@live360

facebook.com
Search "Live 360"

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

instagram.com
@live360_events

EVENT PARTNERS
Microsoft
Magenic

GOLD SPONSOR
SENTRY

SILVER SPONSORS
Progress
GOVERLAN REACH

SUPPORTED BY
msdn magazine
Redmond Channel Partner

FULL AGENDAS AVAILABLE!

Check out live360events.com/orlando for full details.

LAST CHANCE FOR SAVINGS!

REGISTER NOW

REGISTER BY 11/2 AND SAVE $300!

Use promo code: MSDN

See website for details.

6 CO-LOCATED CONFERENCES, 1 GREAT PRICE!

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

Visual Studio Live! features unbiased and practical development training on the Microsoft Platform. Come join us and code in paradise!

SQL Server LIVE!

TRAINING FOR DBAs AND IT PROS

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

TECHMENTOR

IN-DEPTH TRAINING FOR IT PROS

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

NEW! IN 2018

Artificial Intelligence LIVE!

AI FOR DEVELOPERS AND DATA SCIENTISTS

Artificial Intelligence Live! is an innovative, new conference for current and aspiring developers, data scientists, and data engineers covering artificial intelligence (AI), machine learning, data science, Big Data analytics, IoT & streaming analytics, bots, and more.

Office & SharePoint LIVE!

ON-PREMISE, CLOUD & CROSS-PLATFORM TRAINING

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

Modern Apps LIVE!

MOBILE, CROSS-DEVICE & CLOUD DEVELOPMENT

Modern Apps Live!, presented in partnership with Magenic, focuses on how to architect, design and build a complete Modern App from start to finish.

Redmond MAGAZINE | VIRTUALIZATION & Cloud Review | Visual Studio MAGAZINE

PRODUCED BY CONVERGE 360 an 1105 MEDIA company

LIVE360EVENTS.COM/ORLANDO

Publishing Options with .NET Core

Jamie Phillips

The advent of .NET Core brings a new paradigm for publishing your applications with the introduction of two new techniques: framework-dependent deployments (FDDs) and self-contained deployments (SCDs). Each option has advantages and disadvantages that need to be considered before publishing. In this article, I explore both approaches by building a sample application and discussing the strengths and weaknesses of each. I then take a brief look at an additional deployment option—CoreRT—that's in the early stages of development.

Publishing with .NET Core brings a lot of information to consider. Traditionally, the typical deployment option for a desktop or console application in the .NET Framework has been an executable. For an ASP.NET application, it's a DLL. These were the only available options and they were dependent on the type of application being built. With the new paradigm, the significant aspect is whether .NET Core is installed or not. The FDD technique requires that .NET Core be installed while the SCD technique brings the .NET Core runtime with it, which means it can run on a machine that doesn't have .NET Core installed. An ASP.NET application can now additionally be packaged as a standalone console application because it brings its own Web server (Kestrel) as an executable.

For my exploration of the two approaches, I'll use .NET Core SDK 2.1 along with a command-line interface. You can find instructions for installing .NET Core 2.1 at microsoft.com/net/download. I'm using PowerShell almost exclusively to run all the commands that I show in the article, but the command prompt (cmd.exe) should work just fine, as well, and, in fact, these commands will execute on the shells of other OSes, including macOS and Linux. Finally, to experiment with SCDs, the Windows Subsystem for Linux (WSL) is invaluable for enabling first-hand testing of any executables created for Linux. You can learn more about WSL at bit.ly/2Nj7FuQ.

CoreRT is currently in pre-release, so all information is subject to change.

This article discusses:

- Framework-dependent deployments
- Self-contained deployments
- CoreRT—natively compiled deployments

Technologies discussed:

.NET Core, .NET Core CLI, CoreRT

Sample Application

I'm going to create a very basic sample application for testing the two publishing methods using .NET Core 2.1. This will be a console application that prints hello along with the name provided. You can use either cmd.exe or PowerShell to input the following code:

```
> dotnet new console -o Publishing
```

Once the new console application has been created, I open Program.cs and enter the following code:

```
using System;
namespace Publishing
{
  class Program
  {
    static void Main(string[] args) => Console.WriteLine($"Hello {args[0]}!");
  }
}
```

When I execute the application, I see this:

```
> dotnet run -- Jamie
Hello Jamie!
```

As you can see, it works as expected. As a quick aside, the two hyphens syntax (--) is how parameters are passed to the application when using the "dotnet run" command. With my console application complete, I can dive into the publishing methods.

The Publish Command

First, let's look at the options available with the publish command by executing it with the "-h" (help) option:

```
dotnet publish [<PROJECT>] [-c|--configuration] [-f|--framework] [--force]
                           [--manifest]
                           [--no-build] [--no-dependencies] [--no-restore]
                           [-o|--output]
                           [-r|--runtime] [--self-contained]
                           [-v|--verbosity] [--version-suffix]
```

Note that two of these options are especially useful when publishing .NET Core applications: --runtime and --self-contained. These options are used together; runtime specifies which runtime

or runtimes should be targeted when creating an SCD, while self-contained signals that a self-contained package with the runtime needs to be created. If the runtime option is passed, the self-contained option is automatically applied.

Other choices to note are the no-restore and no-build options. The no-restore option won't execute the implicit restore when running the publish command. The no-build option won't build the project or run the restore command, so the publish command will run against the existing build. This is useful in continuous integration/continuous deployment scenarios because a build and restore won't be triggered multiple times. Last, the framework option allows the framework version to be specified. The official documentation can be found at bit.ly/2pizc0L.

Framework-Dependent Deployments

FDDs are the default when executing the publish command under the .NET Core CLI. The publish command creates a platform-agnostic application that can run on any .NET Core runtime that's equivalent to or newer than the minor version used to build the application (though version 2.0 won't run something that targets 1.x, and 3.0 won't run something that targets 2.x). Because the targeted runtime isn't packaged with the application, this option yields the smallest package. If several applications are going to be deployed to a system, allowing the runtime to be shared among the applications reduces overall memory and disk usage on the system. Another advantage of the shared runtime is that any future runtime updates will apply to all applications.

Let's walk through publishing the sample application as an FDD by entering the following at the command line:

```
> dotnet publish
```

This creates output in the bin\Debug\netcoreapp2.0 folder, including the file publishing.dll. This 5KB DLL is the application. Next, run the application using the dotnet CLI, like so:

```
> dotnet Publishing.dll Jamie
Hello Jamie!
```

That's all there is to it.

FDD provides several benefits, including a smaller deployment package, a system-managed runtime, and the ability to share the runtime among multiple apps to reduce disk and memory usage. The app is also guaranteed to run on any platform that has a compatible runtime installed.

The disadvantages of FDDs are that system-wide runtime updates could cause issues as they might be incompatible, and that your application can only run on the framework version (or newer) with which it was compiled. It also requires users to have installed that version of the .NET Core runtime on their machine prior to executing the application.

Self-Contained Deployment

Now let's look at the second approach to publishing provided with the .NET Core CLI—SCDs. I'll use the same publishing command as I did before, but this time I'll pass a runtime identifier (RID) as an option to the publish command. The RID tells the .NET Core CLI which platform or platforms to target when creating the application. In this example, I'll create a Windows 10 x64 build of the application. Any platform can be targeted from any OS, allowing executables for other OSes to be created from Windows. Here's the command I use for this sample:

```
> dotnet publish -r win10-x64
```

Notice that there's a subfolder created under the netcoreapp2.0 folder that's named after the target runtime passed. Inside the win10-x64 folder, you'll see that there's now an executable instead of a DLL. That executable is 77KB, which is 72KB more than the previous application. I can run the executable to show that it still works:

```
> .\Publishing.exe Jamie
Hello Jamie!
```

Now that I have an SCD for Windows 10 x64, I'll build one that targets Ubuntu x64, like so:

```
> dotnet publish -r ubuntu-x64
```

I can then utilize WSL to test that the Ubuntu version executes as desired. I open WSL from Windows and make the newly created application executable, so I can I execute it to make sure it's working. Here's that input and the resulting output:

```
> chmod +x Publishing
> ./Publishing Jamie
Hello Jamie!
```

The upside of SCD is that the .NET Core runtime is controlled and shipped with your application. This is convenient for users, as they aren't required to install the runtime, and any system runtime changes won't impact your application. On the other hand, SCD creates larger deployment packages because the runtime is shipped with the application. You also need to know your target platforms in advance to generate packages for each target. Moreover, every time there's a runtime update that includes security or bug fixes, you must republish every application on the machine to get these fixes rather than installing a single machine-wide runtime update.

In the past, the approach to distributing .NET applications had been to require the .NET Framework to be installed or to package it with the installer. With .NET Core and SCDs, there's no longer a need to create a special installer to deliver a basic application.

As positive as SCD is, there's still one looming issue when deploying these applications to a Linux system. The runtime dependencies for .NET Core aren't included with SCDs, which means users won't need to install the runtime. However, users will need to install six dependencies from the package manager for the Linux distribution. For example, on Ubuntu the following dependencies will still need to be installed:

```
liblttng-ust0
libcurl3
libssl1.0.0
libkrb5-3
zlib1g
libicu52 (for 14.x)
libicu55 (for 16.x)
libicu57 (for 17.x)
libicu60 (for 18.x)
```

To resolve this issue, a developer needs to package the SCD application in a native package format for the distribution so additional dependencies can be defined.

While this may be a negative, the positive is that users don't have to add any of the additional package repositories that would be required if the .NET Core runtime needs to be installed.

SCD is fascinating and deserves more discussion. Though there are several pieces that go into making SCD work, as part of

the .NET CLI installation there are two components that really contribute to this. The first component is the shared runtime, which is a redistributable version of the .NET Core runtime and is consumed by the CLI and end users. The second component is the shared host, which is responsible for consuming the DLL that's generated as part the publish process. The shared host is a generic apphost that allows any .NET Core library (DLL), to be executed as an application. When "dotnet run my.dll" is executed, my.dll is being hosted inside of this shared host. When the SCD application is packaged, what's happening is that the shared runtime, the shared host, and the application DLL are placed together in an executable package, an .exe for Windows, or an appropriate executable file for Linux and macOS. The actual documentation for this design can be found in the .NET CLI repository at bit.ly/2QCgZIp.

Runtime Identifiers

RIDs indicate the supported platforms for creating self-contained deployments for platforms other than Windows. For example, Windows 10 supports x86, x64, ARM and ARM64. Support for ARM addresses Windows 10 Internet of Things (IoT) devices. On Linux, only x64 is supported, with the following distributions: Ubuntu, RedHat, CentOS, Fedora, Debian, Gentoo, OpenSuse, Oracle, Tizen and LinuxMint. In macOS, versions 10.10 through 10.13 are supported. Finally, Android is supported in .NET Core 2.0 or later, which also introduces a portable RID that will build all options for a particular target group. More information can be found in the RID catalog located at bit.ly/2PKXRXi.

CoreRT Deployments

In addition to FDD and SCD, a third option being worked on at Microsoft, called CoreRT, offers the ability to generate native binaries from .NET Core-based code. CoreRT performs ahead-of-time (AoT) compilation using the CoreCLR just-in-time (JIT) compiler. This allows .NET Core code to produce both single executables and libraries that can be consumed by other languages like C++. CoreRT lets .NET developers create libraries and executables that are native to the targeted platform, providing a wider reach for the .NET platform.

Getting started with CoreRT is as simple as adding a NuGet package to a project. Inside the sample project I've been working with, I simply run the following command:

```
> dotnet new nuget
```

With the nuget.config file added, I then call to the .NET MyGet feed using the following lines:

```
<add key="dotnet-core"
  value="https://dotnet.myget.org/F/dotnet-core/api/v3/index.json"/>
<add key="nuget.org"
  value="https://api.nuget.org/v3/index.json" protocolVersion="3"/>
```

Next, I add the CoreRT NuGet package, like so:

```
> dotnet add package Microsoft.DotNet.ILCompiler -v 1.0.0-alpha-*
```

And then I run publish, passing in the platform RID, as shown here:

```
> dotnet publish -r win-x64
```

Finally, I can test the application as a natively compiled application. The application is created in a folder called native and is roughly 6KB in size, which is close to the size of the FDD application and doesn't require any runtime to be shipped with it, as it would using SCD.

The pros of CoreRT are native compilation to a single native binary, with dependencies included, for each target. Execution is faster, because JIT compilation isn't required, and the application should have faster throughput due to compiler optimizations for each target.

The cons are that the application must be compiled for each target platform, which needs to be selected in advance. There's also, for now, the small matter of CoreRT being in pre-release stage.

CoreRT applications work very similarly to how SCD applications work. There's a small native execution engine, the CoreRT native runtime, which provides runtime services like garbage collection that gets compiled with the application into a single native package. And there's a managed portion of CoreRT that's written in C#. The .NET Core application is first compiled with the Roslyn Compiler, then the application, along with CoreRT and CoreFX, is passed through an intermediate language compiler that analyzes the dependencies and shakes the tree so only the absolute minimum number of libraries will be compiled into native code using a compiler based on LLVM. Finally, a linker is used to link the CoreRT native runtime with the compiled native output from the application to produce the final native executable. Matt Warren has an awesome blog post about the subject at bit.ly/2NRS5pj, and of course the GitHub repository for CoreRT has links to parts of the design at github.com/dotnet/corert.

Wrapping Up

SCD offers one big advantage over FDD—it doesn't require the user to install any additional software to support an installed application. I'm used to just building applications for Windows where the .NET framework is typically already available. But when .NET Framework isn't installed, or the wrong version is installed, it can create bad experiences for users.

.NET Core promises to change this, as it will require users to install not only the runtime but a specific version of the runtime to support your application. SCDs may produce larger applications (and, therefore, larger downloads) because the runtime is packaged with the application, but this allows the user to install the application without having to worry about additional requirements. For users outside of Windows, such as those for macOS and Linux, SCD is the common experience that users expect, and it will help with the adoption. In environments that are controlled by the developer or organization, this becomes less of an issue, and FDD would likely have the advantage.

CoreRT—compile to native—deployments are still in the very early stages. These deployments will offer many of the advantages of both FDD and SCD, with no framework installation required and compact application files. However, there's still a journey ahead to make this approach functional. ■

Jamie Phillips *is a senior software development engineer at SentryOne, located in East Tennessee. He's been working with .NET since 2007 and has a keen interest in DevOps and the cloud. He can be found on Twitter: @phillipsj73, his blog at phillipsj.net and GitHub as phillipsj.*

Thanks to the following technical experts for reviewing this article: *Andrew Hall (Microsoft), Daniel Oliver, Cameron Presley (SentryOne)*

Web Site Background Processing with Azure Service Bus Queues

Will Stott

I was working at the United Kingdom Collaborative Trial of Ovarian Cancer Screening (UKCTOCS), when I ran into a problem. Tasked with demonstrating a logistic regression classifier I had developed for a trial project, I began to build a simple, low-budget Web site to support it. But I needed the site to both support long-running background processing and start up the classifier server on-demand.

It was a two-pronged challenge, and one that I was able to tackle using key Microsoft technologies. In this article, I'll explore how I leveraged Azure Functions and Service Bus queues to enable background processing, while in a future piece I'll dive into the Azure Container Service and how it enabled on-demand provisioning of server resources. In that second article I'll show you how to programmatically start and terminate an Azure Container using the Azure.Management.Fluent API.

As for the underlying Web site and its database, they're hosted on Azure as a standard Web App based on ASP.Net Core 2.1 MVC, Entity Framework and SQL Server. Most readers should be able to construct what's required from the system overview shown in **Figure 1** and the OvaryVis model shown in **Figure 2**. Really, the project is just a Web Form that captures three integers representing measurements made of an ovary, saves them in a database record, and displays a results page that shows whether or not the classifier judges the dimensions to be consistent with those of an ovary.

Recreating the project in this article doesn't require much more than rudimentary Web development skills. I do assume that you've already created the ASP.NET Core 2.1 MVC Web App and its SQL Server database, as well as the associated Azure Resources (which are described in the companion article). In terms of tools, you'll need Visual Studio 2017 v15.7 with .NET Core 2.1 SDK and the Web development workload. You'll also need an Azure Subscription, but you can get what you need for free if you're a new customer. All the source code and instructions for creating the Azure resources for the solution shown in **Figure 1** are available at the GitHub repository (bit.ly/2NSiluh).

This article discusses:

- Using Azure Service Bus Queue and Azure Functions to implement background processing for a Web App
- Using Entity Framework to provide database access to an Azure Function
- Provisioning resources using Azure Cloud Shell

Technologies discussed:

Azure Service Bus Queue, Azure Functions, Azure SQL Server, Azure Web App, ASP.NET Core 2.1, Entity Framework Core, C#, Visual Studio 2017

Code download available at:

bit.ly/2NSiluh

Overview

A Web App, like the one developed for this article, supports multiple users by deciding what each of them wants and then returning

an appropriate HTML page (View) to each individual browser. The time taken for the Web App to return a view determines the responsiveness of your Web site and the number of simultaneous users it can support. For this reason there's a pressing need to avoid long-running process when obtaining the data needed by your Web site views. A Service Bus Queue is a good way to address this type of problem, as it allows your Web App to store messages containing the processing details, and then provides an immediate response to the user—even if it's just a view that says come back later for the result. Another software component operates in the background to read the queue and perform the required processing for each message.

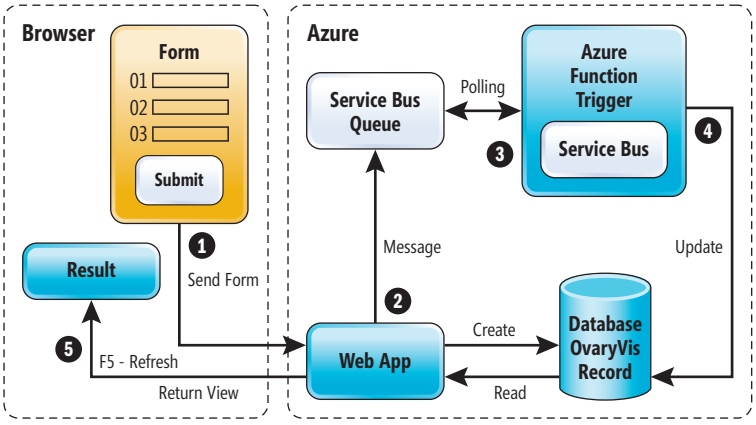


Figure 1 **System Overview**

The advantage of Service Bus is that when lots of users are submitting data for classification, the queue simply gets longer. Yes, this means people may need to wait longer for their results, but it ensures that the Web site remains responsive. It also de-couples your system by separating foreground Web App processing from the component running your background processing. Azure services like Storage queues, EventHubs and EventGrid perform a similar function, but target other types of usage. You can find a useful comparison of Azure Storage queues and Service Bus queues at bit.ly/2QJVwNp, and a comparison of Azure messaging services at bit.ly/2DoY1T5.

> The advantage of Service Bus is that when lots of users are submitting data for classification, the queue simply gets longer.

In the case of my application, Service Bus queue is ideal. It's also cheap and very easy to use, particularly now that there's good integration with Azure Functions, which are perfect for doing my background processing.

Provisioning Azure Resources

Azure Cloud Shell is built into the Azure Portal Web site and allows you to provision Azure Resources using a series of simple commands from the PowerShell Console. To create the Azure Service Bus queue and Function App, you must issue commands from the Cloud Shell, with appropriate values for your subscription, resource group and location. These values will be the same ones you used to provision your Web app and database. You also need to give yourself a unique Service Bus namespace rather than MSDNOvaryVisSBQ. Here are the commands to use:

```
az account set --subscription MsdnOvaryVis
az servicebus namespace create  --name MSDNOvaryVisSBQ --location "WestEurope"
  --resource-group resMSDNOvaryVis --sku Basic
az servicebus queue create --name dimsubmission --namespace-name MSDNOvaryVisSBQ
  --resource-group resMSDNOvaryVis --max-size 1024
```

You'll notice that I opted for the lowest service tier (--sku Basic), which allows me to receive 1 million messages up to 256KB each in size for $0.05 with no monthly charge. In addition to creating a Service Bus queue you also need to create a Function app and a new storage account for it, as shown here:

```
az storage account create -name msdnovaryvisstorageac
  --resource-group resMSDNOvaryVis
  --location "WestEurope" --sku Standard_LRS
az functionapp create --name MSDNOvaryVisFnApp --resource-group resMSDNOvaryVis
  --storage-account msdnovaryvisstorageac --consumption-plan-location westeurope
```

The name of the Function app service is publicly accessible, so you'll need to enter your own name in place of MSDNOvaryVisFnApp. You should also note that creating your first Azure Function results in the creation of a consumption plan, which is

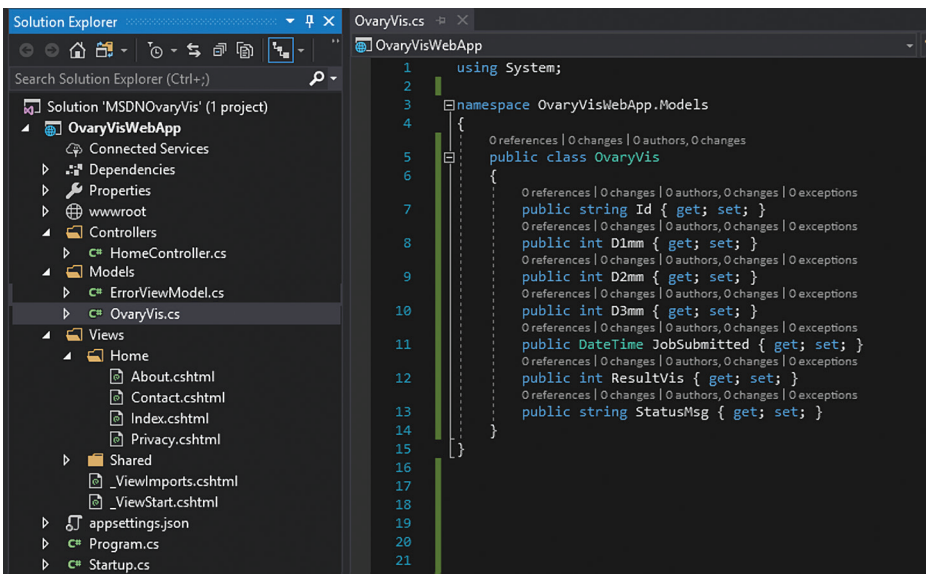


Figure 2 **The OvaryVis Class in the Models Folder**

how you'll be billed for usage, though for a project of this nature the cost should be almost nothing.

Microsoft's intention is that you create Azure Functions without having to explicitly provision a server to run them—known as serverless execution. Microsoft uses consumption plans to bill customers based only on how often a function executes, how long it takes to execute and how much memory it uses. Yes, you can get a fixed-priced bill by electing to run your Azure Functions app service on the same virtual machine (VM) as your Web app, but this option isn't available for Web apps that use shared infrastructure. As a result for this project, it was cheaper to use a consumption plan than to upgrade my Web app service plan.

Now let's apply application settings to the new Function app to complete the provisioning step. Note that you must replace the connection value MMM with the one for your Service Bus, by copying the RootManageSharedAccessKey primary connection string from its Shared access policies blade in the Azure Portal. Here's the code:

```
az functionapp config appsettings set --name MSDNOvaryVisFnApp
  --resource-group resMSDNOvaryVis --settings
'AzureServiceBusQueueName=dimsubmission'
az functionapp config appsettings set --name MSDNOvaryVisFnApp
  --resource-group resMSDNOvaryVis --settings 'AzureWebJobsServiceBus=MMM'
az functionapp config appsettings set --name MSDNOvaryVisFnApp
  --resource-group resMSDNOvaryVis
  --settings 'FUNCTIONS_EXTENSION_VERSION=2.0.12050.0'
```

Of course, in production code you would create an additional key in the Shared access policies blade with just send privileges, and then use its connection string instead of the one from the root key. You also should be aware that your Azure Function executes on a common runtime, which may change if there's an update unless you define a particular version using the FUNCTIONS_EXTENSION_VERSION setting.

I discovered the need to specify my runtime the hard way, as Microsoft updated the pre-release runtime I was working with soon after I completed the development work for this article, causing my function to suddenly stop working. Fortunately, I was able to force Azure Functions to employ a specific runtime version to ensure that my function would only be executed on a server with the same runtime used during development. I don't expect such breaking changes to happen now that the runtime has entered its beta releases, but it's something worth knowing. You can see the runtime version for your own functions by visiting the Azure Function app settings blade in the Azure Portal.

Implementing an Azure Function App Service

The Azure Function needed to perform your Web site's background processing will be developed from a Visual Studio Azure Function app project. The required template is installed on your PC as part of the .NET Core 2.1 SDK and the Web development workload.

> Microsoft's intention is that you create Azure Functions without having to explicitly provision a server to run them—known as serverless execution.

Creating an Azure Function project takes less than a minute. First open the New Project dialog (File | New Project), select Azure Functions from the Cloud template folder in Visual C# and provide a suitable name for your project (OvaryVisFnApp), making sure Add to Solution is selected. In the next dialog provide the type of project (in this case, Empty) and select the storage account you just created. The new OvaryVisFnApp Project will then appear in your Solution Explorer together with an initial set of files.

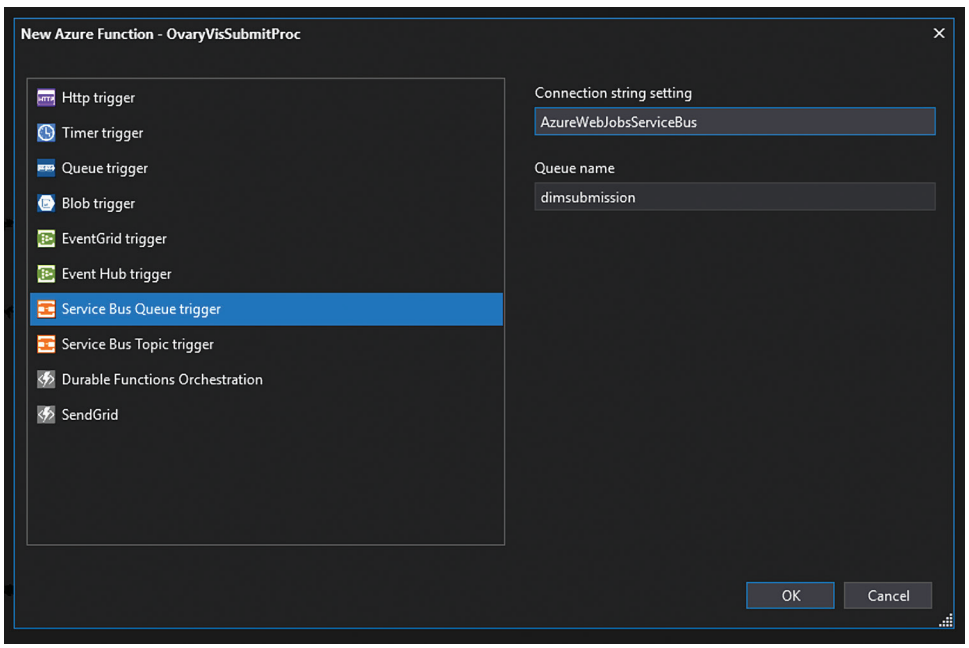
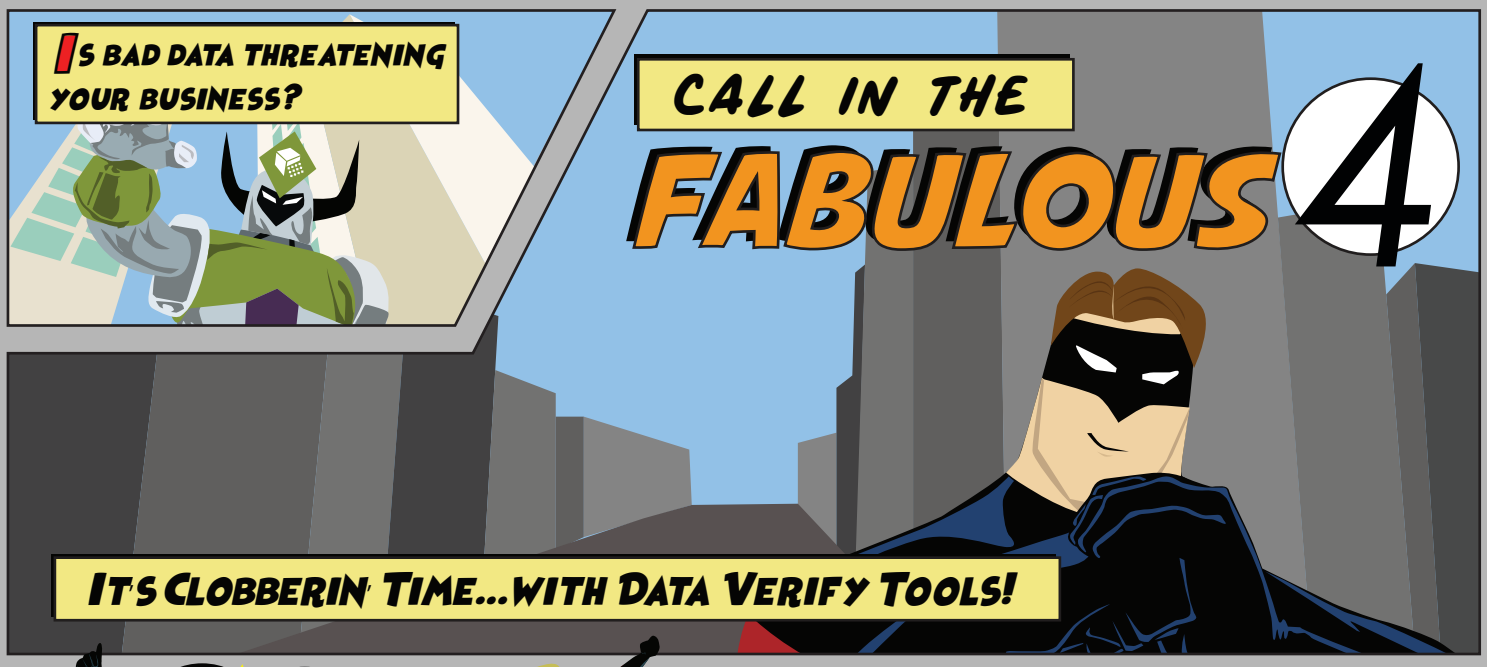


Figure 3 **Creating an Azure Function for Service Bus Queue Trigger**

Now is a good time to make sure you have the latest versions of the packages needed for the project. These are as displayed in the NuGet Solution window when you click Tools | NuGet Package Manager | Manage NuGet Packages for Solution. I used the following for this article, but you might want to try later versions for your own work:

- NETStandard.Library v2.0.3
- Microsoft.NET.Sdk.Functions v1.0.19

To implement the Azure Function itself, you need to add a new class to your OvaryVisFnApp project. Again Visual Studio provides good support. Select the project, right-click Add | New Azure Function and name the file OvaryVisSubmitProc.cs. Next, select Service Bus Queue Trigger as the

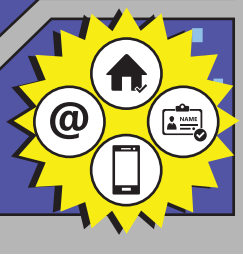


Visit Melissa Developer Portal to quickly combine our APIs (address, phone, email and name verification) and enhance ecommerce and mobile apps to prevent bad data from entering your systems.

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

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

Let's Team Up to Fight Bad Data Today!



melissadeveloper.com
1-800-MELISSA

melissa

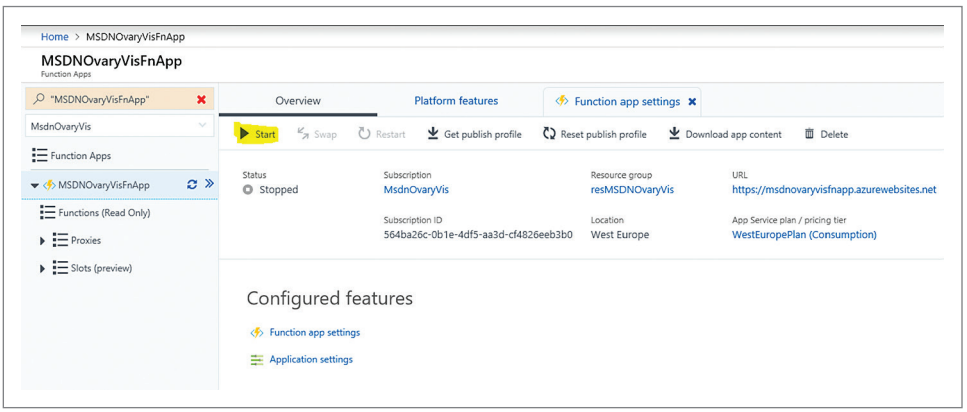


Figure 4 **Azure Function App Service Overview Blade**

type of Azure Function to create with AzureWebJobsServiceBus as the connection string setting and dimsubmission as the queue name (see **Figure 3**). This creates the required class, which you should update by providing myQueueItem as the log.Info parameter, as follows:

```
public static class OvaryVisSubmitProc
{
  [FunctionName("OvaryVisSubmitProc")]
  public static async Task Run([ServiceBusTrigger("dimsubmission",
    Connection = "AzureWebJobsServiceBus")]string myQueueItem,
    TraceWriter log)
  {
    log.Info(myQueueItem));
  }
}
```

Recall that AzureWebJobsServiceBus is the name of the Application setting you applied to your Azure Function app earlier, while dimsubmission is the actual name of your Service Bus queue.

Now you can publish your Azure Function project. Just select the project and right-click Publish. You need to select an existing (rather than new) Azure Function App Service so you can publish your project to the MSDNOvaryVisFnApp resource you created earlier. Helpfully, MSDNOvaryVisFnApp appears in the Resource Group folder in the next dialog box. The only thing to worry about is that the Azure App Service is halted before you publish from Visual Studio, as its DLLs cannot be overwritten when in use by a running process. You can achieve this by issuing the following command from the Cloud Shell:

```
az functionapp stop --name
MSDNOvaryVisFnApp --resource-group
resMSDNOvaryVis
```

After successfully publishing your project, you can start the service by either issuing a start command from the Cloud Shell, or by clicking the start button in the Overview blade, as shown in **Figure 4**.

> Testing your function in the Azure Portal is a good idea at this point, particularly if you've already opened the Overview blade of your Azure Function App Service.

Testing your function in the Azure Portal is a good idea at this point, particularly if you've already opened the Overview blade of your Azure Function App Service. The OvaryVisSubmitProc function is listed on the left of the blade, and selecting it displays a page that allows you to test it, though you might have to click the Test item on the right vertical menu to show this page fully.

The Test page is not particularly pretty, as you can see in **Figure 5**, but it is useful.

The first thing you need to do is type a suitable message in the Request body box—for example, "hello world." Later we'll replace this string by the OvaryVis record ID created by the Web app HomeController.Index (HttpPost) method. However, be aware this message can have much larger and complex content, as well as metadata to describe payload and handling instructions (for more on that, see the Azure Service Messaging document, "Messages, Payloads, and Serialization," at bit.ly/2OCnljX). Indeed, depending on your selected service tier, you

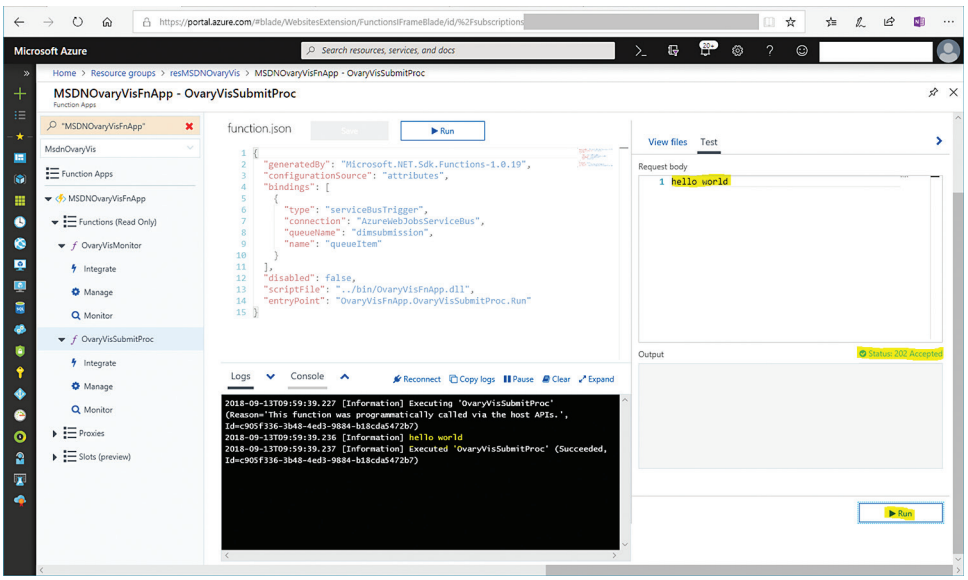


Figure 5 **Azure Function App Service Test Blade**

can create Service Bus queues able to hold up to 80GB of messages with each message being up to 1MB in size.

Once you've created the Request body text in the Test page, you need to click Run to simulate an event being sent to your Run function. The processing of this event can be seen in the Log window, shown at the bottom of **Figure 5**. You should note that the value of the data field in the log is the same as that in the request body. When your function returns, it will cause a Status 202 Accepted message to appear in the Test page. This indicates that your function and the Azure Function App Service are working correctly.

Sending Messages to the Service Bus Queue from a Web App

It's presumed that you've already built a Web app like the one described in the accompanying online article. It just needs to have two Controller methods that handle the creation of a view with a simple form and its subsequent posting back to the Web app. In our case that's a HomeController with Index methods for HttpGet and HttpPost.

Figure 6 **Setting Up the Service Bus Queue Client in the Web App's Startup Class**

```
public class Startup
{
  private static IQueueClient _queueClient = null;
  private static readonly object _accesslock = new object();
  private static void SetupServiceBus(string connection, string queueName)
  {
    lock (_accesslock)
    {
      if (_queueClient == null)
        _queueClient = new QueueClient(connection, queueName);
    }
  }
  public static IQueueClient GetQueueClient() { return _queueClient; }
  public Startup(IConfiguration configuration)
  {
    Configuration = configuration;
    SetupServiceBus(Configuration["OvaryVisServiceBus:Connection"],
      Configuration["OvaryVisServiceBus:QueueName"]);
  }

}
```

Figure 7 **HomeController Index Method**

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Index([Bind("Id,D1mm,D2mm,D3mm")] OvaryVis form)
{
  OvaryVis record = new OvaryVis
  {
    D1mm = form.D1mm,  // First ovary dimension
    D2mm = form.D2mm,  // Second ovary dimension
    D3mm = form.D3mm,  // Third ovary dimension
    JobSubmitted = DateTime.UtcNow,
    ResultVis = -1,    // Result code: -1 = not yet processed
    StatusMsg = String.Format("Job Created at {0}",
      DateTime.UtcNow.ToString("HH:mm:ss"))
  };
                     // Add record to database
  _context.Add(record);
  await _context.SaveChangesAsync();
                     // Send message with ID value to Service Bus Queue
  var message = new Message(Encoding.UTF8.GetBytes(record.Id));
  await Startup.GetQueueClient().SendAsync(message);
                     // Call Result method to return Result page
  return RedirectToAction("Result", new { id = record.Id });
}
```

Installing Azure.ServiceBus package in your Web app project will allow it to use the component needed for sending an event message to your Service Bus queue. The Visual Studio NuGet Package Manager Console enables you to add such packages, as described in Microsoft Docs at bit.ly/2QIiOmZ. You should open the console from the Tools menu (Tools | NuGet Package Manager | Package Manager Console) and then issue the following command:

```
Install-Package Microsoft.Azure.ServiceBus -Version 3.1.0 -Project OvaryVisWebApp
```

This also installs dependencies like WebJobs.Extensions.ServiceBus (3.0.0-beta8), so you should now have everything you need to send messages to your Service Bus queue.

Now it's time to set up the Service Bus queue client. This requires you to pass to it the name of your queue and its connection. I found it more convenient to keep these values as application configuration settings, so I initialized the Service Bus queue client in the Web app's Startup class constructor, as shown in **Figure 6**. I then used the Cloud Shell to give the following commands, again replacing MMM with the same Service Bus connection string used previously when setting up the Azure Function app:

```
az webapp config appsettings set --name MSDNOvaryVisWebApp
  --resource-group resMSDNOvaryVis --settings
'OvaryVisServiceBus:QueueName=dimsubmission'
az webapp config appsettings set --name MSDNOvaryVisWebApp
  --resource-group resMSDNOvaryVis --settings
'OvaryVisServiceBus:Connection=MMM'
```

You'll see in **Figure 6** that the queue client is held in a static variable, which is initially set as null and then initialized by the SetupServiceBus method. The use of lock makes this method thread safe, while testing that _queueClient is null avoids more than one client being constructed. This is overkill for a Web app as the Startup class is initialized only once, but it does allow the SetupServiceBus method to be copied to the Azure Function you'll create in the next article, where such protection will be needed.

Next, I need to update the HttpPost Index method of the HomeController class so that its code is like that shown in **Figure 7**. This allows you to send an event message to your Service Bus queue in the same way you did using the Azure Function App Service Test page. However, you also need to add the following using statements at the top of the file:

```
using Microsoft.Azure.ServiceBus;
using System.Net.Http;
```

You'll see from **Figure 7** that after initially saving the form data as a record in the OvaryVis table, you create a Message object with its body set as the record ID string. After you build and run your Web App locally (Ctrl+F5), the Result page will display Job Created as its StatusMsg upon submission of the Index page's form. In addition, you'll see the record ID value displayed in the Portal's Azure Function log. This indicates that you have not only succeeded in saving a record to the database, but you've also transmitted its ID to the Azure Function. All you need to do now is update your Azure Function so it uses this ID to read the corresponding record from the database in the Azure Function.

Adding Entity Framework to the Azure Function

Entity Framework provides a simple way of retrieving the OvaryVis record from your database using the ID value passed in the Service Bus event message. Implementation involves adding the

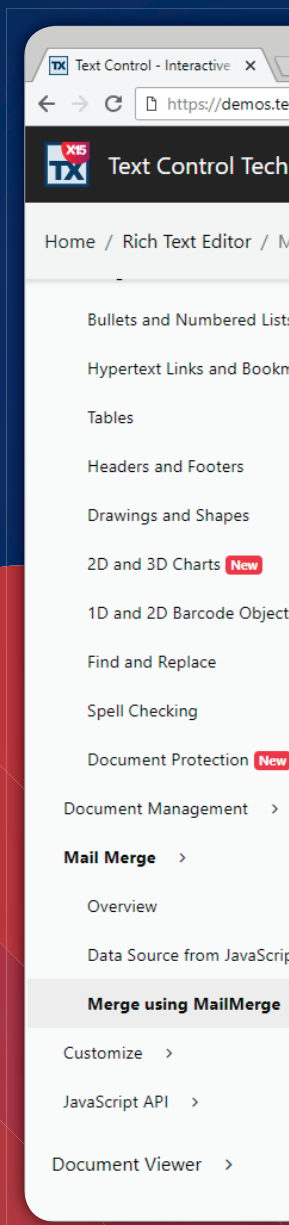
TEXT CONTROL

Integrate Documents and Reporting in Any Platform

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

See our technology live:

demos.textcontrol.com



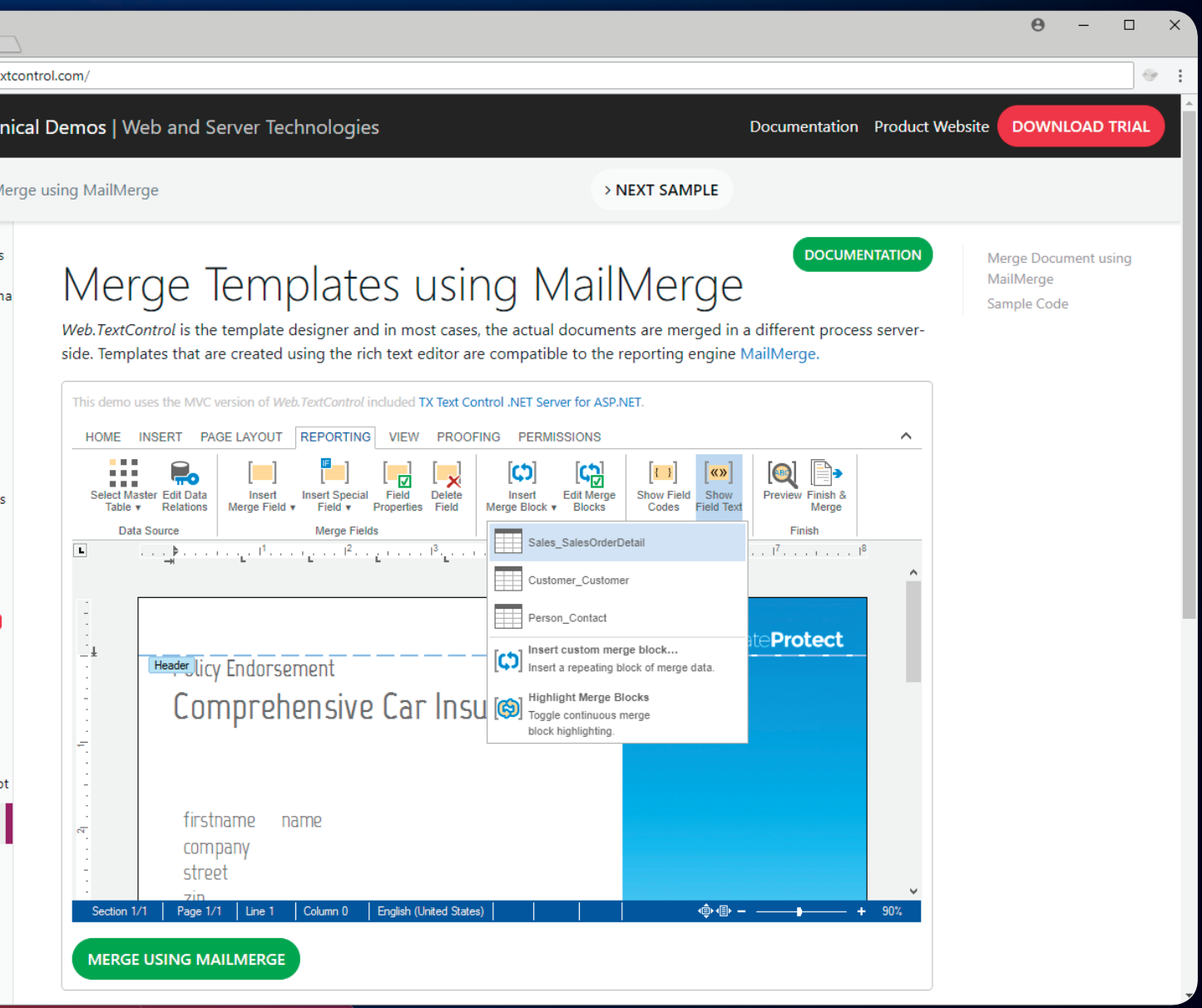
WE ARE CHANGING THE WAY YOU LOOK AT REPORTING

Visual Studio
Partner

SEE THIS LIVE

TX Text Control X15 Technical Demos

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



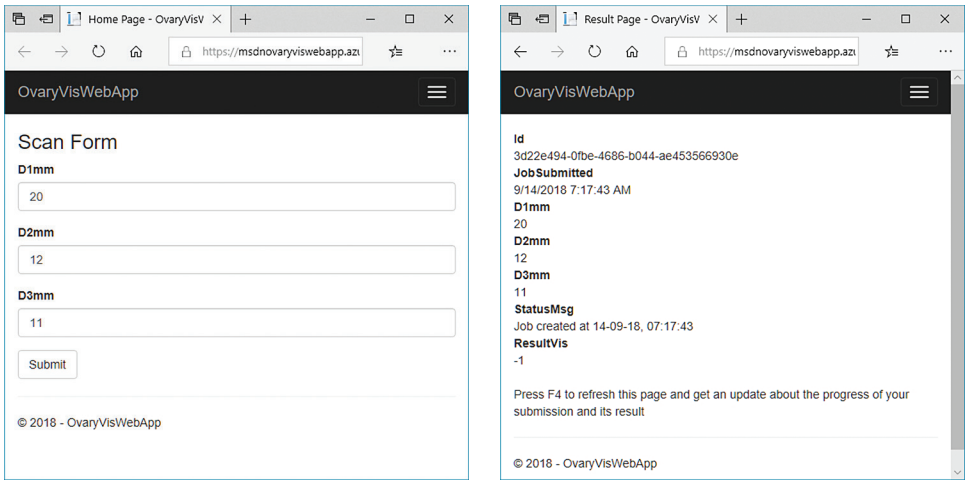


Figure 8 **The Web Site Input Form and Result Page**

Entity Framework package to your Azure Function project, using the same Package Manager Console you used to install the Service Bus package, like so:

```
Install-Package Microsoft.EntityFrameworkCore.SqlServer
  -Project OvaryVisFnApp -version 2.1.2
```

You also need to provide a connection string so your Azure Function can access your Azure SQL Database. This is the same connection string used by your Web app and it's best set from your Azure Function's Applications Settings, which is accessible from its Overview blade in the Portal shown in **Figure 4**. You need to scroll down to the Connection Strings section and then click Add new connection string with the name DefaultConnection, type SQLAzure and the following value, though with the name of your own database server and its username and password:

```
Server=tcp:msdnovaryvisdbsvr.database.windows.net,1433;Database=Msdn.OvaryVisDb;
  User  ID=XXX;Password=YYY;Encrypt=true;Connection Timeout=30;
```

After adding the string, don't forget to scroll back to the top and click Save.

The last thing you need to do to access your database from your Azure Function is copy the DataContext class and the OvaryVis model class from your Web app project to your Function app project. You'll need to change the namespace of both files to OvaryVisFnApp, but otherwise you should be able to rebuild your solution without any problems (press F7). Finally, you need to implement the following method and then call it from the Run method of the OvaryVisSubmitProc function. Here's the code:

```
private static async Task<string> FormSubmittedProc(IConfigurationRoot config,
  ApplicationDbContext dbContext, string queueItem)
{
  string rc = "FormSubmittedProc: ";
  var record = await dbContext.OvaryVis.SingleOrDefaultAsync(a => a.Id ==
queueItem);
  if (record == null)
    rc += string.Format("record not found: Id={0}", msg.Id);
  else
    rc += string.Format("record Id={0} found, ", record.Id);
  return rc;
}
```

The config parameter passed to FormSubmittedProc by the Azure Function Run method is obtained using ConfigurationBuilder and provides access to its application settings, including the database connection string. This is then used to create the ApplicationDbContext object that forms the second parameter. You can find the exact implementation details in the sample code download.

You've completed the development work for this article, so now it's time to rebuild your Visual Studio Solution, publish its Azure Function app project and give it a quick smoke test. If you open the Azure Function App Service Test Blade that I described earlier (**Figure 5**), you should see the ovary record Id displayed in the log whenever you submit an input form from your Web site, as shown in **Figure 8**. This indicates that your Azure Function has successfully found the record corresponding to the Id value contained in the Service Bus message sent from the Web app, so all the parts of the system shown in **Figure 1** are working as intended.

Wrapping Up

It's worth reflecting how easy Azure makes it to develop a robust background processing mechanism for any Web site. With a production site, you'd want to provide some way to automatically update the results page, rather than rely on the user pressing F4—but this isn't too difficult to achieve using SignalR or a similar mechanism. However, the provisioning of an Azure Function App Service and Service Bus queue was achieved with just a few commands from the Azure Portal, and integration with your Web app's Home Controller required only a few lines of code.

There's very little associated cost for this sort of implementation beyond the small monthly charge for an App Service Plan and database server. If you're running on the free tier of the App Service Plan, you should be able to host a test Web site for less than the cost of a couple cups of coffee a month. This makes it a truly compelling solution.

In the next article, I'll explore how to extend an Azure Function by submitting the dimension values in the record to the classifier running on a server provisioned from a Docker image. You'll also see how to start this server automatically when traffic arrives at your Web site, and then stop it after the traffic ceases, implementing an on-demand server that's billed only for the actual time used. ■

Dr. Will Stott *has more than 25 years of experience working as a contractor/consultant for a wide range of companies in the United Kingdom and Europe, including IBM, Cap-Gemini, Logica CMG and Accenture. However, for the last 10 years most of his time has been spent doing research at University College London (UCL) on Ovarian Cancer Screening. Dr. Stott has spoken at many conferences both in the United Kingdom and internationally. He's also the author of papers published in various journals, as well as the book "Visual Studio Team System: Better Software Development for Agile Teams" (Addison-Wesley Professional, 2007).*

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

Create Your Own Script Language with Symbolic Delegates

Thomas Hansen

I love my C# compiler. My girlfriend claims I love it more than I love her, although, for obvious reasons, I'd never admit to that in public. Strong typing, generics, LINQ, garbage collection, CLI, the list of compelling traits goes on. However, every now and then I need a couple of days off from the comfortable securities with which my compiler normally provides me. For those days, I choose to code in dynamic programming languages—and the more dynamic the language, the more dangerous and fun.

OK. Enough introduction. Now let's do what real programmers do when left alone—let's create code:

```
hello world
```

What? You thought I said "*code*"? Yup, I certainly did, and that text actually is code. To understand, let's see what you can do with this code. Take a look at **Figure 1**, which shows a minimalistic example of what I refer to as *symbolic delegates*. Create an empty console app in Visual Studio and type in the code from **Figure 1**.

In just 25 lines of code I've created my own micro programming language. To understand its advantages, realize that the "code" variable can be sent over the network; it can be fetched from a database; it can be stored in a file; and I can dynamically change the string from "hello world" to "world hello," or to "hello hello world world," for that matter, and completely change the result to which it evaluates. This ability means I can dynamically combine delegates to end up with a list of function objects, sequentially evaluated, according to the code's content. All of a sudden I've turned a statically compiled programming language into a dynamic scripting language, simply by splitting a sentence into its words, and using the individual words as lookup keys into a dictionary. The dictionary

This article discusses:

- Symbolic delegates
- Creating a domain-specific language with the Lizzie programming language
- Using Lizzie like JSON, but for code
- The need to simplify modern programming

Technologies discussed:

C#, Delegates

Figure 1 **A Minimalistic Example of Symbolic Delegates**

```
using System;
using System.Collections.Generic;

class MainClass
{
  public static void Main(string[] args)
  {
    // The "programming language"
    var keywords = new Dictionary<string, Action> {

      // The "hello" keyword
      ["hello"] = () => {
        Console.WriteLine("hello was invoked");
      },

      // The "world" keyword
      ["world"] = () => {
        Console.Write("world was invoked");
      }
    };

    // The "code"
    string code = "hello world";

    // "Tokenising" the code
    var tokens = code.Split(' ');

    // Evaluating the tokens
    foreach (var token in tokens) {
      keywords[token]();
    }
  }
}
```

Figure 2 **Dynamically Chaining Delegates**

```
using System;
using System.Linq;
using System.Collections.Generic;

public class Chain<T> : List<Func<T, T>>
{
  public T Evaluate(T input)
  {
    foreach (var ix in this)
    {
      input = ix(input);
    }
    return input;
  }
}

class MainClass
{
  public static void Main(string[] args)
  {
    var keywords = new Dictionary<string, Func<string, string>>
    {
      ["capitalize"] = (input) =>
      {
        return input.Replace("e", "EE");
      },
      ["replace"] = (input) =>
      {
        return input.Replace("o", "0");
      }
    };
    string code = "capitalize replace";
    var tokens = code.Split(' ');
    var chain = new Chain<string>();
    chain.AddRange(tokens.Select(ix => keywords[ix]));
    var result = chain.Evaluate("join the revolution, capitalize and replace");
    Console.WriteLine(result);
  }
}
```

in **Figure 1**, therefore, becomes a "programming language." In effect, it's a symbolic delegate.

This example is obviously not that interesting and is only meant to illustrate the core idea. I'm going to create something slightly more intriguing by chasing these ideas a little bit further down the rabbit hole, as shown in **Figure 2**.

Now I have something that looks almost useful—the ability to dynamically chain together delegates, which results in a lambda object of loosely coupled functions. Thus the code is declaring a chain of functions that transforms an object sequentially, according to the symbols I choose to put into my code. For the record, you shouldn't normally inherit from List, but to keep the example short, I decided to do it to illustrate the main idea.

Figure 3 **Creating a Domain-Specific Language**

```
using System;
using lizzie;

class Demo1
{
  [Bind(Name = "write")]
  object write(Binder<Demo1> binder, Arguments arguments)
  {
    Console.WriteLine(arguments.Get(0));
    return null;
  }
}

class MainClass
{
  public static void Main(string[] args)
  {
    var code = "write('Hello World')";
    var lambda = LambdaCompiler.Compile(new Demo1(), code);
    lambda();
  }
}
```

Expanding upon this idea is simple. The exercise is to find the smallest common denominator for a generic delegate that can describe any programming construct you know, which just so happens to be the following delegate:

```
delegate object Function(List<object> arguments);
```

This delegate can represent nearly every programming structure ever invented. Everything in computing can take a list of input arguments and return something back to the caller. This delegate is the very definition of input/output fundamental to all computing ideas, and becomes an atomic programming structure that you can use to solve all your computing problems.

Meet Lizzie

As I wrote this article, I created a programming language embodying the preceding idea. I wrote the entire language—which I call Lizzie—after my girlfriend, Lisbeth—in a couple of all out, full-moon weekends. The language is entirely contained in a single assembly, roughly 2,000 lines of code. When compiled, it's only 45KB on my disc, and its "compiler" is only 300 lines of C# code. Lizzie is also easily extendable and lets anyone add their own "keywords" to it, allowing you to easily create your own domain-specific language (DSL). One use case for such a language is a rule-based engine, where you need to tie together code more dynamically than C# allows. With Lizzie you can add dynamic script code to your statically compiled C# application that's Turing-complete snippets of functionality. Lizzie is to C# what spice is to your dinner. You don't want to eat only spice, but if you add some spice to your steak, your experience obviously becomes more pleasant. To try Lizzie out, create an empty console application in C#, add Lizzie as a NuGet package, and use the code in **Figure 3**.

In just 22 lines of code I've arguably created my own DSL and added my own domain-specific keyword to the language.

The main feature of Lizzie is that you can bind your Lizzie code to a context type. The LambdaCompiler.Compile method in **Figure 3** is, in fact, a generic method, but its type argument is automatically inferred by its first argument. Internally, Lizzie will create a binder that it binds to your type, making all methods with the Bind attribute available to you from your Lizzie code. When your Lizzie code is evaluated, it has an extra keyword called "write." You can bind any method to your Lizzie code as long as it has the correct signature. And you can bind your Lizzie code to any type.

Lizzie has several default keywords, which it makes available to you for your own code, but you don't have to use these if you don't want to. **Figure 4** shows a more complete example that uses some of these keywords.

The Lizzie code in **Figure 4** first creates a function called "my-function," then it invokes that function with two integer arguments. Finally, it writes out the result of the function invocation to the console. With 21 lines of C# code and eight lines of Lizzie code, I've evaluated a piece of dynamic code that creates a function in a

LIVE! 360
TECH EVENTS WITH PERSPECTIVE

NEW! IN 2018

Artificial Intelligence LIVE!

AI FOR DEVELOPERS AND DATA SCIENTISTS

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

Full Agenda Available Now!
ATTENDAILIVE.COM

A New AI Event for Developers and Data Scientists

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

TRACK TOPICS INCLUDE:

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

REGISTER NOW

SAVE $300 WITH EARLY BIRD PRICING!
REGISTER BY NOVEMBER 2 Use Promo Code MSDN

A Part of Live! 360: The Ultimate Education Destination

6 CONFERENCES, 1 GREAT PRICE

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

ATTENDAILIVE.COM

EVENT PARTNERS
Microsoft
Magenic

GOLD SPONSOR
SENTRY

SILVER SPONSORS
Progress
GOVERLAN REACH

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

PRODUCED BY
CONVERGE 360
An 1105 MEDIA company

AGENDA-AT-A-GLANCE

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

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

Speakers and sessions subject to change

Figure 4 **Using Some Default Keywords**

```
using System;
using lizzie;

class Demo1
{
  [Bind(Name = "write")]
  object write(Binder<Demo1> binder, Arguments arguments)
  {
    Console.WriteLine(arguments.Get(0));
    return null;
  }
}

class MainClass
{
  public static void Main(string[] args)
  {
    var code = @"

// Creating a function
var(@my-function, function({
  +('The answer is ', +(input1, input2))
}, @input1, @input2))

// Evaluating the function
var(@result, my-function(21,2))

// Writing out the result on the console
write(result)

";
    var lambda = LambdaCompiler.Compile(new Demo1(), code);
    lambda();
  }
}
```

dynamic scripting language, from within my C# code, while adding a new keyword to the scripting language I'm using. It took just 33 lines of code in total, including comments. These 33 lines of code allow you to claim you've created your own programming language. Anders Hejlsberg, move over rover, and let little Jimmy take over …

Is Lizzie a "Real" Programming Language?

To answer that question, you need to think about what you consider a real programming language to be. Lizzie is Turing-complete and, at least in theory, allows you to solve every computing problem you might imagine. So, according to the formal definition of what constitutes a "real" programming language, it's certainly as real as any other programming language. On the other hand, it's neither interpreted nor compiled, because every function invocation is simply a dictionary lookup. Moreover, it features only a handful of constructs, and everything is centered around the "function-(arguments)" syntax. In fact, even if statements follow the function syntax previously defined in the generic delegate:

```
// Creates a function taking one argument
var(@foo, function({

  // Checking the value of the argument
  if(eq(input, 'Thomas'), {
    write('Welcome home boss!')
  }, {
    write('Welcome stranger!')
  }) // End of if

}, @input)) // End of function

// Invoking the function
foo('John Doe')
```

The syntax of if is as follows:

```
if(condition, {lambda-true}, [optional] {lambda-else})
```

The first argument to the "if" keyword is a condition. The second argument is a lambda block that's evaluated if the condition yields non-null (true). The third argument is an optional lambda block that's evaluated if the condition yields null (false). So the "if" keyword is in fact a function to which you can supply a lambda argument to, using the "{ … code … }" syntax to declare your lambda. This might feel slightly weird in the beginning, because everything happens in between the opening and closing parentheses of your keywords, unlike other programming languages that use a more traditional syntax. However, in order to create a programming language compiler in 300 lines of code, some bold decisions simply had to be made. And Lizzie is all about simplicity.

Lizzie's functions are surprisingly similar to the structure of an s-expression from Lisp, though without the weird Polish notation. Because an s-expression can describe anything, and Lizzie's functions are simply s-expressions with the symbol (first argument) outside of its parentheses, you can describe anything with Lizzie. This arguably turns Lizzie into a dynamic implementation of Lisp for the Common Language Runtime (CLR), with a more intuitive syntax for C#/JavaScript developers. It allows you to add dynamic code on top of your statically compiled C#, without having to read thousands of pages of documentation to learn a new programming language. In fact, the entire documentation for Lizzie is only 12 pages of text, which means a software developer can literally learn Lizzie in about 20 minutes.

Lizzie—JSON for Code

One of my favorite features of Lizzie is its lack of features. Let me illustrate this with a partial list of what Lizzie can't do. Lizzie can't:

- read or write from your file system
- execute SQL queries
- ask you for your password
- change the state of your computer at all

In fact, a piece of Lizzie code out of the box can't be malicious, not even in theory! This lack of features gives Lizzie some unique abilities that Roslyn and C# scripting don't provide.

In its original state, Lizzie is completely safe, allowing you to securely transmit code over a network, from one computer to another computer, the same way you'd use JSON to transmit data. Then at your endpoint that accepts Lizzie code, you'd have to explicitly implement support for whatever functions you need your Lizzie code to have access to, in order to make it work for your use case. This might include a C# method that reads data from a SQL database or the ability to update data in a SQL database or to read or write files. However, all of these function invocations can be delayed until you've made sure that the code trying to do whatever it's trying to do is actually allowed to do it. Hence, you can easily implement authentication and authorization before you allow it to, for example "insert sql," "read file" or anything else.

This property of Lizzie allows you to create a generic HTTP REST endpoint where the client layer sends Lizzie code to your server and where it's then evaluated. You can then have your server create a JSON response that it sends back to your client. And more interestingly, you can implement this securely. You can implement a single HTTP REST endpoint that accepts only POST requests containing

Lizzie code, and literally replace your entire back end with a 100 percent dynamic and generic Lizzie evaluator. This allows you to move your entire business logic and data access layer into your front-end code, such that your front-end code dynamically creates Lizzie code that it transmits to the server for evaluation. And you can do this securely, assuming you authenticate and authorize your clients before you allow them to evaluate your Lizzie code.

Basically, your entire application is, all of a sudden, easily built in JavaScript or TypeScript or ObjectiveC or whatever, and you can build clients in whatever programming language you like that dynamically creates Lizzie code and send this code to your server.

Lessons from Einstein

When Albert Einstein wrote down his famous equation to explain our universe, it had only three simple components: energy, mass and the speed of light squared. That equation could be easily understood by any 14 year old with a decent grasp of math. Understanding computer programming, on the other hand, requires thousands of pages and millions if not trillions of words, acronyms, tokens, and symbols, plus an entire WikiPedia section on paradigms, numerous design patterns, and a multitude of languages, each with completely different structures and ideas, all of which require "crucial" frameworks and libraries you need to add to your "solution" before you can start working on your domain problem. And you're expected to know all these technologies by heart before you can start referring to yourself as an intermediate software developer. Am I the only one who sees the problem here?

Lizzie is not a magic bullet, and neither are symbolic delegates, but they're definitely a step in the direction of "20 GOTO 10." And sometimes, in order to move forward, you need to start by taking one step back. Sometimes you need to neutrally observe yourself from the outside. If we, as a professional community, do that, we just might realize that the cure for our current madness is simplicity, and not 50 more design patterns, 15 new query languages, 100 new language features, and a million new libraries and frameworks, each with a trillion moving parts.

Less is more, always, so give me more less, and less more! If you want less, join me at github.com/polterguy/lizzie. That's where you'll find Lizzie, with zero type safety, no keywords, no operators, no OOP, and definitely not as much as a single library or framework in sight.

Wrapping Up

Most of the computing industry tends to disagree with most of my ideas. If you asked the average software architect what he thinks about these ideas, he'd probably throw entire libraries in your face as sources to prove how wrong I am. For instance, the prevalent assumption in software development is that strong typing is good and weak typing is bad. For me, however, simplicity is the only game in town, even when it requires throwing strong typing out the window. Keep in mind, though, that ideas such as Lizzie are intended to "spice" up your existing statically typed C# code, not replace it. Even if you never use the coding ideas presented in this article directly, understanding the key concepts may help you write standard code in a traditional programming language more effectively, and help you work toward the goal of simplicity. ■

A Programming History Lesson

Back when I was a junior developer, I used to create 3-tier applications. The idea was to separate the data layers from the business logic layers and the UI layers. The problem is that as front-end frameworks grow in complexity, you're forced to create a 6-tier application architecture. First you need to create a 3-tier server-side architecture, then a 3-tier client-side architecture. And, as if that weren't enough, you then have to port your code to Java, ObjectiveC or whatever to support all possible clients out there. Sorry to be blunt here, but this type of architecture is what I call "insanity-driven design" because it grows the complexity of apps to the point where they're often almost impossible to maintain. A single change in the front-end UI often propagates through 15 layers of architecture and four different programming languages, and forces you to make changes in all those layers just to add a simple column to a data grid in the front end. Lizzie solves this by allowing your front end to send code to your back end, which your back end evaluates and returns to your client as JSON. Sure, you lose type safety, but when type safety comes at the cost of having to tie together the different layers of your applications, such that changes in one place propagate to all other layers in your project, the cost of type safety is simply not worth paying.

I started coding when I was 8 years old, on an Oric 1 in 1982. I clearly remember the first computer program I wrote. It went like this:

```
10 PRINT "THOMAS IS COOL"
20 GOTO 10
```

If an 8-year-old kid today wants to reproduce that experience, following all best practices, using a client-side framework such as Angular and a back-end framework such as .NET, this kid would probably need to know thousands of pages of technical computer science literature by heart. In contrast, I started out with a book that was roughly 300 pages, and a handful of computer magazines. Before I made it to page 100, I had created my own computer game, at the age of 8. Sorry if this makes me sound old, but this is not evolution and improvement, this is "devolution" and madness. And, yes, before you start frenetically writing down your objections, this problem has been scientifically researched, and neutrally observed and confirmed, by professors and Ph.D.s, all much smarter than me.

The fact is that computer programming as a (human) profession is at the brink of extinction, because the complexity that's continually being added may soon reach the point where it surpasses the human brain's capacity to create computer programs. Because programming is becoming so demanding from a cognitive perspective, it may be that no human being will be able to do it 10 years down the road. At the same time, humans are becoming ever more dependent on computers and software every single day. Call me old fashioned here, but I kind of like the idea that a human being somewhere out there is able to understand the things that are crucial to my happiness, existence and quality of life.

Thomas Hansen *has been creating software since he was 8 years old, when he started writing code using the Oric-1 computer in 1982. He refers to himself as a Zen computer programmer who seeks to reduce the complexity of modern programming and refuses to proclaim any belief in technological dogmas. Hansen works for Bright Code in Cyprus where he creates FinTech software.*

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

Sensors in Sports: Analyzing Olympic Diving with Sensors and Vision AI

Kevin Ashley, Phil Cheetham, Olga Vigdorovich, Dan Laak, Kevin Kang, Daria Fradkin

Internet of Things (IoT) and Machine Learning (ML) have been used for everything from inventory tracking and manufacturing quality control to traffic management and weather forecasting. One area where these two disciplines have come together to herald big changes is the arena of high-stakes sports. In the April issue of *MSDN Magazine* (msdn.com/magazine/mt846466), we explored how sensors are being used to track and improve performance in Winter Olympic sports events like alpine skiing. Now in this second installment, the focus shifts to the Summer Olympics, where IoT and ML are being leveraged to improve springboard diving.

It's an area of urgency for the U.S. Olympic Team. There are four diving events—3-meter springboard, 10-meter platform, 3-meter synchronized diving and 10-meter synchronized diving—that account for 24 total available medals in each Olympics. Sensors and ML promise to improve training regimes and maximize athlete performance on the biggest stage.

This article discusses:

- Combining IoT and video for sports movement analysis
- Using the OpenCV computer vision library for video analysis of athletes
- Analyzing high-performance sports with sensors

Technologies discussed:

SensorKit, OpenCV, R, C#

Today, video is the predominant method of analysis in diving, as it has been for years. Coaches use slow motion and frame-by-frame video to analyze dives and give immediate feedback. However, this qualitative approach does not allow rigorous analysis and effective comparison of performance changes. To truly measure and document changes, quantitative analysis that marries video with data captured from specialized sensors is needed.

In this project, we use video and sensor technology to measure the diving takeoff—the critical first moments that are key to a successful dive. We designed a sensor to be placed underneath the tip of the springboard. It includes an inertial measurement unit (IMU), an accelerometer and a gyroscope to measure the important characteristics of the springboard takeoff. This includes the approach to the end of the board, the hurdle onto the end of the board, the pressing down and flexing of the springboard, and the lift into the air by the springboard. A great takeoff is critical for a great dive, with maximum height and spin needed for incredibly difficult dives like a forward 4 1/2 somersault.

Sensor data can determine if the board was maximally depressed on the way down, and if the diver rode the board all the way back up and waited for it to throw him or her into the air. Many divers don't do this well, and tend to "skip off the board" on the way up, losing height and rotation in the process. This project aims to build a system that gives divers and coaches the information they need to rapidly improve the quality of the takeoff, allowing difficult and more difficult dives to be performed successfully.

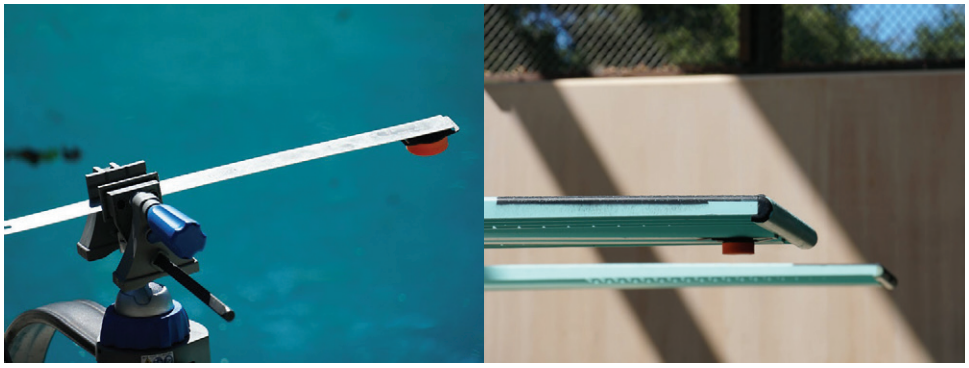


Figure 1 **SensorKit Simulation and Springboard Setup**

Inside the Sensor

Attached to the underside tip of the diving board, the sensor is small and light enough to not affect the performance of the board, and operates for several hours on a single charge (**Figure 1**). The sensor is then ready to measure the angles, accelerations and timing during a diver's takeoff. The most important parameter is maximum flexion angle, which determines how much energy the diver has transmitted into the board prior to takeoff.

A partial list of typical parameters that can be measured and are important to the coach include:

- Maximum downward flexion of the board; indicated by maximum board flexion.
- Hurdle flight time, which indicates maximum hurdle height attained.
- Board contact time, which measures the time from hurdle landing to takeoff.
- Velocity at board contact after hurdle, estimated from the time in the air.
- Velocity at takeoff, calculated from the angular velocity of the board tip and its acceleration.
- Maximum sideway tilt angle of the board tip, which determines if the takeoff occurred from one side of the board.
- Acceleration profile, which provides the raw data for more detailed analysis.

In the end-to-end solution, which is beyond the scope of this research article, this sensor data will be synchronized with video captured of the dive to a smartphone or tablet and then uploaded to the cloud. An intuitive UI will give coaches and athletes immediate access to the video, with controls for slow motion, step-forward and step-backward functions. The integration of sensor telemetry with video gives coaches and divers powerful insight into the mechanics of each dive, so they can understand what needs to be improved. In this solution we will only be calculating maximum flexion angle, maximum springboard velocity and maximum springboard acceleration.

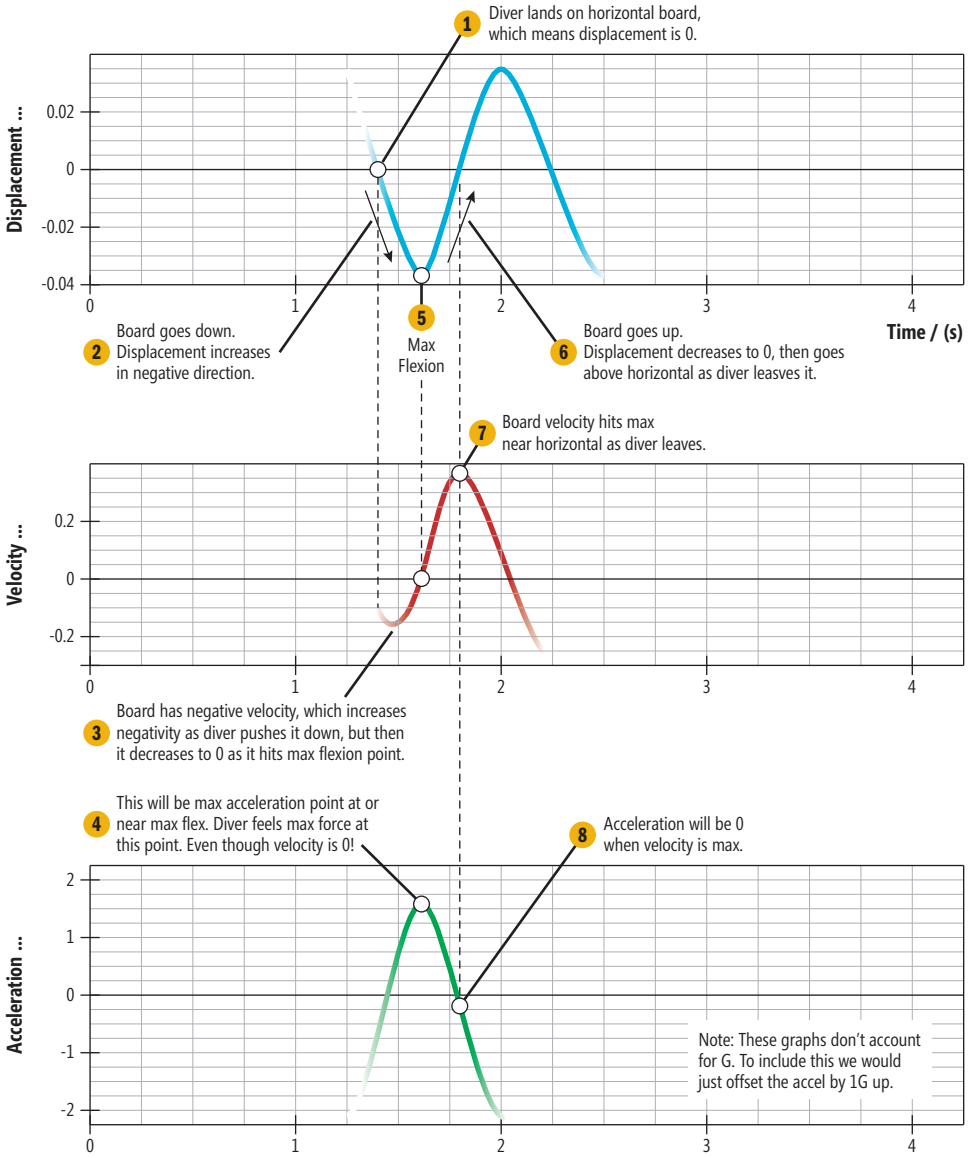


Figure 2 **Physics of the Springboard Explained**

Figure 3 **SensorDiveData and SensorHurdleData Classes**

```
public class SensorDiveData
{
  public long t { get; set; } // Timestamp
  public double dt { get; set; } // Duration
  public double count { get; set; } // Hurdle count
  public double g { get; set; } // Acceleration, m/s^2
  public double w { get; set; } // Angular velocity, dps
  public double angle { get; set; } // Max flexion angle, deg
}
public class SensorHurdleData
{
  public long t { get; set; } // Timestamp
  public double dt { get; set; } // Duration
  public double angle { get; set; } // Flexion angle, deg
  public double g { get; set; } // Acceleration
  public double w { get; set; } // Angular velocity, dps
}
```

Sensor Kit and the Diving App

The data is collected on our SensorKit sensor. The accelerometer and gyroscope provide the data to calculate flexion angle, hurdle flight and dive data, and transmit it wirelessly to the mobile app where the coach can see it. The app can trigger video recording based on sensor detection of a hurdle, and then stop recording after a set amount of time has passed after the sensor-detected takeoff. The app automatically uploads the captured data to the cloud using the SensorKit SDK, and provides a UI for coaches to tag and comment on athletes and their dives.

The sensor needs to be smart enough to process signal patterns specific to diving, which we narrowed to detecting hurdle steps from springboard oscillations, estimating takeoff time and sending calculated values back to the app (**Figure 2**). You can find the source code for the SensorKit SDK for diving apps at bit.ly/2ImaNQT.

SensorKit SDK provides data structures for the hurdle and the dive. The two classes are depicted in the code shown in **Figure 3**.

This data is transmitted to the app through the SensorKit SDK, where it can be stored, for example using Cosmos DB. SensorKit provides a helpful class to handle Cosmos DB data.

Physics of the Dive

Our sensors work in two modes: raw and calculated. We'll work with raw data from the sensors, both from our simulated springboard and from actual dives. There are several types of dives. For this article, we'll focus on dives that start in a standing position at the back of the board, followed by a walk forward and a hurdle on to the end of the board, pressing it down to initiate a takeoff into the air. From our GitHub repository at github.com/kevinash/MSDNDivingSensor, you can use the diving.R file to load and parse sensor data, and calculate the values we used in the analytics section of this article.

In terms of the physics involved, consider pushing against something heavy coming toward you. The force you exert slows the object down, stops it and eventually accelerates it in the other direction. All this impacts the way we must look at finding the diving takeoff events from the velocity and acceleration curves. The problem is to determine where we are in the takeoff based on angular velocity or linear acceleration curves. A chart of board displacement looks like an oscillation graph: From the horizontal position the board goes down to its max flexion point, then goes back via horizontal 0 position and oscillates until it's still.

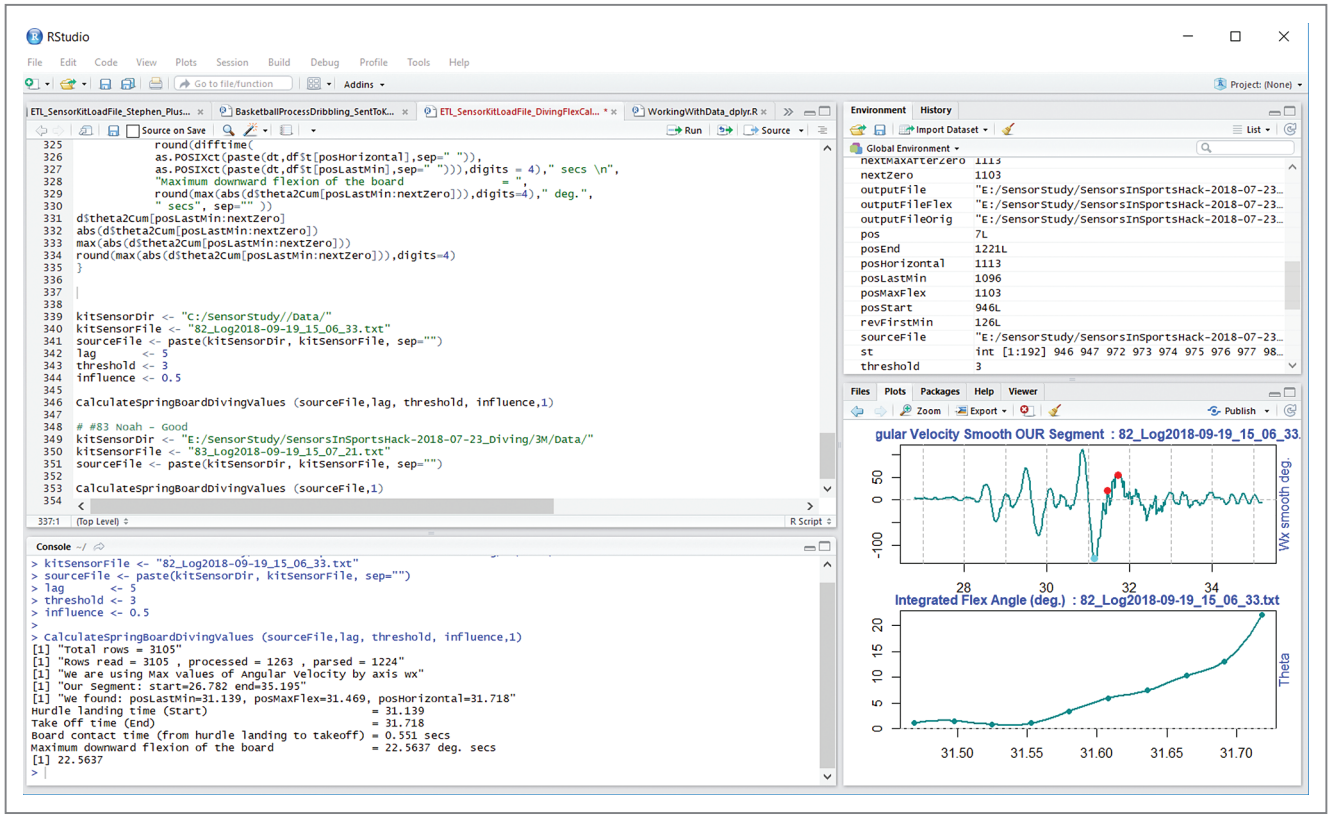
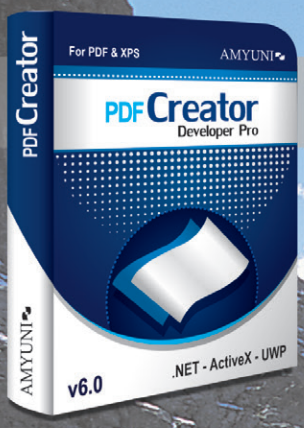


Figure 4 **Sensor Data Analysis in R**

Switch to Amyuni® PDF

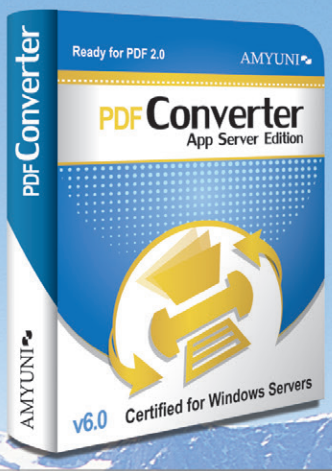
AMYUNI



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

NEW v6.0

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



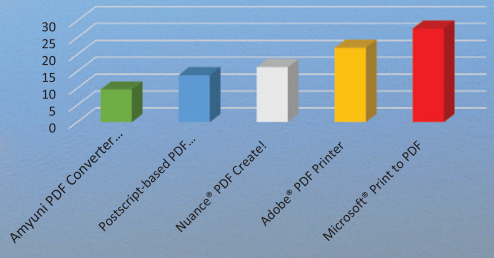
High Performance PDF Printer for Desktops and Servers

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

Complete Suite of PDF, XPS and DOCX Components

- All your PDF processing, conversion and editing in a single package
- Combines Amyuni PDF Converter and PDF Creator for easy licensing, integration and deployment
- Includes our Microsoft® WHQL certified PDF and DOCX printer driver
- Export PDF documents into other formats such as images, HTML5 or editable DOCX
- Import/Export XPS files using our native libraries

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



Other Developer Components from Amyuni®

- Postscript to PDF Library: For document workflow applications that require processing of Postscript documents
- OCR Module: Free add-on to PDF Creator uses the Tesseract 3 engine for accurate character recognition
- Javascript Engine: Integrate a full JS interpreter into your PDF processing applications
- USB Mobile Monitor: Mirror the display of your Windows or Linux system onto your Android device

Windows Server 2016 Certified

Windows Server 2012 Certified

Windows

All development tools available at
www.amyuni.com

AMYUNI Technologies

USA and Canada
Toll Free: 1866 926 9864
Support: 514 868 9227
sales@amyuni.com

Europe
UK: 0800-015-4682
Germany: 0800-183-0923
France: 0800-911-248

To calculate max flex displacement, we can use angular velocity of the tip of the board or linear acceleration. Angular velocity will be out of phase with corresponding acceleration, however, because acceleration is the time derivative of velocity. The same relationship exists between displacement and velocity, because velocity is the time derivative of displacement. That means that velocity will be 0 at the max flexion (bottom point of displacement) of the diving board and acceleration is at max (**Figure 2**, Step 5). After that, the board goes to horizontal position where displacement equals 0 and velocity is max (**Figure 2**, Step 7). We calculated max flexion displacement angle by integrating angular velocity over time from our first point of zero (**Figure 2**, Step 5) to our second point of the takeoff where angular velocity becomes max (**Figure 2**, Step 7).

In our calculations we use the International System of Units (SI).

To start, in the R code (**Figure 4**) you can call the do-it-all function that parses files and displays results, like so:

```
CalculateSpringBoardDivingValues (sourceFile,lag, threshold, influence,1)
```

Let's dig into the details of what's happening inside this call. First, the function loads data from the raw sensor file source into the data frame, with this code:

```
df <- ReadNewFormatFile(dat2, debug_print = debug_print)
```

Sensors like gyro and accelerometer use somewhat-specific data formats, mostly for data optimization purposes. For example, the gyroscope produces angular velocity values in millidegrees per second (mdps). This is done to avoid using floating-point numbers. In our calculations we use the International System of Units (SI), used throughout most physics calculations. For acceleration, we also convert raw sensor values to m/s^2, a standard unit for acceleration. Here are the calculations:

```
#   wx, wy, wz *  0.001 * degrees/second (ang. Velocity)
#   ax, at, az = * 0.001 * 9.81 = m/s^2 (acceleration)
df$wx <- df$wx *  0.001 # convert to degrees per second
df$wy <- df$wy *  0.001 # convert to degrees per second
df$wz <- df$wz *  0.001 # convert to degrees per second
df$ax <- df$ax * 0.001 * 9.81
df$ay <- df$ay * 0.001 * 9.81
df$az <- df$az * 0.001 * 9.81
```

The direction of the angular velocity vector points perpendicular to the plane of the motion, while rotation happens around the axis of rotation, as shown in **Figure 5**. We can determine the axis of rotation by the axis with the most significant value change. A little math here, for angular velocity is the rate of change of the angle over time:

$$\omega = \frac{d0}{dt}$$

To obtain the angle, we can simply integrate angular velocity over time:

$$\omega(t) = \int_0^t w(t)\,dt \approx \sum_0^t \omega(t)\,T_s$$

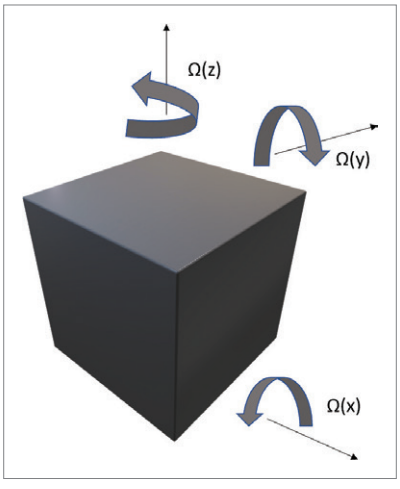


Figure 5 **Gyroscope-Sensing Angular Orientation**

Our drift, or error, is minimized because we only look at the moment when the springboard starts bouncing. In R, this is the code for our integration:

```
# Interested only in [posStart:posEnd]
# When displacement is at MAX, velocity passes 0 point
revFirstMin <- which.min(rev(wSm$avgFilter[posStart:posEnd]))
# After been max negative
posLastMin <- posEnd - revFirstMin +1
# Next index when Angular Velocity goes
# through 0 is our starting integration point
nextZero <- min(which(floor(wSm$avgFilter[posLastMin:posEnd])>0))
# 0 position relative to Last Minimum
nextZero <- nextZero + posLastMin -1
# Max position after 0 position
nextMaxAfterZero <- which.max(wSm$avgFilter[nextZero:posEnd])
nextMaxAfterZero <- nextMaxAfterZero + nextZero -1
```

We also provided plotting functions to draw the results, as shown here:

```
PlotSingleChart(wSm$avgFilter[posStart:posEnd],
  "Angular Velocity Smooth OUR Segment ","cyan4", kitSensorFile,
  "Wx smooth deg.",TRUE, FALSE, d$Start[posStart:posEnd])
```

In the charts on **Figure 6**, we plot data from the gyro sensor while the board is still, through the hurdle and after the takeoff moment. It's clear that the board reaches its maximum amplitude before the takeoff, after which the oscillation diminishes with time. To make our calculation more precise, and minimize drift error from the integration, we only use the moment between when the springboard starts oscillating to the moment when oscillation reaches its maximum.

Finally, we calculate the maximum flexion angle and the takeoff time, with this code:

```
timeTakeOff <- d$Start[posHorizontal]
timeStart <- d$Start[posLastMin]

writeLines (paste (
  "Hurdle landing time (Start)  = ", timeStart,"\n",
  "Take Off time (End)  = ", timeTakeOff,"\n",
  "Board contact time (from hurdle landing to takeoff) = ",
  round(difftime(
  as.POSIXct(paste(dt,df$t[posHorizontal],sep=" ")),
  as.POSIXct(paste(dt,df$t[posLastMin],sep=" "))),digits = 4)," secs \n",
  "Maximum downward flexion of the board  = ",
  round(max(abs(d$theta2Cum[posLastMin:nextZero])),digits=4)," deg.",
  " secs", sep="" ))
```

Getting this sensor data is absolutely essential for diving coaches and athletes, as it provides them with valuable information about the dive, especially the takeoff component that's so vital to a successful dive.

Video Capture and Analysis

Video Analysis Overview Currently, diving coaches rely heavily on what they see to give feedback to their divers. They need to be able to associate the data they receive from our sensors with what they see with their own eyes. By correlating sensor data with video, we give coaches the ability to better understand what they're seeing.

Syncing sensor telemetry with captured video is actually a bit of a challenge. To do so, we turned to a computer vision library, called openCV, to visually determine the exact moment when the diver is at his lowest point, when the board is fully flexed just before takeoff. We then synced that point on the video with the lowest measured point reported by

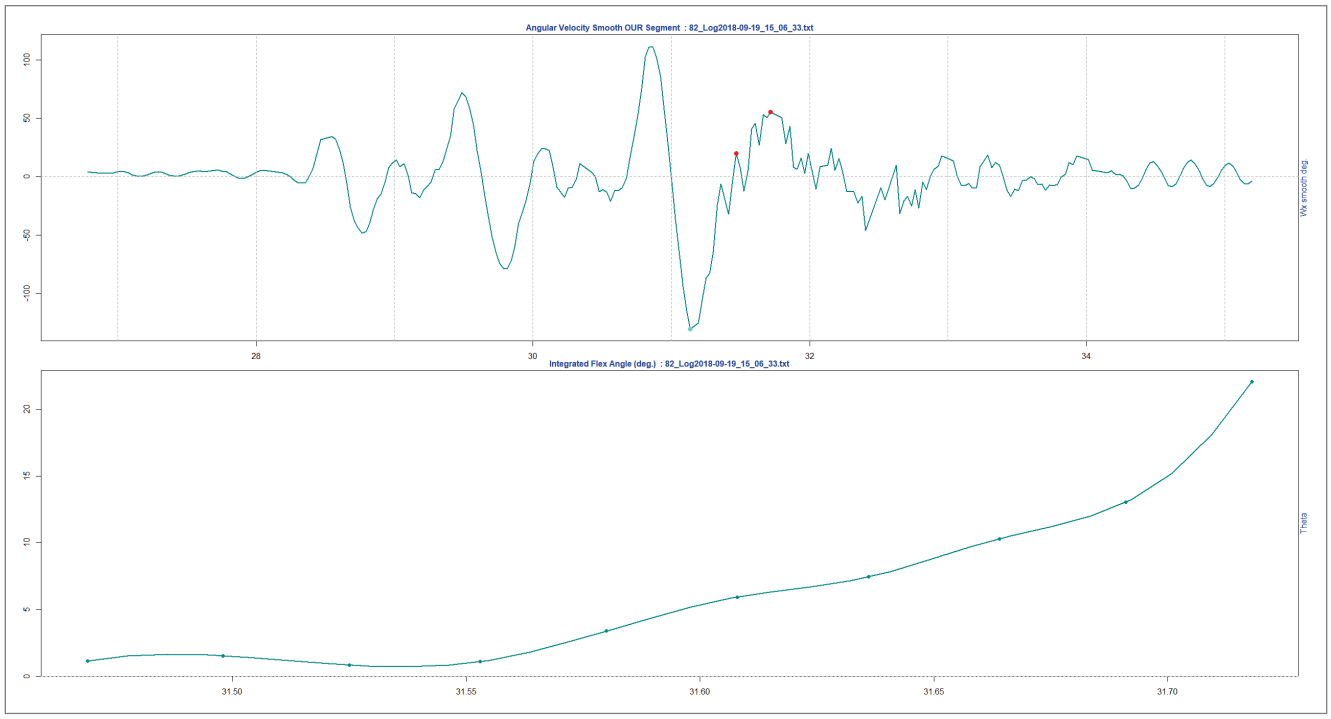


Figure 6 **Charts for Flexion Angle During the Dive**

the sensor. The details of this implementation are discussed further in the code sample available on GitHub at bit.ly/2IjjkU0.

Tracking Diver in Flight Computer vision is also used to determine the exact moment when the diver leaves the board. A simple technique called optical flow, often used in object tracking, provides information on the directionality and velocity of a selected point from one image to another. We used the Lucas-Kanade Optical Flow algorithm to determine the time the diver jumps off the board, as well as to determine the speed and the height of the jump. This video-motion analysis (as shown in **Figure 7**) gives coaches the ability to grasp more insight from each dive.

You can get the source code for movement tracking at bit.ly/2QduA7X. Initially, we specify the source video file for our capture:

```
cap = cv2.VideoCapture("YOUR-VIDEO-FILE.MP4")
```

Once the file is selected, we prompt the user to select a point in the initial video image to be tracked. With this selected, we call the calcOpticalFlowPyrLK method from OpenCV, which calculates an optical flow for a sparse feature set using the iterative Lucas-Kanade method, as shown in **Figure 8**.

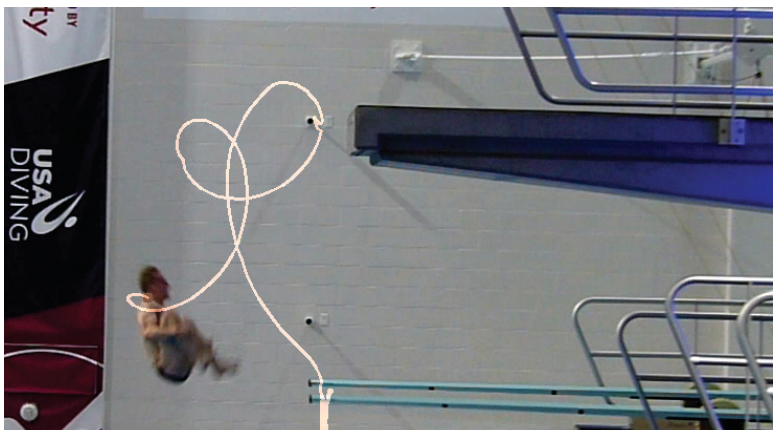


Figure 7 **Tracking Diver Movement in the Air with Video-Motion Analysis**

Wrapping Up

The goal is for this solution to be easy to use for coaches and invisible to divers. These attributes will allow integration into routine training sessions and make it a standard coaching tool, much the way video playback has been over the past two decades. Coaches will be able to assess the quality of the diver's takeoff by analyzing key metrics:

- Time in the air during the hurdle indicates how much potential energy is available to the dive.
- Amount of board flexion indicates how effectively the diver has "loaded the springboard" to launch them into the air.
- Time on the board indicates how well the diver rides the flexed springboard and takes advantage of its energy, converting it into height and spin at takeoff.

The benefits of our combined sensor and video approach are clear. From sensors, we gather immediate insight into important takeoff characteristics, allowing coaches to correct technique. Tracking performance indicators over time also allows assessment of changes in diver takeoffs historically and tracks the progress. This feedback loop can act as a powerful incentive for divers to improve their takeoff metrics at each practice, yielding more height and, thus, more time to complete each dive.

The ultimate goal of the project is to provide more detailed knowledge about takeoff and dive biomechanics for our elite U.S. divers, so they can achieve

Figure 8 **Calling calcOpticalFlowPyrLK**

```
Code samplecv2.calcOpticalFlowPyrLK(old_gray, gray_frame, old_points,
  None, **lk_params)

For visualization, we track the diver on the video:
for i,(new,old) in enumerate(zip(new_points,old_points)):
    a,b = new.ravel()
    if start is None:
      start = a
    c,d = old.ravel()
    mask = cv2.line(mask, (a,b),(c,d), color[i].tolist(), 2)
    frame = cv2.circle(frame,(a,b),5,color[i].tolist(),-1)
    if b > lowest_point and abs(a - start) <= THRESH:
      low = np.zeros_like(frame)
      low = cv2.circle(low, (a,b), 5, color[i+1].tolist(), -1)
      lowest_point = b
  img = cv2.add(frame,mask) # essey
  img = cv2.add(img, low)
  cv2.imshow('Frame',img)
  old_points = new_points.reshape(-1,2)
```

higher scores in competition. To that end, we hope to improve vision analytics, which entails the next steps:

- By using visual distance 1m and 2m markings on the springboard (see **Figure 5**), we can also estimate the size of the board relative to the trajectory to calculate height of the diver in air.
- Using machine vision and ML to determine exactly when the diver lands on the board and takes off from the board will allow accurate calculation of hurdle time, board-contact time and takeoff instant.
- Consider algorithms beyond Lucas-Kanade, which has some constraints. Lucas-Kanade only works on corners, it doesn't work well with lighting changes, and it struggles to register large movements.
- Explore implementing body position and pose estimation from captured video.
- Synchronize video with other useful features in the app, so we can trigger actions in the app based on the analyzed video. ■

Kevin Ashley *is a senior architect at Microsoft, and an expert on IoT, machine learning and sports.*

Phil Cheetham *is currently the senior sport technologist and biomechanist for the U.S. Olympic Committee (USOC) at the Olympic Training Center in Chula Vista, Calif.*

Dan Laak *was named high performance director for USA Diving in 2018 after serving as head diving coach at the University of Georgia the past 31 years.*

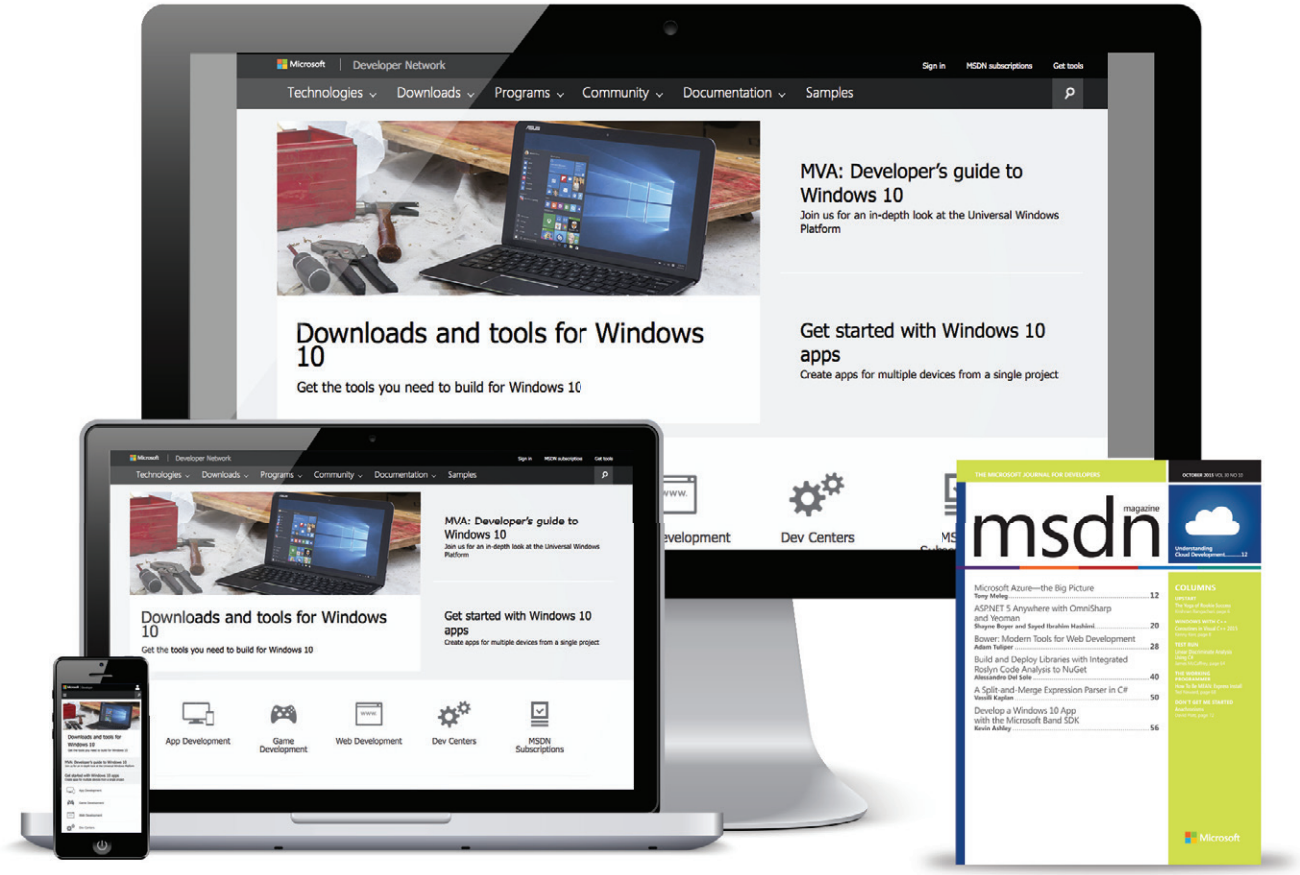
Olga Vigdorovich *is a data scientist and an avid skier. She works with IoT sensor data analysis and builds data models for scalable cloud platforms based on Microsoft Azure.*

Daria Fradkin *is a junior at the University of Pennsylvania in the school of Engineering, studying Digital Media Design, a major that combines Computer Science with higher-level courses in Computer Graphics.*

Kevin Kang *is a fourth-year undergraduate student studying Computer Science with a Data Science option at the University of Washington, Seattle.*

msdn magazine

Where you need us most.



Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS

LIVE! 360
TECH EVENTS WITH PERSPECTIVE

MSDN.microsoft.com

AGENDA-AT-A-GLANCE

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

START TIME	END TIME				
Visual Studio Live! Full Day Hands-On Labs: Sunday, December 2, 2018					
9:00 AM	6:00 PM	**VSS01** Hands-On Lab: Develop an ASP.NET Core 2 and EF Core 2 App in a Day - *Philip Japikse*	**VSS02** Hands-On Lab: From 0-60 in a Day with Xamarin and Xamarin.Forms - *Roy Cornelissen & Marcel de Vries*	**VSS03** Hands-On Lab: Busy Developer's Workshop on VueJS - *Ted Neward*	
2:00 PM	7:00 PM	Pre-Conference Registration - Royal Pacific Resort Conference Center			
Visual Studio Live! Pre-Conference Workshops: Monday, December 3, 2018					
8:30 AM	5:30 PM	**VSM01** Workshop: Web Development in 2018 - *Chris Klug*	**VSM02** Workshop: Architect and Build a Modern ASP.NET App in the Azure Cloud with a full CI/CD Pipeline with VSTS - *Brian Randell and Miguel Castro*	**VSM03** Workshop: Cross-Platform C# Using .NET Core and WebAssembly - *Rockford Lhotka & Jason Bock*	
6:30 PM	8:00 PM	Dine-A-Round Dinner @ Universal CityWalk - 6:30pm - Meet at Conference Registration Desk to walk over with the group			
Visual Studio Live! Day 1: Tuesday, December 4, 2018					
8:00 AM	9:00 AM	**VISUAL STUDIO LIVE! & MODERN APPS LIVE! KEYNOTE: .NET Everywhere and for Everyone** *James Montemagno, Principal Program Manager - Mobile Developer Tools, Microsoft*			
9:15 AM	10:30 AM	**VST01** ASP.NET Core 2 For Mere Mortals - *Philip Japikse*	**VST02** Busy Developer's Guide to Kotlin - *Ted Neward*	**VST03** Azure 101 - *Laurent Bugnion*	**VST04** Modernizing Your Source Control: Migrating to Git from Team Foundation Version Control (TFVC) - *Colin Dembovsky*
10:30 AM	11:00 AM	Networking Break • Visit the EXPO - *Pacifica 7*			
11:00 AM	12:00 PM	**LIVE! 360 KEYNOTE: To Be Announced** - *Pacifica 6*			
12:00 PM	12:45 PM	Lunch • Visit the EXPO			
12:45 PM	1:30 PM	Dessert Break • Visit the EXPO			
1:30 PM	1:50 PM	**VST05** Fast Focus: Serverless Computing: Azure Functions and Xamarin in 20 minutes - *Laurent Bugnion*	**VST06** Fast Focus: Getting Git - *Jason Bock*	**VST07** Fast Focus: Xamarin.Essentials - Cross-Platform APIs for Your Mobile Apps - *James Montemagno*	
2:00 PM	2:20 PM	**VST08** Fast Focus: Getting Started with ASP.NET Core 2.0 Razor Pages - *Walt Ritscher*	**VST09** Fast Focus: Cross Platform Device Testing with xUnit - *Oren Novotny*	**VST10** Fast Focus: Get Your Full .NET Code into .NET Standard - *Rockford Lhotka*	
2:20 PM	2:45 PM	Networking Break • Visit the EXPO - *Pacifica 7*			
2:45 PM	4:00 PM	**VST11** Introduction to Webpack - *Chris Klug*	**VST12** Xamarin.Forms Takes You Places! - *Sam Basu*	**VST13** Azure, Windows and Xamarin: Using the Cloud to Power Your Cross-platform Applications - *Laurent Bugnion*	**VST14** Writing Testable Code and Resolving Dependencies – DI Kills Two Birds with One Stone - *Miguel Castro*
4:15 PM	5:30 PM	**VST15** Angular Application Testing Outside the Church of TDD - *Chris Klug*	**VST16** Busy .NET Developer's Guide to Python - *Ted Neward*	**VST17** HoloLens, Mixed Reality & VR Development with the Cloud - *Nick Landry*	**VST18** Testing in Production Using Azure and Visual Studio Team Services (VSTS) - *Colin Dembovsky*
5:30 PM	7:30 PM	Exhibitor Reception - *Pacifica 7*			
Visual Studio Live! Day 2: Wednesday, December 5, 2018					
8:00 AM	9:15 AM	**VSW01** The Whirlwind Tour of Authentication and Authorization with ASP.NET Core - *Chris Klug*	**VSW02** Essential Tools for Xamarin Developers! - *Sam Basu*	**VSW03** Introduction to Windows Containers and Docker - *Marcel de Vries*	**VSW04** OWASP DevSlop: DevSecOps with VSTS & Azure - *Tanya Janca*
9:30 AM	10:45 AM	**VSW05** Assembling the Web - A Tour of WebAssembly - *Jason Bock*	**VSW06** Cross-Platform App Dev with Xamarin and CSLA .NET - *Rockford Lhotka*	**VSW07** Microservices with AKS (Azure Kubernetes Service) - *Vishwas Lele*	**VSW08** To Be Announced
10:45 AM	11:30 AM	Networking Break • Visit the EXPO - *Pacifica 7*			
11:30 AM	12:30 PM	**LIVE! 360 KEYNOTE: Enterprise Transformation (And You Can Too)** *Donovan Brown, Principal DevOps Manager, Cloud Developer Advocacy Team, Microsoft*			
12:30 PM	1:30 PM	Birds-of-a-Feather Lunch			
1:30 PM	2:00 PM	Dessert Break • Visit the EXPO			
2:00 PM	3:15 PM	**VSW09** No Strings Attached: JavaScript without Webpack, Transpilers, or Frameworks - *Ashley Grant*	**VSW10** DevOps for the SQL Server Database - *Brian Randell*	**VSW11** Programming with Microsoft Flow - *Walt Ritscher*	**VSW12** What's New in C# 7 - *Phil Japikse*
3:15 PM	4:00 PM	Networking Break • Visit the EXPO • Expo Raffle @ 3:30 p.m. - *Pacifica 7*			
4:00 PM	5:15 PM	**VSW13** Electron: Desktop Development For Web - *Chris Woodruff*	**VSW14** Entity Framework for Enterprise Applications - *Benjamin Day*	**VSW15** The Mystical World of I/O Bindings in Azure Functions - *Ashley Grant*	**VSW16** Signing Your Code the Easy Way - *Oren Novotny*
7:30 PM	9:00 PM	Live! 360 Dessert Luau - *Wantilan Pavilion*			
Visual Studio Live! Day 3: Thursday, December 6, 2018					
8:00 AM	9:15 AM	**VSH01** Encrypting the Web - *Robert Boedigheimer*	**VSH02** Learning The Language Of HTTP For A Better Data Experience In Your Mobile Apps - *Chris Woodruff*	**VSH03** How to Interview a Developer - *Billy Hollis*	**VSH04** Creating a Release Pipeline with Team Services - *Esteban Garcia*
9:30 AM	10:45 AM	**VSH05** Advanced Fiddler Techniques - *Robert Boedigheimer*	**VSH06** Making XAML Do Things You Didn't Realize It Could - *Billy Hollis*	**VSH07** What Developers Want - *Rabeb Othmani*	**VSH08** To Be Announced
11:00 AM	12:00 PM	**VISUAL STUDIO LIVE! PANEL DISCUSSION: What Matters Most for the future of Modern Apps: AI, Data, Security, or UX?** *Brian Randell (Moderater), Billy Hollis, Tanya Janca, Oren Novotny, & Rabeb Othmani*			
12:00 PM	1:00 PM	Lunch on the Lanai - *Lanai / Pacifica 7*			
1:00 PM	2:15 PM	**VSH09** Get Func-y: Understanding Delegates in .NET - *Jeremy Clark*	**VSH10** Non-Useless Unit Testing Entity Framework & ASP.NET MVC - *Benjamin Day*	**VSH11** Modernizing the Enterprise Desktop Application - *Oren Novotny*	**VSH12** DevOps: A Catalyst for Enterprise Agility - *Heidi Araya & Esteban Garcia*
2:30 PM	3:45 PM	**VSH13** IEnumerable, ISaveable, IDontGetIt: Understanding .NET Interfaces - *Jeremy Clark*	**VSH14** Finding Your Place in the Cosmos: When and Why You Should Consider Azure Cosmos DB - *Eric Potter*	**VSH15** Pushing Left Like a Boss: Application Security Foundation - *Tanya Janca*	**VSH16** Scrum, Kanban, or Scrumban? - *Heidi Araya*
4:00 PM	5:00 PM	Next? Live! 360 Networking Event *Heidi Araya, Andrew Brust, Jeremy Clark, Ben Curry, Benjamin Day, Peter De Tender, Brent Edwards, Kevin Ford, Grant Fritchey, Esteban Garcia, Seth Juarez, Dave Kawula, Cristal Kawula, Sami Laiho, Thomas LaRock, Vishwas Lele, Rockford Lhotka, Karen Lopez, Matthew McDermott, Brian Randell, Paul Schaeflein, Jen Underwood, Rob Windsor*			
Visual Studio Live! Post-Conference Workshops: Friday, December 7, 2018					
8:00 AM	05:00 PM	**VSF01** Workshop: UX Design for Developers: Basics of Principles and Process - *Billy Hollis*	**VSF02** Workshop: Microservices with AKS (Managed Kubernetes) - *Vishwas Lele*		

Speakers and sessions subject to change

JAMES MCCAFFREY

Introduction to the ML.NET Library

The ML.NET library is an open source collection of machine learning (ML) code that can be used directly in .NET applications. Most ML libraries, such as TensorFlow, Keras, CNTK, and PyTorch, are written in Python and call into low-level C++ routines. However, if you use a Python-based library, it's not so easy for a .NET application to access a trained ML model. Fortunately, the ML.NET library integrates seamlessly into .NET applications.

The best way to see where this article is headed is to take a look at the demo program in **Figure 1**. The demo creates an ML model that predicts whether a patient will die or survive based on the patient's age, sex, and score on a kidney medical test. Because there are only two possible outcomes, die or survive, this is a binary classification problem.

Behind the scenes, the demo program uses the ML.NET library to create and train a logistic regression model. As I'm writing this article, the ML.NET library is still in preview mode, so some of the information presented here may have changed by the time you're reading this.

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

Begin ML.NET (v0.3.0 preview) demo run
Predict die/survive based on patient age, sex, kidney test

Starting training

Automatically adding a MinMax normalization transform, use 'norm=Warn' or
'norm=No' to turn this behavior off.
LBFGS multi-threading will attempt to load dataset into memory. In case of
 out-of-memory issues, add 'numThreads=1' to the trainer arguments and 'ca
che=-' to the command line arguments to turn off multi-threading.
Warning: Too few instances to use 12 threads, decreasing to 1 thread(s)
Beginning optimization
num vars: 4
improvement criterion: Mean Improvement
L1 regularization selected 4 of 4 weights.
Not training a calibrator because it is not needed.

Training complete

Model accuracy = 66.67%
Making prediction for 50-year old male with kidney = 4.80:
Prediction = survive

End ML.NET demo
```

Figure 1 **ML.NET Demo Program in Action**

The demo uses a set of training data with 30 items. After the model was trained, it was applied to the source data, and achieved 66.67 percent accuracy (20 correct and 10 wrong). The demo concludes by using the trained model to predict the outcome for a 50-year-old male with a kidney test score of 4.80—the prediction is that the patient will survive.

This article assumes you have intermediate or better programming skill with C#, but doesn't assume you know anything about the ML.NET library. The complete code and data for the demo program are presented in this article and are also available in the accompanying file download.

The Demo Program

To create the demo program, I launched Visual Studio 2017. The ML.NET library will work with either the free Community Edition or any of the commercial editions of Visual Studio 2017. The ML.NET documentation doesn't explicitly state that Visual Studio 2017 is required, but I couldn't get the demo program to work with Visual Studio 2015. I created a new C# console application project and named it Kidney. The ML.NET library will work with either a classic .NET or a .NET Core application type.

After the template code loaded, I right-clicked on file Program.cs in the Solution Explorer window and renamed the file to KidneyProgram.cs and I allowed Visual Studio to automatically rename class Program for me. Next, in the Solution Explorer window, I right-clicked on the Kidney project and selected the Manage NuGet Packages option. In the NuGet window, I selected the Browse tab and then entered "ML.NET" in the Search field. The ML.NET library is housed in the Microsoft.ML package. I selected that package and clicked the Install button. After a few seconds Visual Studio responded with a "successfully installed Microsoft.ML 0.3.0 to Kidney" message.

At this point I did a Build | Rebuild Solution and got a "supports only x64 architectures" error message. In the Solution Explorer window, I right-clicked on the Kidney project, and selected the Properties entry. In the Properties window, I selected the Build tab on the left, and then changed the Platform Target entry from "Any CPU" to "x64." I also made sure I was targeting the 4.7 version of the .NET Framework. With earlier versions I got an error related to one of the math library dependencies and had to manually edit

Code download available at msdn.com/magazine/1118magcode.

Figure 2 **Kidney Data**

```
48, +1, 4.40, survive
60, -1, 7.89, die
51, -1, 3.48, survive
66, -1, 8.41, die
40, +1, 3.05, survive
44, +1, 4.56, survive
80, -1, 6.91, die
52, -1, 5.69, survive
56, -1, 4.01, survive
55, -1, 4.48, survive
72, +1, 5.97, survive
57, -1, 6.71, die
50, -1, 6.40, survive
80, -1, 6.67, die
69, +1, 5.79, survive
39, -1, 5.42, survive
68, -1, 7.61, die
47, +1, 3.24, survive
45, +1, 4.29, survive
79, +1, 7.44, die
44, -1, 2.55, survive
52, +1, 3.71, survive
55, +1, 5.56, die
76, -1, 7.80, die
51, -1, 5.94, survive
46, +1, 5.52, survive
48, -1, 3.25, survive
58, +1, 4.71, survive
44, +1, 2.52, survive
68, -1, 8.38, die
```

the global .csproj file. Ugh. Then I did a Build | Rebuild Solution and was successful. When working with preview-mode libraries such as ML.NET, you should expect glitches like this to be the rule rather than the exception.

The Demo Data

After creating the skeleton of the demo program, the next step was to create the training data file. The data is presented in **Figure 2**. In the Solution Explorer window, I right-clicked on the Kidney project and selected Add | New Item. From the new item dialog window, I selected the Text File type and named it KidneyData.txt. If you're following along, copy the data from **Figure 2** and paste it into the editor window, being careful not to have any extra trailing blank lines.

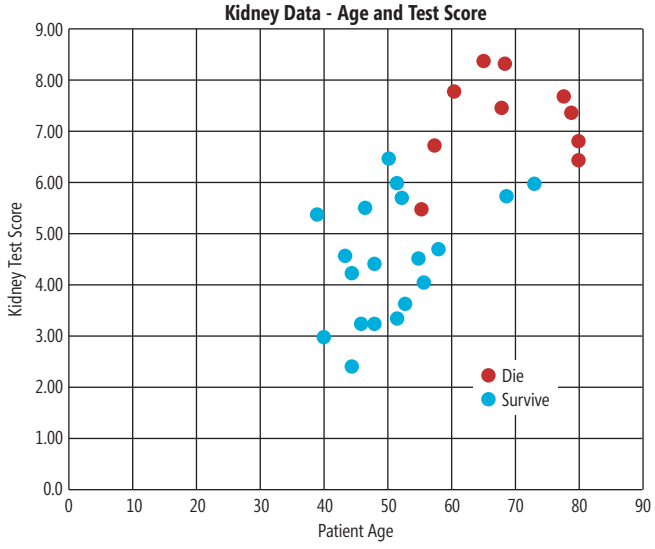


Figure 3 **Kidney Data**

The 30-item dataset is artificial and should be self-explanatory for the most part. The sex field is encoded as male = -1 and female = +1. Because the data has three dimensions (age, sex, test score), it's not possible to display it in a two-dimensional graph. But you can get a good idea of the structure of the data by examining the graph of just age and kidney test score in **Figure 3**. The graph suggests that the data may be linearly separable.

The Program Code

The complete demo code, with a few minor edits to save space, is presented in **Figure 4**. At the top of the Editor window, I removed all namespace references and replaced them with the ones shown in the code listing. The various Microsoft.ML namespaces house all ML.NET functionality. The System.Threading.Tasks namespace is needed to save or load a trained ML.NET model to file.

The demo program defines a class named KidneyData, nested inside the main program class, that defines the internal structure of the training data. For example, the first column is:

```
[Column(ordinal: "0", name: "Age")]
public float Age;
```

> You can think of the pipeline object as an untrained ML model plus the data needed to train the model.

Notice that the age field is declared type float rather than type double. In ML, type float is the default numeric type because the increase in precision you get from using type double is almost never worth the resulting memory and performance penalty. The value-to-predict must use the name "Label," but predictor field names can be whatever you like.

The demo program defines a nested class named KidneyPrediction to hold model predictions:

```
public class KidneyPrediction
{
  [ColumnName("PredictedLabel")]
  public string PredictedLabels;
}
```

The column name "PredictedLabel" is required but, as shown, the associated string identifier doesn't have to match.

Creating and Training the Model

The demo program creates an ML model using these seven statements:

```
var pipeline = new LearningPipeline();
string dataPath = "..\\..\\KidneyData.txt";
pipeline.Add(new TextLoader(dataPath).
  CreateFrom<KidneyData>(separator: ','));
pipeline.Add(new Dictionarizer("Label"));
pipeline.Add(new ColumnConcatenator("Features", "Age", "Sex", "Kidney"));
pipeline.Add(new LogisticRegressionBinaryClassifier());
pipeline.Add(new PredictedLabelColumnOriginalValueConverter()
  { PredictedLabelColumn = "PredictedLabel" });
```

You can think of the pipeline object as an untrained ML model plus the data needed to train the model. Recall that the values-to-predict in the data file are either "survive" or "die." Because

LIVE! 360
TECH EVENTS WITH PERSPECTIVE

TECHMENTOR

IN-DEPTH TRAINING FOR IT PROS

Training Conference for IT Pros

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

Where IT Training Meets the Sunshine

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

TRACK TOPICS INCLUDE:

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

REGISTER NOW

LAST CHANCE FOR SAVINGS!
REGISTER BY NOVEMBER 2 AND SAVE $300!

Use Promo Code MSDN

A Part of Live! 360: The Ultimate Education Destination

6 CONFERENCES, 1 GREAT PRICE

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

TECHMENTOREVENTS.COM/ORLANDO

EVENT PARTNERS
Microsoft
Magenic

GOLD SPONSOR
SENTRY

SILVER SPONSORS
Progress
GOVERLAN REACH

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

PRODUCED BY
CONVERGE 360
An 1105 MEDIA company

AGENDA-AT-A-GLANCE

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

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

Speakers and sessions subject to change

ML models only understand numeric values, the weirdly named Dictionarizer class is used to encode two strings to 0 or 1. The ColumnConcatenator constructor combines the three predictor variables into an aggregate; using a string result parameter with the name of "Features" is required.

There are many different ML techniques you can use for a binary classification problem. I use logistic regression in the demo program to keep the main ideas as clear as possible because it's arguably the simplest and most basic form of ML. Other binary classifier algorithms supported by ML.NET include AveragedPerceptronBinaryClassifier, FastForestBinaryClassifier and LightGbmClassifier.

The demo program trains and saves the model using these statements:

```
var model = pipeline.Train<KidneyData, KidneyPrediction>();
string ModelPath = "..\\..\\KidneyModel.zip";
Task.Run(async () =>
{
  await model.WriteAsync(ModelPath);
}).GetAwaiter().GetResult();
```

The ML.NET library Train method is very sophisticated. If you refer back to the screenshot in **Figure 1**, you can see that Train performs automatic normalization of predictor variables, which scales them so that large predictor values, such as a person's annual income, don't swamp small values, such as a person's number of children. The Train method also uses regularization, which is an advanced technique to improve the accuracy of a model. In short, ML.NET does all kinds of advanced processing, without you having to explicitly configure parameter values.

Saving and Evaluating the Model

After the model has been trained, it's saved to disk like so:

```
string ModelPath = "..\\..\\KidneyModel.zip";
Task.Run(async () =>
{
  await model.WriteAsync(ModelPath);
}).GetAwaiter().GetResult();
```

Because the WriteAsync method is asynchronous, it's not so easy to call it. The approach I prefer is the wrapper technique shown. The lack of a non-async method to save an ML.NET model is a bit surprising, even for a library that's in preview mode.

The demo program makes the assumption that the program executable is two directories below the project root directory. In a production system you'd want to verify that the target directory exists. Typically, in an ML.NET project, I like to create a Data subdirectory and a Models subdirectory off the project root folder (Kidney in this example) and save my data and models in those directories.

The model is evaluated with these statements:

```
var testData = new TextLoader(dataPath).
  CreateFrom<KidneyData>(separator: ',');
var evaluator = new BinaryClassificationEvaluator();
var metrics = evaluator.Evaluate(model, testData);
double acc = metrics.Accuracy * 100;
Console.WriteLine("Model accuracy = " +
  acc.ToString("F2") + "%");
```

In most ML scenarios you'd have two data files—one set used just for training and a second test dataset used only for model evaluation. For simplicity, the demo program reuses the single 30-item data file for model evaluation.

Figure 4 **ML.NET Example Program**

```
using System;
using Microsoft.ML;
using Microsoft.ML.Data;
using Microsoft.ML.Runtime.Api;
using Microsoft.ML.Trainers;
using Microsoft.ML.Transforms;
using Microsoft.ML.Models;
using System.Threading.Tasks;

namespace Kidney
{
  class KidneyProgram
  {

    public class KidneyData
    {
      [Column(ordinal: "0", name: "Age")]
      public float Age;
      [Column(ordinal: "1", name: "Sex")]
      public float Sex;
      [Column(ordinal: "2", name: "Kidney")]
      public float Kidney;
      [Column(ordinal: "3", name: "Label")]
      public string Label;
    }

    public class KidneyPrediction
    {
      [ColumnName("PredictedLabel")]
      public string PredictedLabels;
    }

    static void Main(string[] args)
    {
      Console.WriteLine("ML.NET (v0.3.0 preview) demo run");
      Console.WriteLine("Survival based on age, sex, kidney");
      var pipeline = new LearningPipeline();

      string dataPath = "..\\..\\KidneyData.txt";
      pipeline.Add(new TextLoader(dataPath).
        CreateFrom<KidneyData>(separator: ','));
      pipeline.Add(new Dictionarizer("Label"));
      pipeline.Add(new ColumnConcatenator("Features", "Age",
        "Sex", "Kidney"));
      pipeline.Add(new LogisticRegressionBinaryClassifier());
      pipeline.Add(new
        PredictedLabelColumnOriginalValueConverter()
        { PredictedLabelColumn = "PredictedLabel" });
      Console.WriteLine("\nStarting training \n");
      var model = pipeline.Train<KidneyData,
        KidneyPrediction>();
      Console.WriteLine("\nTraining complete \n");

      string ModelPath = "..\\..\\KidneyModel.zip";
      Task.Run(async () =>
      {
        await model.WriteAsync(ModelPath);
      }).GetAwaiter().GetResult();

      var testData = new TextLoader(dataPath).
        CreateFrom<KidneyData>(separator: ',');
      var evaluator = new BinaryClassificationEvaluator();
      var metrics = evaluator.Evaluate(model, testData);
      double acc = metrics.Accuracy * 100;
      Console.WriteLine("Model accuracy = " +
        acc.ToString("F2") + "%");

      Console.WriteLine("Predict 50-year male, kidney 4.80:");
      KidneyData newPatient = new KidneyData()
        { Age = 50f, Sex = -1f, Kidney = 4.80f };
      KidneyPrediction prediction = model.Predict(newPatient);
      string result = prediction.PredictedLabels;
      Console.WriteLine("Prediction = " + result);

      Console.WriteLine("\nEnd ML.NET demo");
      Console.ReadLine();
    } // Main
  } // Program
} // ns
```

The Evaluate method returns an object containing several metrics, including log loss, precision, recall, F1 score and so on. The return object also has a neat ConfusionMatrix object that can be used to display counts such as the number of patients who were predicted to survive but in fact died.

Using the Trained Model

The demo program shows how to use the trained model to make a prediction:

```
Console.WriteLine("Predict 50-year male kidney = 4.80:");
KidneyData newPatient = new KidneyData()
  { Age = 50f, Sex = -1f, Kidney = 4.80f };
KidneyPrediction prediction = model.Predict(newPatient);
string result = prediction.PredictedLabels;
Console.WriteLine("Prediction = " + result);
```

> In most ML scenarios you'd have two data files—one set used just for training and a second test dataset used only for model evaluation.

Notice that the numeric literals for age, sex and kidney score use the "f" modifier because the model is expecting type float values. In this example, the trained model was available because the program just finished training. If you wanted to make a prediction from a different program, you'd load the trained model using the ReadAsync method along the lines of:

```
PredictionModel<KidneyData,
  KidneyPrediction> model = null;
Task.Run(async () =>
{
  model2 = await PredictionModel.ReadAsync
  <KidneyData, KidneyPrediction>(ModelPath);
}).GetAwaiter().GetResult();
```

Wrapping Up

Even though the ML.NET library is new, its origins go back many years. Shortly after the introduction of the Microsoft .NET Framework in 2002, Microsoft Research began a project called "text mining search and navigation", or TMSN, to enable software developers to include ML code in Microsoft products and technologies. The project was very successful, and over the years grew in size and usage internally at Microsoft. Somewhere around 2011 the library was renamed to "the learning code" (TLC). TLC is widely used within Microsoft and is currently in version 3.9. The ML.NET library is a direct offshoot of TLC, with Microsoft-specific features removed. ■

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

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

msdn magazine

STATEMENT OF OWNERSHIP, MANAGEMENT AND CIRCULATION

1. Publication Title: MSDN Magazine
2. Publication Number: 1528-4859
3. Filing Date: September 30, 2018
4. Frequency of Issue: Monthly with a special issue in November
5. Number of Issues Published Annually: 13
6. Annual Subscription Price: US $35, International $60
7. Complete Mailing Address of Known Office of Publication: 6300 Canoga Ave., Ste. 1150, Woodland Hills, CA 91367
8. Complete Mailing Address of the Headquarters of General Business Offices of the Publisher: Same as above.
9. Full Name and Complete Mailing Address of Publisher, Editor, and Managing Editor: Dan LaBianca, Chief Revenue Officer, 14901 Quorum Drive, Ste 425, Dallas, TX 75254
   Michael Desmond, Editor-in-Chief, 8251 Greensboro Drive, Suite 510, McLean, VA 22102
   Wendy Hernandez, Group Managing Editor, 4 Venture, Suite 150, Irvine, CA 92618
10. Owner(s): 1105 Media, Inc. dba: 101 Communications LLC, 6300 Canoga Ave., Ste. 1150, Woodland Hills, CA 91367 Listing of shareholders in 1105 Media, Inc.
11. Known Bondholders, Mortgagees, and Other Security Holders Owning or Holding 1 Percent or more of the Total Amount of Bonds, Mortgages or Other Securities:
    Nautic Partners V, L.P., 50 Kennedy Plaza, 12th Flr., Providence, RI 02903
    Kennedy Plaza Partners III, LLC, 50 Kennedy Plaza, 12th Flr., Providence, RI 02903
    Alta Communications IX, L.P., 1000 Winter Street, South Entrance, Suite 3500, Waltham, MA 02451
    Alta Communications IX, B-L.P., 1000 Winter Street, South Entrance, Suite 3500, Waltham, MA 02451
    Alta Communications IX, Associates LLC, 1000 Winter Street, South Entrance, Suite 3500, Waltham, MA 02451
12. The tax status has not changed during the preceding 12 months.
13. Publication Title: MSDN Magazine
14. Issue date for Circulation Data Below: September 2018
15. Extent & Nature of Circulation:

	Average No. Copies Each Month During Preceding 12 Months	No. Copies of Single Issue Published Nearest to Filing Date
a. Total Number of Copies (Net Press Run)	80,917	86,516
b. Legitimate Paid/and or Requested Distribution		
1. Outside County Paid/Requested Mail Subscriptions Stated on PS Form 3541	60,355	53,498
2. In-County Paid/Requested Mail Subscriptions Stated on PS Form 3541	0	0
3. Sales Through Dealers and Carriers, Street Vendors, Counter Sales, and Other Paid or Requested Distribution Outside USPS®	1,328	797
4. Requested Copies Distributed by Other Mail Classes Through the USPS®	0	0
c. Total Paid and/or Requested Circulation	61,683	54,295
Nonrequested Distribution		
1. Outside County Nonrequested Copies Stated on PS Form 3541	18,195	25,405
2. In-County Nonrequested Copies Distribution Stated on PS Form 3541	0	0
3. Nonrequested Copies Distribution Through the USPS by Other Classes of Mail	0	0
4. Nonrequested Copies Distributed Outside the Mail	929	6,711
e. Total Nonrequested Distribution	19,124	32,116
f. Total Distribution	80,807	86,411
g. Copies not Distributed	110	105
h. Total	80,917	86,411i. Percent
paid and/or Requested Circulation	76.33%	62.83%

16. Electronic Copy Circulation
    a. Requested and Paid Electronic Copies
    b. Total Requested and Paid Print Copies (Line 15c) + Requested/Paid Electronic Copies
    c. Total Requested Copy Distribution (Line15f) + Requested/Paid Electronic Copies (Line 16a)
    d. Percent Paid and/or Requested Circulation (Both print & Electronic Copies) (16b divided by 16c x 100)
    ☑ **I certify that 50% of all my distributed copies (electronic and paid print are legitimate request or paid copies.**
17. Publication of Statement of Ownership for a Requester Publication is required and will be printed in the November 2018 issue of this publication.
18. I certify that all information furnished on this form is true and complete:
    Peter B. Weller, Manager, Print Production

LIVE! 360
TECH EVENTS WITH PERSPECTIVE

SQL Server LIVE!

TRAINING FOR DBAs AND IT PROS

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

Data. Driven.

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

TRACK TOPICS INCLUDE:

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

REGISTER NOW

LAST CHANCE FOR SAVINGS!
REGISTER BY NOVEMBER 2
AND SAVE $300!

Use Promo Code MSDN

A Part of Live! 360: The Ultimate Education Destination

6 CONFERENCES, 1 GREAT PRICE

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

SQLLIVE360.COM

EVENT PARTNERS
Microsoft
Magenic

GOLD SPONSOR
SENTRY

SILVER SPONSORS
Progress
GOVERLAN REACH

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

PRODUCED BY
CONVERGE 360
An 1105 MEDIA company

AGENDA-AT-A-GLANCE

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

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

Speakers and sessions subject to change

For Whom The Bell Tolls

I've been seeing more discussion in the media these days about employees questioning the morality of the projects on which they're working. Are they enabling repressive political regimes? Poisoning the planet? Hoodwinking uneducated customers, especially children? And so on.

We deal with this dilemma in every job that we consider. I remember a job interview in engineering school, where a company asked if I had reservations about working on nuclear weapons. The interviewer said, "Other people decide what we need, we just build them." My editor won't let me quote my exact reply, but an accurate paraphrase is: "I doubt that argument would save my neck at my war crimes trial." They didn't offer me the job, which is probably just as well.

Here in politically liberal Massachusetts, it's sometimes hard to hire people for military projects. I once worked for a company that made chaff decoys for warships—as defensive a tool as you can find. "Chaff decoys don't blow people up," I said. My cynical-but-realistic friends riposted, "No, destroyers shielded by chaff decoys blow people up." True, but I clung to the fig leaf that let me sleep at night.

I once met a guy at a TechEd conference whose company made cigarettes. Few products today are more reviled. Even state-legal marijuana enjoys a certain plucky underdog image, aided partly by Microsoft (see my December 2016 column at msdn.com/magazine/mt763241). "What's the job like?" I asked. "About like any other job, I suppose. By state law, there's no smoking in the office. It's kinda funny actually, seeing everyone go outside to smoke. But the product is legal, and prohibition would be worse. I see it as a Darwin award generator, you know? Improving the human species by killing off the dumb ones?"

Playing the cynical realist, I explained that to earn a Darwin, he'd have to kill off smokers before they reproduced. Could he make cigarettes deadlier? I pointed him to Christopher Buckley's 1994 satirical novel "Thank You for Smoking," and the 2005 film made from it. I probably would have cashed that company's check, had they offered me one. But I'd have a hard time explaining it to my daughters.

What happens when your company produces something generic, say, chain saws? These can obviously be used for good or evil, depending on the intent of the wielder. There are something like a billion Windows PCs in the world. It's a pretty good bet that at any hour of the day or night, somebody somewhere is using one for a purpose Microsoft would detest if the company knew about it. I've taught more than 1,000 students at Harvard over the years, and more at companies around the world. It's a statistical certainty that some of them are using the knowledge I gave them for purposes I'd consider evil. Both Microsoft and I manage to muddle through by hoping there's a lot more of the good. Maybe it's time to restart my campaign for a geek's version of the Hippocratic oath, as I wrote in my September 2012 column (see msdn.com/magazine/jj618306).

In my travels as an itinerant consultant, a week here, a month there, I've worked with hundreds of companies. Every employee at every one of them has concocted a rationalization as to why they took that job, a way of thrusting down the bad stuff and concentrating on the good stuff. You have to do that with any career—in fact, with all of life—or you'd never get anywhere. Of course, by definition, the ones who couldn't do that at a company wouldn't be the employees I met there. Guys who can stomach weapons but not cigarettes work at Raytheon, guys with the opposite gastric preferences at Philip Morris.

What happens when your company produces something generic, say, chain saws? These can obviously be used for good or evil, depending on the intent of the wielder.

I think about people whose altruistic work I admire. Gisli Olafsson, flying into Liberia at the height of the 2014-2016 Ebola outbreak to bolster the IT of the relief effort (msdn.com/magazine/dn818503). Owen Walker, defending guilty criminals as a way of defending the U.S. Constitution (January 2017, msdn.com/magazine/mt791803). Atul Gawande, who inspired me to write "Why Software Sucks." And I ask myself what I'm doing. A thousand or so Harvard students is what I have to show for it, and about the same number of corporate students. And you, dear readers, as my grandmother used to say, "All you can do is all you can do." Would she think I've done enough? ■

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

Spreadsheets Everywhere.

SpreadsheetGear 2017 Released

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

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

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

Scalable Reporting

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

WIN Windows Forms | Silverlight | WPF

Powerful Controls

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

Comprehensive Charting

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

Fastest Calculations

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

Download your free fully functional evaluation at SpreadsheetGear.com

SpreadsheetGear

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