THE MICROSOFT JOURNAL FOR DEVELOPERS

DECEMBER 2015 VOL 30 NO 13

msdn magazine

THE MICROSOFT JOURNAL FOR DEVELOPERS

DECEMBER 2015 VOL 30 NO 13

msdn magazine

COLUMNS

Microsoft

XLS
DOC
PDF

msdn magazine

DECEMBER 2015 VOLUME 30 NUMBER 13

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

ENTERPRISE COMPUTING GROUP

President
Henry Allain

Chief Revenue Officer
Dan LaBianca

Chief Marketing Officer
Carmel McDonagh

ART STAFF
Creative Director **Jeffrey Langkau**
Associate Creative Director **Scott Rovin**
Senior Art Director **Deirdre Hoffman**
Art Director **Michele Singh**
Assistant Art Director **Dragutin Cvijanovic**
Graphic Designer **Erin Horlacher**
Senior Graphic Designer **Alan Tao**
Senior Web Designer **Martin Peace**

PRODUCTION STAFF
Director, Print Production **David Seymour**
Print Production Coordinator **Anna Lyn Bayaua**

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

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

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

MARKETING
Chief Marketing Officer **Carmel McDonagh**
Vice President, Marketing **Emily Jacobs**
Senior Manager, Marketing **Christopher Morales**
Marketing Coordinator **Alicia Chew**
Marketing & Editorial Assistant **Dana Friedman**

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

1105 MEDIA INC
YOUR GROWTH. OUR BUSINESS.

Chief Executive Officer
Rajeev Kapur

Chief Operating Officer
Henry Allain

Senior Vice President & Chief Financial Officer
Richard Vitale

Executive Vice President
Michael J. Valenti

Vice President, Information Technology & Application Development
Erik A. Lindgren

Chairman of the Board
Jeffrey S. Klein

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

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

CORPORATE ADDRESS 1105 Media, 9201 Oakdale Ave. Ste 101, Chatsworth, CA 91311 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.magreprints.com/QuickQuote.asp

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

Reaching the Staff

Staff may be reached via e-mail, telephone, fax, or mail. A list of editors and contact information is also available online at Redmondmag.com.

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

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

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

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

Microsoft

BPA WORLDWIDE

1906 100 2006 AMERICAN BUSINESS MEDIA
A Century Moving Business Forward

Western Publications Association

LEADTOOLS®

THE WORLD LEADER IN IMAGING SDKs

```
// Create an OCR Engine for each processor on the machine. This
// allows for optimal use of thread during recognition and processing.
ocrEngines = new List<IOcrEngine>();
for (int i = 0; i < Environment.ProcessorCount; i++)
{

ocrEngines.Add(OcrEngineManager.CreateEngine(
```

MILLIONS OF LINES OF CODE AT YOUR FINGERTIPS

Document Imaging	**Medical** Imaging	**Multimedia** Imaging
Document Viewer	*DICOM, CCOW & HL7*	*Play, Capture, Convert & DVR*
Document Converter	*PACS Client & Server Framework*	*Stream - MPEG2-TS & RTSP*
OCR, MICR, OMR & ICR	*DICOM Storage Server Framework*	*Media Streaming Server*
Barcode & Forms Recognition	*Web & Desktop Viewers*	*MPEG-4, H.264 & H.265*
Create, Save, Edit & View PDF	*Image Processing & Annotations*	*Media Foundation & DirectShow*
Annotations, TWAIN & SVG	*Medical 3D (MPR, MIP, VRT)*	*Distributed Transcoding*

.NET Windows API WinRT Linux iOS OS X Android HTML5

DOWNLOAD OUR 60 DAY EVALUATION
WWW.LEADTOOLS.COM

LEAD TECHNOLOGIES INCORPORATED

SALES@LEADTOOLS.COM
800.637.1840

EDITOR'S NOTE

MICHAEL DESMOND

Goodbye, Kenny

They say things happen in threes. That's been the case here at *MSDN Magazine*, where we've seen three big changes to our columnist lineup over the past month.

Things got started in November, when we welcomed Mark Michaelis on board as author of our new Essential .NET column, which focuses on C# and .NET Framework development issues and challenges. Then we learned that Modern Apps columnist Rachel Appel was returning to work at Microsoft and would no longer have time to write her column. Frank La Vigne—a Microsoft technical evangelist and an occasional *MSDN Magazine* author in his own right—will be helming Modern Apps starting in the January issue.

The third shakeup in our trifecta? Kenny Kerr, author of the long-running Windows with C++ column and a guy who has been working with *MSDN Magazine* going back to 2004, is ending his tenure as a columnist. Like Appel, Kerr has accepted a position at Microsoft, joining the Windows Engineering team to help produce a new Windows Runtime language projection for Standard C++.

Kerr has been manning an important flank here at *MSDN Magazine*, providing expert coverage of C++ programming. While most of our readership is committed to C# development (fully 70 percent identified C# as their primary programming language in our latest reader poll), C++ remains the primary language of 10 percent of *MSDN Magazine* subscribers. That makes it the second-most engaged programming language among readers.

Over the years, Kerr has written some of the most-widely read columns in the magazine. His piece, "The Evolution of Threads and I/O in Windows" (msdn.com/magazine/jj883951), in January 2013 was the fifth-most-viewed column we've published over the past five years, based on first-month traffic. Two other columns, "A Modern C++ Library for DirectX Programming" (msdn.com/magazine/dn201741) and "Using Regular Expressions with Modern C++" (msdn.com/magazine/dn519920), placed 12th and 14th, respectively, out of more than 450 total published columns over the time period. Not bad for a column that should appeal to, at most, 10 percent of readers.

Kerr's favorite *MSDN Magazine* article, however, isn't one of his columns. It's the feature he wrote for the *MSDN Magazine* 2014 special issue on Visual Studio 2015 and Microsoft Azure, "Visual C++ 2015 Brings Modern C++ to the Windows API" (msdn.com/magazine/dn879357). Based on the Implements class template he wrote for Microsoft, it's Kerr's single-most-visited article going back to 2009.

Kerr has been manning an important flank here at *MSDN Magazine*, providing expert coverage of C++ programming.

"Implements is a variadic class template that implements IUnknown and IInspectable entirely with modern C++ metaprogramming techniques, rather than traditional macro-heavy approaches popularized by ATL," Kerr says. "It was the first time I had written anything useful with variadic templates and proved to be quite novel and very effective. It is coincidentally one of the foundational metaprogramming constructs in Modern C++ for the Windows Runtime."

Can you tell that we're going to miss Kerr around here? There is a bit of good news to report: Kerr plans to return to our pages, writing the occasional C++-themed feature. And as Kerr puts it, our readers aren't the only ones who benefit from his articles.

"There is simply no better way for me to truly understand some technology, language or technique than to have to try to explain it to someone else," Kerr says. "I hope developers have benefited from my content, but I suspect I benefited the most."

On the contrary. We've been fortunate to have you here for so long.

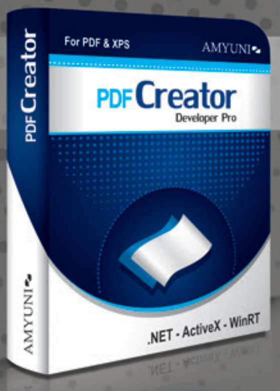
Visit us at msdn.microsoft.com/magazine. Questions, comments or suggestions for *MSDN Magazine*? Send them to the editor: mmeditor@microsoft.com.

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

Switch to Amyuni PDF

AMYUNI

NEW v5.5



Create and Edit PDFs in .NET, COM/ActiveX, WinRT & UWP

- Edit, process and print PDF 1.7 documents
- Create, fill-out and annotate PDF forms
- Fast and lightweight 32- and 64-bit components for .NET and ActiveX/COM
- New Universal Apps DLLs enable publishing C#, C++, CX or Javascript apps to windows Store
- Updated Postscript/EPS to PDF conversion module

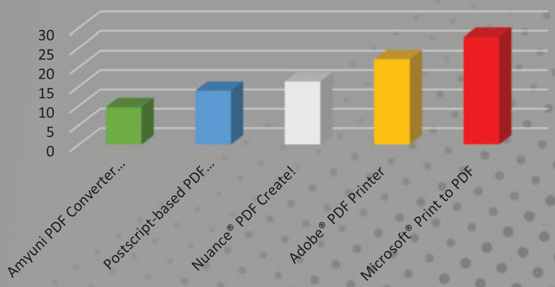
Complete Suite of Accurate PDF Components

- All your PDF processing, conversion and editing in a single package
- Combines Amyuni PDF Converter and PDF Creator for easy licensing, integration and deployment.
- Includes our Microsoft WHQL certified PDF Converter printer driver
- Export PDF documents into other formats such as Jpeg, PNG, XAML or HTML5
- Import and Export XPS files using any programming environment

High Performance PDF Printer for Desktops and Servers

- Print to PDF in a fraction of the time needed with other tools. WHQL tested for all Windows platforms. Version 5.5 updated for Windows 10 support

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



CERTIFIED FOR Windows Server® 2008

Windows Server 2012 Certified

Windows 7

Windows

Other Developer Components from Amyuni®

- WebkitPDF: Direct conversion of HTML files into PDF and XAML without the use of a web browser or a printer driver
- PDF2HTML5: Conversion of PDF to HTML5 including dynamic forms
- Postscript to PDF Library: For document workflow applications that require processing of Postscript documents
- OCR Module: Free add-on to PDF Creator uses the Tesseract engine for character recognition
- Javascript engine: Integrate a full Javascript interpreter into your applications to process PDF files or for any other need

All development tools available at

www.amyuni.com

AMYUNI
Technologies

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

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

The Long Game

Early in my career, I tried compulsively to find my "passion." At first, this passion was coding, until the novelty wore off and my responsibilities grew challenging. I interpreted this unpleasantness to mean my passion lay elsewhere. Over the coming years I switched roles, companies and industries repeatedly, sometimes back and forth. I read innumerable books on careers, took personality tests, spoke to career counselors and interviewed non-stop.

I'd start a new job, fall in love with it, conclude I'd finally found my passion, then (eventually) get bored and switch again.

In retrospect, I'd made discomfort a problem to be fixed, not a stage through which to pass. I realized that my repetitive cycles had only one common denominator (me), and no matter what I fixed, I was eventually unhappy again. I was "passion hunting." I'd imagined a career with no difficulties and a workplace of unending joy, and when reality conflicted with my fantasy, I'd bolt!

> I was "passion hunting." I'd imagined a career with no difficulties and a workplace of unending joy, and when reality conflicted with my fantasy, I'd bolt!

If I wanted a change in my environment to make me happy, I was avoiding taking full responsibility for my own future. For example, I believed that by switching to a start-up, I'd improve my social life because I'd have younger coworkers and develop more friendships. Alas, that was a way for me to avoid working on my own social fears. Instead of facing my character difficulties head-on, I was blaming the company I worked for!

Today, instead of seeking a hit from "doing what I love," I abide in the possibility that I'm already there. Finding my passion isn't a one-time event, it's not even a one-job or one-career event—it's a lifelong process of self development by "loving what I do."

I've given myself permission to have different careers and jobs over time. When I see my career choices as all-or-nothing, make-it-or-break-it moments, I burden myself emotionally and allow zero room for error. At work, I set a loftier goal not focused on myself—say, to serve my customers and coworkers—independent of whatever role or industry I'm in. If I make any changes along the way, I'm not accepting defeat; I'm just growing.

Enthusiasm Overload

In one of my first jobs, I quickly burned out from working 60- to 80-hour weeks. I worked myself to such numbness that I could recover only by quitting altogether. I wasted months in a haze of helpless unproductivity, as my body attempted to recover from the intensity of years of overwork.

I had defined myself by my own assumptions about what it meant to be a Real Programmer. I'd heard of start-up execs who wouldn't go home for days, and engineers who'd ignore their families, only to achieve grand IPOs years later. I saw anything less than such commitment as a failure.

Over time, I healed the burnout by not looking to work as my only savior. Today, I go to spiritual retreats, cultivate hobbies, take self-development classes, develop close relationships, journal, mentor, get mentored, spend time in nature, take naps, meditate, read and hike. Each of these fills a need that work by itself can't fill.

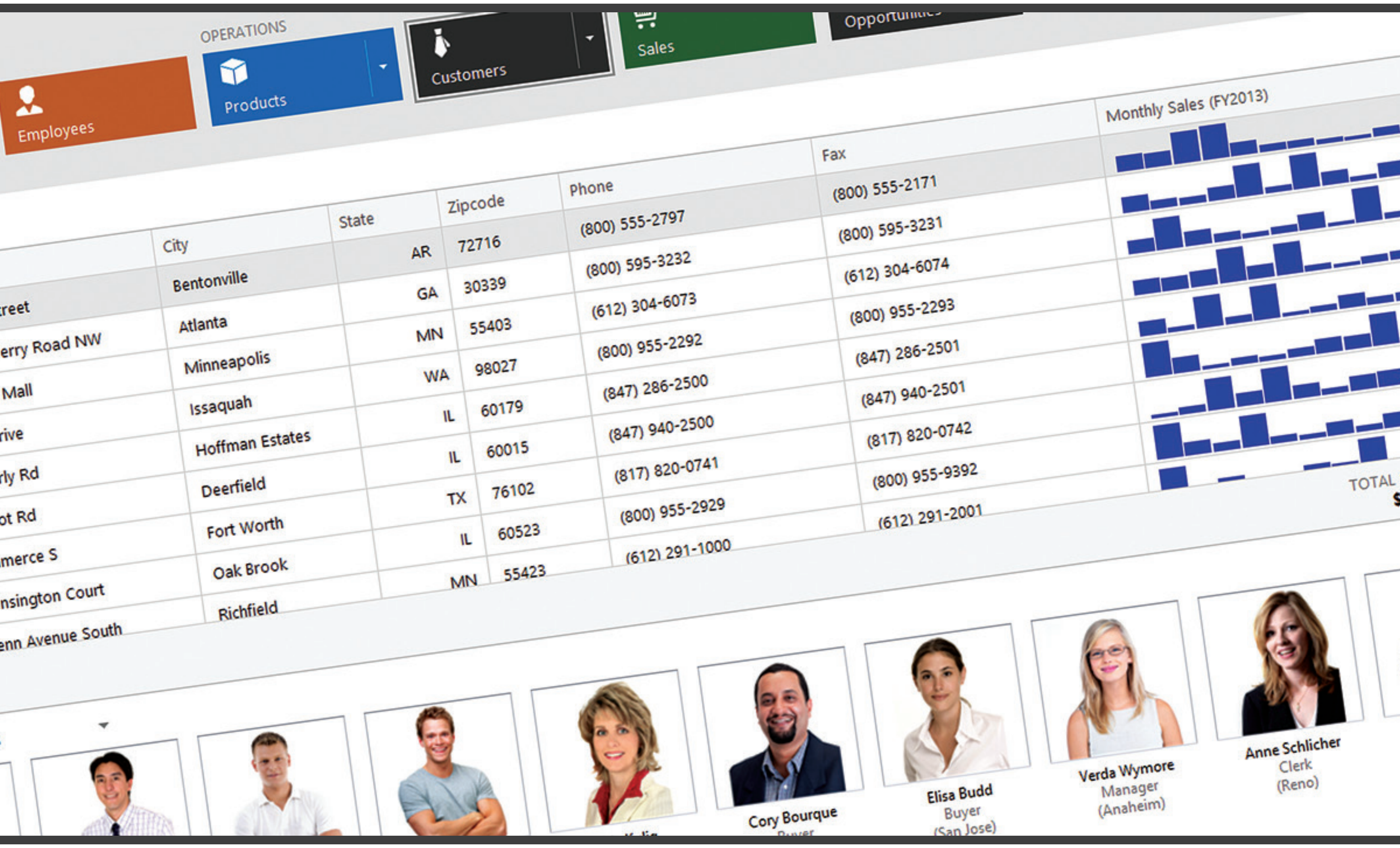
I also look to early indicators (for example, getting angry at loved ones, zoning out or feeling down) as reminders for me to take a "gentleness break." I initially felt self-conscious in taking care of myself—leaving work on time, not working on weekends or evenings, or removing work IM from my phone—because I was surrounded by so many coworkers who *didn't* engage in self-care.

As I've seen positive results—peace and joy, promotions and rewards, productivity—I realize I'm playing the *long* game, not just for work but for all areas of my life. When I take care of myself and avoid burnout, I'm just recharging so that I can make better, more consistent contributions—to work, family and friends—for the next six decades, not just for the next six months. ■

KRISHNAN RANGACHARI *is a career coach for the tech industry. He is one of the youngest-ever graduates of UC Berkeley. At age 18, he joined Microsoft full-time, and by age 20, he became the fastest-promoted employee in his division. Rangachari has worked in senior development and product roles for many tech firms. Visit radicalshifts.com to get his free career success kit.*

Touch-Optimized Hybrid Apps

DevExpress Universal provides a comprehensive collection of UX tools so you can create touch-first apps that are built for today and ready for tomorrow.



reet	City	State	Zipcode	Phone	Fax
erry Road NW	Bentonville	AR	72716	(800) 555-2797	(800) 555-2171
Mall	Atlanta	GA	30339	(800) 595-3232	(800) 595-3231
rive	Minneapolis	MN	55403	(612) 304-6073	(612) 304-6074
rly Rd	Issaquah	WA	98027	(800) 955-2292	(800) 955-2293
ot Rd	Hoffman Estates	IL	60179	(847) 286-2500	(847) 286-2501
merce S	Deerfield	IL	60015	(847) 940-2500	(847) 940-2501
nsington Court	Fort Worth	TX	76102	(817) 820-0741	(817) 820-0742
enn Avenue South	Oak Brook	IL	60523	(800) 955-2929	(800) 955-9392
	Richfield	MN	55423	(612) 291-1000	(612) 291-2001

Cory Bourque
Buyer

Elisa Budd
Buyer
(San Jose)

Verda Wymore
Manager
(Anaheim)

Anne Schlicher
Clerk
(Reno)

Your Next Great Hybrid App Starts Here

Build touch-first apps for any Windows device and leverage existing code investments.
devexpress.com/hybrid

JULIE LERMAN

Aurelia Meets DocumentDB: A Matchmaker's Journey, Part 2

The new JavaScript framework, Aurelia, and the NoSQL document database service, Azure DocumentDB, are two different technologies that have captured my interest lately. In June, I explored DocumentDB at a high level (msdn.com/magazine/mt147238). Then, in September, I played with data binding in Aurelia (msdn.com/magazine/mt422580). Still intrigued by both technologies, I thought I'd like to put the two together, using Aurelia as the front end of an application and DocumentDB as the data store. While my curiosity gave me some advantage (persistence), my limited experience with these two new technologies, and with JavaScript frameworks in general, set me along a number of dead-end attempts to combine them. This was exacerbated by the fact that nobody had done this combo before—at least not publicly. These failed attempts were documented in my November column (msdn.com/magazine/mt620011), the first of this two-part series. One obvious and easy path I chose to skip was to use my existing Web API that wrapped interaction with DocumentDB in place of the Web API I worked with in my first Aurelia article. I made this choice because there was no challenge in it—and therefore not much fun.

I finally came to the decision to use a different server-side solution: Node.js. There's an existing example that uses the DocumentDB Node.js SDK and a different front end. I inspected it carefully and shared what I learned with you in the second half of my November column. Then I set about recreating my Ninja example from the September column, this time using Node.js as my go-between from the Aurelia front end to DocumentDB. The stumbling blocks were many, and they went far beyond data-related issues. I had loads of support from members of the Aurelia core team and especially from Patrick Walters (github.com/pwkad), who not only helped ensure I was taking advantage of Aurelia features, but also indirectly forced me to further hone my Git skills. What I'll focus on here are the pieces of the solution that involve accessing, updating and binding the data. The ReadMe file that comes with the accompanying downloadable example explains how you can set up your own DocumentDB and import the data for this solution. It also explains how to set up the requirements to run the sample.

Code download available at msdn.com/magazine/msdnmag1215.

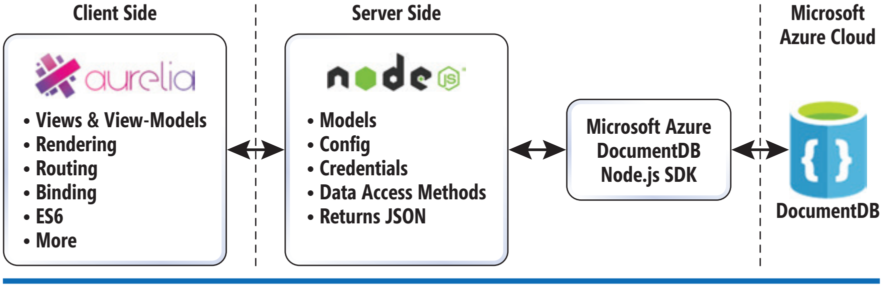


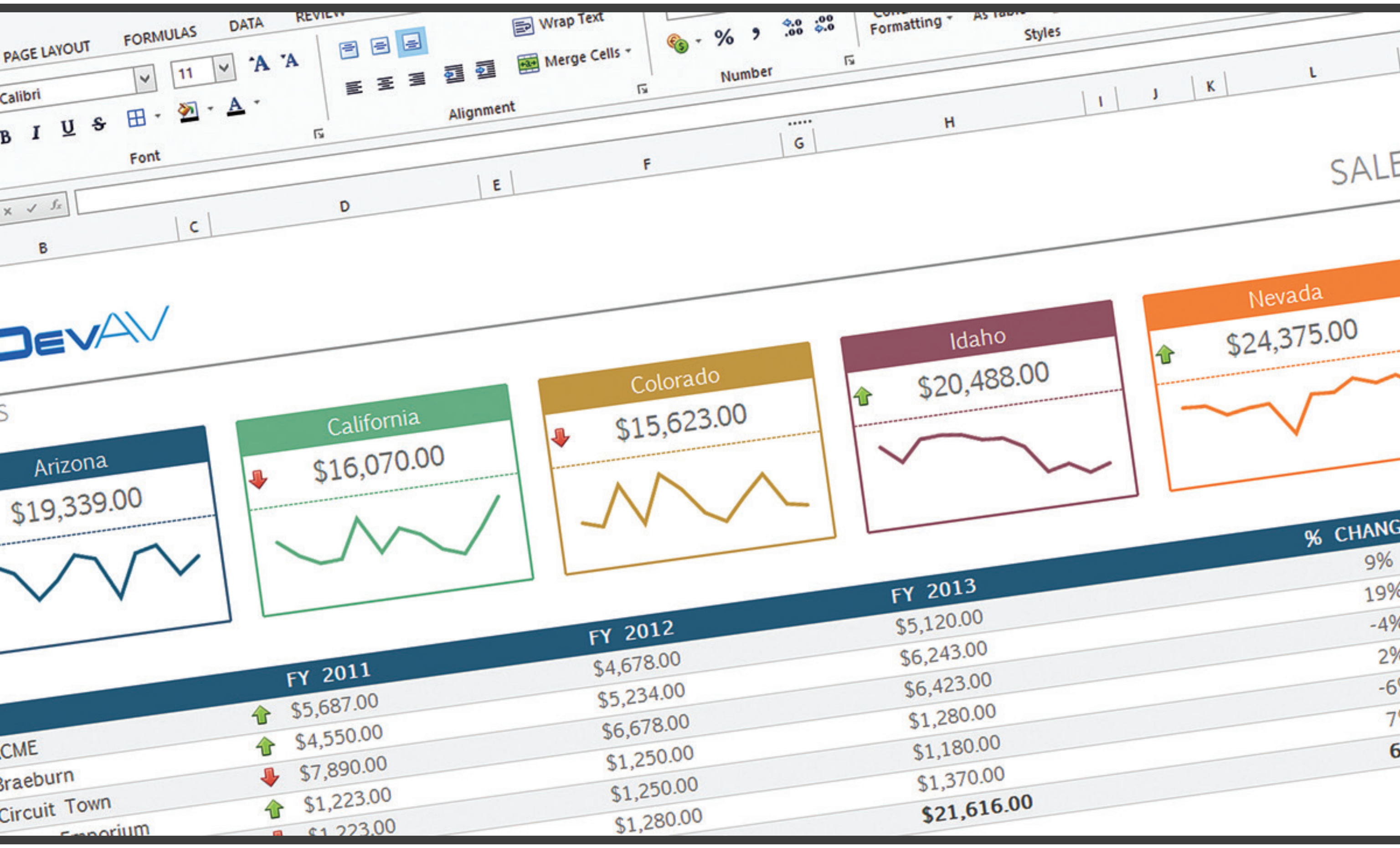
Figure 1 **The Overall Structure of This Solution**

Solution Architecture

First, let's take a look at the overall architecture of this solution, as shown in **Figure 1**.

Aurelia is a client-side framework so it's here only to handle client-side requirements, such as rendering HTML based on the view and view-model pairings, as well as routing, data binding and other relevant tasks. (Note that all of the client-side code can be debugged in developer tools available in browsers such as Internet Explorer and Chrome.) In the solution's path, all of the client-side code is in the public/app/src folder, while the server-side code is in the root folder. When the Web site is running, the client-side files are compiled by Aurelia to ensure browser-compatibility and distributed into a folder called "dist." These files are what get sent to the client.

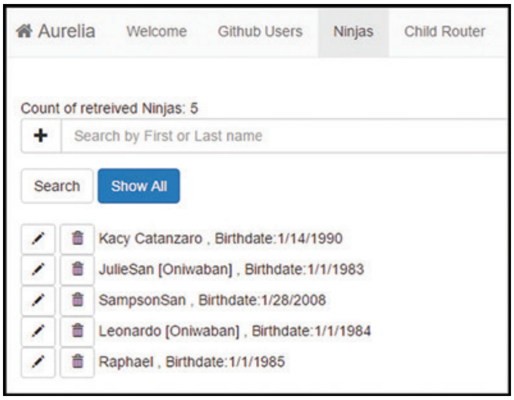
The server-side code should be familiar if you've done Web development in the Microsoft .NET Framework (closer to my own background). In the .NET environment, your codebehind from WebForms, controllers and other logic is traditionally compiled into DLLs that live on the server. Of course, much of this changes with ASP.NET 5 (which I now feel more prepared for, thanks to working on this particular project). My Aurelia project also contains server-side code, but even that is JavaScript code by way of Node.js. This code isn't compiled into a DLL but remains in its own individual files. More important, it doesn't get pushed down to the client and is therefore not readily exposed (this is, of course, always quite dependent on security measures). As I described last time, the driving reason for wanting code that stays on the server is that I needed a way to store my credentials for interacting with DocumentDB. Because I wanted to try out the Node.js path, the aforementioned SDK made conversing with DocumentDB a lot easier. That I had the original DocumentDB Node example to lean on for some basics on how to use the SDK was a huge help.



	FY 2011	FY 2012	FY 2013	% CHANG
CME	$5,687.00	$4,678.00	$5,120.00	9%
Braeburn	$4,550.00	$5,234.00	$6,243.00	19%
Circuit Town	$7,890.00	$6,678.00	$6,423.00	-4%
Emporium	$1,223.00	$1,250.00	$1,280.00	2%
	$1,223.00	$1,250.00	$1,180.00	-6
		$1,280.00	$1,370.00	7
			$21,616.00	6

My server-side code (which I'll refer to as my API) consists of four key ingredients:

- An api.js file that acts as a router to ensure my API functions can be easily found.
- My core module, ninjas.js, containing the API functions: getNinjas, getNinja and updateDetails.
- A controller class, DocDbDao (called by those API functions) that performs the interaction with DocumentDb.
- A DocumentDb utility file that knows how to ensure the relevant database and collection exist.



Count of retreived Ninjas: 5

Kacy Catanzaro , Birthdate:1/14/1990
JulieSan [Oniwaban] , Birthdate:1/1/1983
SampsonSan , Birthdate:1/28/2008
Leonardo [Oniwaban] , Birthdate:1/1/1984
Raphael , Birthdate:1/1/1985

Figure 2 **The Ninja Listing in My Aurelia Web Site**

Wiring up the Client, Server and Cloud

In my earlier Aurelia sample, the method to get all ninjas made a direct HTTP call to an ASP.NET Web API that used .NET and Entity Framework to interact with a SQL Server database:

```
retrieveNinjas() {
  return this.http.createRequest
    ("/ninjas/?page=" + this.currentPage + "&pageSize=100&query=" +
      this.searchEntry)
      .asGet().send().then(response => {
        this.ninjas = response.content;
    });
  }
```

The results were passed to a pairing of the client side view and view-model (ninjaList.js and ninjaList.html) that outputs the page shown in **Figure 2**.

In the new solution, this method, renamed to getNinjas, now calls into my server-side Node.js API. I'm using the more advanced httpClient.fetch (bit.ly/1M8EmnY) instead of httpClient this time to make the call:

```
getNinjas(params){
  return this.httpClient.fetch(`/ninjas?q=${params}`)
    .then(response => response.json())
    .then(ninjas => this.ninjas = ninjas);
}
```

I've configured httpClient elsewhere to know the base URL.

Notice that my fetch method is calling a URI that includes the term ninjas. But this is not referring to my ninjas.js in the server-side API. It could be /foo—just a random reference that will be resolved by my server-side router. My server-side router (which uses Express, not Aurelia, because Aurelia only handles the client side) specifies that calls to api/ninjas be routed to the getNinjas function of the ninjas module. Here's the code from api.js where this is defined:

```
router.get('/api/ninjas', function (request, response) {
  ninjas.getNinjas(req, res);
});
```

Now my getNinjas function (**Figure 3**) uses string interpolation (mzl.la/1fhuSIg) to build up a string to represent the SQL for querying DocumentDB, then asks the controller to execute the query and return the results. If a filter on the name property has been requested in the UI, I will append a WHERE clause to the query. The main query projects only the relevant properties I'll need on the page, including the ID. (See more about DocumentDB projection queries at documentdb.com/sql/demo.) The query is passed into the find method of the docDbDao controller. The find method, which is unchanged from the original I described in the first part of this series, uses the Node.js SDK along with the credentials stored in config.js to query the database for the ninjas. getNinjas then takes those results and returns them to the client that requested them. Note that although the call to DocumentDB does return results as JSON, I still need to explicitly use the response.json function to pass the results back. This alerts the caller that the results are JSON-formatted.

Client-Side Response to an Edit Request

As you can see in **Figure 2**, it's then possible to edit a ninja by clicking on the pencil icon. Here's a link I built up in the page markup:

Figure 3 **The getNinjas Function in the ninjas.js Server-Side Module**

```
getNinjas: function (request, response) {
  var self = this;
  var q = '';
  if (request.query.q != "undefined" && req.query.q > "") {
    q= `WHERE CONTAINS(ninja.Name,'${req.query.q}')`;
  }

  var querySpec = {
    query:
   `SELECT ninja.id, ninja.Name,ninja.ServedInOniwaban,
     ninja.DateOfBirth FROM ninja ${q}`
  };

  self.docDbDao.find(querySpec, function (err, items) {
    if (err) {
      // TODO: err handling
    } else {
      response.json(items);
    }
  })
},
```

Figure 4 **The Document I Requested, Which Is Stored in my DocumentDb Ninjas Collection**

```
{
  "id": "1",
  "Name": "Kacy Catanzaro",
  "ServedInOniwaban": false,
  "Clan": "American Ninja Warriors",
  "Equipment": [
    {
      "EquipmentName": "Muscles",
      "EquipmentType": "Tool"
    },
    {
      "EquipmentName": "Spunk",
      "EquipmentType": "Tool"
    }
  ],
  "DateOfBirth": "1/14/1990",
  "DateCreated": "2015-08-10T20:35:09.7600000",
  "DateModified": 1444152912328
}
```

```
<a href="#/ninjas/${ninja.id}" class=
  "btn btn-default btn-sm">
  <span class="glyphicon glyphicon-pencil" />
</a>
```

By clicking the edit icon for my first row, whose ID happens to be "1," I get this URL:

```
http://localhost:9000/app/#/ninjas/1
```

Using the Aurelia routing feature on the client side in app.js, I've specified that when this URL pattern is requested, it should then call the edit module and pass in the id to the activate method of the edit view-model as a parameter, indicated by the wildcard (*Id):

```
{ route: 'ninjas/*Id', moduleId: 'edit', title:'Edit Ninja' },
```

Edit refers to the client-side edit.js module that's paired with the edit.html view. The activate function of my edit module then calls another function in this module named retrieveNinja, passing in the requested Id:

```
retrieveNinja(id) {
  return this.httpClient.fetch(`/ninjas/${id}`)
    .then(response => response.json())
    .then(ninja => this.ninja = ninja);

}
```

Passing the Edit Request to the Server-Side API

Again, I'm using httpClient.fetch to request api/ninjas/[id] (in my case api/ninjas/1) from the API. The server-side router says that when a request with this pattern comes in, it should route to the getNinja function of the ninjas module. Here's what that routing looks like:

```
router.get('/api/ninjas/:id', function(request, response) {
  ninjas.getNinja(request, response);
});
```

The getNinja method then makes another request to the docDbDao controller, this time to the getItem function, to get the ninja data from DocumentDb. The results are JSON documents that are stored in my DocumentDB database, as shown in **Figure 4**:

Passing the Results Back to the Client

This JSON object is returned to the ninjas.getNinja function, which then returns it to the caller, in this case the edit.js module on the client. Aurelia then binds edit.js to the edit.html template and outputs a page, which is designed to display this graph.

The edit view allows users to modify four pieces of data: the ninja's name and birthdate (both strings), the clan (a dropdown) and whether the ninja served in Oniwaban. The Clan dropdown uses a special type of Web component called a custom element. The Aurelia implementation of the custom element spec is unique. Because I'm using that feature to bind data, and this is a data column, let me show you how that's done.

Data Bind with an Aurelia Custom Element

Like other view and view-model pairs in Aurelia, a custom element is composed of a view for the markup and a view-model for the logic. **Figure 5** shows the third file that's involved, clans.js, which provides a list of clans.

The view for this element (dropdown.html) uses a Bootstrap "select" element, which I still think of as a dropdown:

```
<template>
  <select value.bind="selectedClan">
    <option repeat.for="clan of clans" model.bind="clan.name">${clan.name}</option>
  </select>
</template>
```

Figure 5 **Clans.js**

```
export function getClans(){
  var clans = [], propertyName;
  for(propertyName in clansObject) {
    if (clansObject.hasOwnProperty(propertyName)) {
      clans.push({ code: clansObject[propertyName], name: propertyName });
    }
  }
  return clans;
}

var clansObject = {
  "American Ninja Warriors": "anj",
  "Vermont Clan": "vc",
  "Turtles": "t"
};
```

Figure 6 **Custom Element Code for the Model in Dropdown.js**

```
import $ from 'jquery';
import {getClans} from '../clans';
import {bindable} from 'aurelia-framework';
export class Dropdown {
  @bindable selectedClan;

  attached() {
    $(this.dropdown).dropdown();
  }

  constructor() {
    this.clans = getClans();
  }

}
```

Notice the value.bind and repeat.for, which should be familiar if you read my earlier column on data binding with Aurelia. The options within the select element bind to the clan model defined in clans.js and then display the clan name. Because my Clans object is simple, with only a name and code (which is extraneous in this example), I could simply use value.bind there. But I'm sticking with using model.bind because it's a better pattern for me to remember.

The dropdown.js module wires up the markup with the clans, as shown in **Figure 6**.

> More importantly, I can very easily reuse my custom element in other views.

Additionally, the custom element makes it possible for me to use much simpler markup in my edit view:

```
<dropdown selected-clan.two-way="ninja.Clan"></dropdown>
```

Notice that I'm using two Aurelia-specific features here. First, my property is called selectedClan in the view-model, but selected-clan in the markup. The Aurelia convention, based on HTML's need for attributes to be lowercase, is to make all custom properties of export names lowercase and hyphenated, so it expects that hyphen to be where the camel-cased letters begin. Second, rather than value.bind, I am explicitly using two-way binding here so the clan will be rebound to the ninja when the selection changes.

More important, I can very easily reuse my custom element in other views. In my case, it simply provides more readability and, therefore, more maintainability. But in larger applications, the reuse of the markup and logic is a great benefit.

Pushing Edits Back to DocumentDB

My markup is reading the two pieces of equipment in the graph and displaying them. For the sake of brevity, I'll leave editing the equipment data to another day.

The last bit of work now is to pass changes back up to the API and send them to DocumentDB. This is triggered with the Save button.

The markup for the Save button uses another Aurelia paradigm, click.delegate—which uses JavaScript event delegation—allowing me to delegate the action to a save function defined in edit.js.

The save function, shown in **Figure 7**, creates a new object, ninjaRoot, using the relevant properties from the ninja property that was defined in getNinja, which is then bound to the markup, allowing the user to update from the browser.

Save then uses the now-familiar httpClient.fetch to request an API URL called udpateDetails and pass in the ninjaRoot object as the body. Notice, also, that I'm specifying this as a post method, not a get. The API router tells Node.js to route to the updateDetails method of the ninjas module.

```
router.post('/api/updateDetails', function(request,response){
  ninjas.updateDetails(request,response);
});
```

Figure 7 **The ninjas.save Function**

```
save() {
  this.ninjaRoot = {
    Id: this.ninja.id,
    ServedInOniwaban: this.ninja.ServedInOniwaban,
    Clan: this.ninja.Clan,
    Name: this.ninja.Name,
    DateOfBirth: this.ninja.DateOfBirth
  };
  return this.httpClient.fetch('/updateDetails', {
    method: 'post',
    body: json(this.ninjaRoot)
  }).then(response => {this.router.navigate('ninjaList');
  });
}
```

Figure 8 **The updateItem Method in the docDbDao Controller Uses the Node.js SDK to Talk to DocumentDB**

```
updateItem: function (item, callback) {
  var self = this;

  self.getItem(item.Id, function (err, doc) {
    if (err) {
      callback(err);

    } else {
      doc.Clan=item.Clan;
      doc.Name=item.Name;
      doc.ServedInOniwaban=item.ServedInOniwaban;
      doc.DateOfBirth=item.DateOfBirth;
      doc.DateModified=Date.now();
      self.client.replaceDocument(doc._self, doc, function (err, replaced) {
        if (err) {
          callback(err);

        } else {
          callback(null, replaced);
        }
      });
    }
  });
},
```

Now let's look at the server-side updateDetails method in ninjas.js:

```
updateDetails: function (request,response) {
  var self = this;
  var ninja = request.body;
  self.docDbDao.updateItem(ninja, function (err) {
    if (err) {
      throw (err);
    } else {
      response.send(200);
    }
  })
},
```

I extract the ninjaRoot stored in the request body and set that to a ninja variable, then pass that variable into the controller's updateItem method. As **Figure 8** shows, I've modified updateItem a bit since part one of my article to accommodate my ninja type.

UpdateItem retrieves the document from the database using the id, updates the relevant properties of that document and then uses the SDK DocumentDBClient.replaceDocument method to push the change to my Azure DocumentDB database. A callback alerts me to the operation's completion. That callback is then noted by the updateDetails method, which returns a 200 response code back to the client module that called the API. If you look back at the client-side save method, you'll notice that its callback routes to ninjaList. So when the update has been successfully posted, the user is presented with the original listing page of the ninjas. Any edits to the ninja just changed will be visible in that list.

Are We Ninjas Yet?

This solution threw me into an abyss of difficulties and dead ends. While my core goal was to have Aurelia talk to the DocumentDB database, I also wanted to use these technologies to take advantage of their benefits. That meant having to comprehend so many things in the JavaScript world, manage node installations, use Visual Studio Code for the first time because of its ability to debug the Node.js and learn even more about DocumentDB. While there are many other things you may want to do in even a simple sample like mine, this article should give you a basic understanding of how things function from one end to another.

One important point to keep in mind is that DocumentDB—like any NoSQL database—is designed for large volumes of data. It's not cost-effective for tiny little bits of data such as I use in my sample. But in order to explore the functionality of connecting to the database and interacting with the data, large amounts of data were not required, so five objects sufficed.■

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

THANKS to the following technical expert for reviewing this article:
Patrick Walters

Modern Tools for Web Development: Grunt and Gulp

Adam Tuliper

There are many tools available to the modern Web developer. Two that are often found in today's Web projects are the JavaScript task runners Grunt and Gulp. Using JavaScript to run a task might seem like a foreign concept if you've never done it, or if you're used to plain vanilla Visual Studio Web development, but there are good reasons to give them a try. JavaScript task runners, which work outside of the browser and typically use Node.js at the command line, allow you to easily run front-end development-related tasks, including minification, concatenating multiple files, determining script dependencies and injecting script references *in proper order* to HTML pages, creating unit test harnesses, processing front-end build scripts like TypeScript or CoffeeScript, and more.

Which One—Grunt or Gulp?

Choosing a task runner is mostly a personal or project preference, unless there's a plug-in you want to use that only supports a particular task runner. The primary differences are that Grunt is driven by JSON configuration settings and that each Grunt task typically must create intermediate files to pass things off to other tasks, while Gulp is driven by executable JavaScript code (that is, not just JSON) and can stream results from one task to the next without having to use temporary files. Gulp is the newer kid on the block and, as such, you see a lot of newer projects using it. Still, Grunt has plenty of well-known supporters, such as jQuery, which uses it to build … jQuery. Both Grunt and Gulp work via plug-ins, which are modules you install to handle a particular task. There's a vast ecosystem of plug-ins available, and often you'll find a task package that supports both Grunt and Gulp, so, again, using one or the other generally comes down to a personal choice.

This article discusses:

- Installing and Using Grunt
- Installing and Using Gulp
- The Task Runner Explorer
- Gulp and ASP.NET 5

Technologies discussed:

Visual Studio 2015, Grunt, Gulp, Node.js, ASP.NET 5

Installing and Using Grunt

The installer for both Grunt and Gulp is Node Package Manager (npm), which I briefly covered in my October article (`msdn.com/magazine/mt573714`). The command to install Grunt actually has two parts. The first is a one-time install of the Grunt command-line interface. The second is installing Grunt into your project folder. This two-part install lets you use multiple versions of Grunt on your system, and use the Grunt command-line interface from any path:

```
#only do this once to globally install the grunt command line runner
npm install -g grunt-cli
#run within your project folder one time to create package.json
#this file will track your installed npm dependencies
#like grunt and the grunt plug-ins, similar to bower.json (see the last article)
npm init
#install grunt as a dev dependency into your project (again see last article)
#we will still need a gruntfile.js though to configure grunt
npm install grunt -save-dev
```

Figure 1 **The Grunt Configuration File**

```
module.exports = function (grunt) {
  // Where does uglify come from? Installing it via:
  // npm install grunt-contrib-uglify --save-dev
  grunt.initConfig({
    uglify: {
      my_target: {
        files: {
          'dest/output.min.js': '*.js'
        }
      }
    }
  });
  // Warning: make sure you load your tasks from the
  // packages you've installed!
  grunt.loadNpmTasks('grunt-contrib-uglify');

  // When running Grunt at cmd line with no params,
  // you need a default task registered, so use uglify
  grunt.registerTask('default', ['uglify']);

  // You can include custom code right inside your own task,
  // as well as use the above plug-ins
  grunt.registerTask('customtask', function () {
    console.log("\r\nRunning a custom task");
  });

};
```

Grunt Configuration

The Grunt configuration file is simply a JavaScript file with a wrapper function containing the configuration, plug-in loading and task definition, as shown in **Figure 1**.

You can run the tasks in **Figure 1** at the command line simply by calling:

```
#no params means choose the 'default' task (which happens to be uglify)
grunt
#or you can specify a particular task by name
grunt customtask
```

After doing so, you'll find the uglified (minified) and concatenated result at wwwroot/output-min.js. If you've used ASP.NET minification and bundling, you'll see that this process is different—it isn't tied to running your app or even compilation, and there are many more options you can choose for tasks like uglifying. Personally, I find this workflow to be more straightforward and understandable.

You can chain tasks together with Grunt to be dependent on one another. These tasks will run synchronously, so that one must complete before moving on to the next.

```
#Specify uglify must run first and then concat. Because grunt works off
#temp files, many tasks will need to wait until a prior one is done
grunt.registerTask('default', ['uglify', 'concat']);
```

Figure 2 **The Gulp Configuration File**

```
// All the 'requires' to load your
// various plug-ins and gulp itself
var gulp = require('gulp');
var concat = require('gulp-concat');

// A custom task, run via: gulp customtask
gulp.task('customtask', function(){
  // Some custom task
});

// Define a default task, run simply via: gulp
gulp.task('default', function () {
  gulp.src('./lib/scripts/*.js')
    .pipe(concat('all-scripts.js'))
    .pipe(gulp.dest('./wwwroot/scripts'));
});
```

Installing and Using Gulp

Installing Gulp is similar to installing Grunt. I'll go into a little more detail with Gulp, but note that you can do similar things with either; I just don't want to be too repetitive. Gulp has both a global install, so you can use it from any path on your system, and a local install into your project folder that's versioned to your particular project. The globally installed Gulp will hand off control to the one installed in your local project if it finds it, thus respecting your project's version of Gulp:

```
#Only do this once to globally install gulp
npm install -g gulp
#Run within your project folder one time to create package.json
#this file will track your installed npm dependencies
#like gulp and the gulp plug-ins
npm init
#Install gulp as a dev dependency into your project
#we will still need a gulpfile.js to configure gulp
npm install gulp --save-dev
```

Gulp Configuration and API

Gulp configuration is significantly different from Grunt. The gulpfile.js configuration file, which typically has the structure shown in **Figure 2**, contains the "requires" for loading plug-ins and then defining tasks. Note that I'm not using JSON configuration settings here; instead, tasks are code-driven.

> The Grunt configuration file is simply a JavaScript file with a wrapper function containing the configuration, plug-in loading and task definition.

Gulp works via several key APIs and concepts: src, dest, pipe, task and globs. The gulp.src API tells Gulp which files to open to work with, and those files are then typically piped to some other function, instead of temp files being created. This is a key difference from Grunt. The following example shows some basic examples of gulp.src without piping the results, which I'll cover shortly. This API call takes what's known as a glob as a parameter. A glob is basically a pattern you can enter (somewhat like a regular expression) to, for example, specify a path to one or more files (see more about globs at github.com/isaacs/node-glob):

```
#Tell gulp about some files to work with
gulp.src('./input1.js');

#This will represent every html file in the wwwroot folder
gulp.src('./wwwroot/*.html')

#You can specify multiple expressions
gulp.src(['./app/**/*.js', './app/*.css']
```

The dest (destination) API, as you might imagine, specifies a destination and also takes a glob. Because globs are so flexible for defining paths, you can either output individual files or output to a folder:

```
#Tell dest you'll be using a file as your output
gulp.dest ('./myfile.js');

#Or maybe you'll write out results to a folder
gulp.dest ('./wwwroot');
```

DOMAINS | MAIL | HOSTING | eCOMMERCE | **SERVERS**

1&1 CLOUD SERVER

TEST THE BEST!

TOP PERFORMER
CLOUD SPECTATOR

Powered by
intel®
Cloud Technology

Easy to use – ready to go.

The 1&1 Cloud Server provides superior CPU, RAM, and SSD performance. Give your cloud projects the perfect combination of flexibility and efficient features.

- ✓ **Load balancing**
- ✓ **SSD storage**
- ✓ **Billing by the minute**
- ✓ **Intel® Xeon® Processor E5-2660 v2 and E5-2683 v3**

1 month free!

Then from $4.99/month.*

1 TRIAL TRY FOR 30 DAYS ★★★★★

1 MONTH FLEXIBLE PAYMENT OPTIONS ★★★★★

1 CALL SPEAK WITH AN EXPERT 24/7 ★★★★★

☎ 1 (877) 461-2631

1&1®

1and1.com

* 1&1 Cloud Server is available free for one month, after which regular pricing starts from $4.99/month. No setup fee is required. Visit 1and1.com for full offer details, terms and conditions. Intel, the Intel Logo, Intel Inside, the Intel Inside logo, Intel Experience What's Inside are trademarks of Intel Corporation in the U.S. and/or other countries. 1&1 and the 1&1 logo are trademarks of 1&1 Internet, all other trademarks are property of their respective owners. ©2015 1&1 Internet. All rights reserved.

Figure 3 **Gulpfile.js**

```
var gulp = require('gulp');
var fs = require('fs');
var concat = require("gulp-concat");
var header = require("gulp-header");

// Read *.js, concat to one file, write headers, output to /processed
gulp.task('concat-header', function () {
  var appVersion = fs.readFileSync('version.txt');
  var copyright =fs.readFileSync('copyright.txt');

  gulp.src('./scripts/*.js')
  .pipe(concat('all-scripts.js'))
  .pipe(header(copyright, {version: appVersion}))
  .pipe(gulp.dest('./scripts/processed'));
});
```

Tasks in Gulp are simply the code you write to do something. The format for tasks is pretty simple, but tasks can be used in several ways. The most straightforward way is to have a default task and one or more other tasks:

```
gulp.task('customtask', function(){
  // Some custom task to ex. read files, add headers, concat
});
gulp.task('default', function () {
  // Some default task that runs when you call gulp with no params
});
```

Visual Studio provides support for Gulp and Grunt via the Task Runner Explorer, which is included in Visual Studio 2015 and available as a Visual Studio Extension.

Tasks can be executed in parallel or they can be dependent on each other. If you don't care about the order, you can just chain them together, as follows:

```
gulp.task('default', ['concatjs', 'compileLess'], function(){});
```

This example defines the default task, which does nothing but run separate tasks to concatenate JavaScript files and compile LESS files (assuming that code was in the tasks named here). If the requirements dictate one task completes before the other executes, you need to make one task dependent on another and then run multiple tasks. In the following code, the default task waits for concat to complete first, which in turn waits for uglify to complete:

```
gulp.task('default', ['concat']);
gulp.task('concat', ['uglify'], function(){
  // Concat
});
gulp.task('uglify', function(){
  // Uglify
});
```

The pipe API is used to pipe results from one function to another using the Node.js stream API. The workflow typically is: read src, pipe to a task, pipe the results to dest. This complete gulpfile.js example reads all JavaScript files in /scripts, concatenates them into a single file and writes the output to another folder:

```
// Define plug-ins - must first run: npm install gulp-concat --save-dev
var gulp = require('gulp');
var concat = require('gulp-concat');

gulp.task('default', function () {
  #Get all .js files in /scripts, pipe to concatenate, and write to folder
  gulp.src('./lib/scripts/*.js')
    .pipe(concat('all-scripts.js'))
    .pipe(gulp.dest('./wwwroot/scripts'));
}
```

Here's a practical, real-world example. Often you'll want to concatenate files and/or add informational headers to your source code files. You can easily do that in a few steps by adding a couple of files to the root of your Web site (you could do this task all in code, as well—see the gulp-header docs). First, create a file called copyright.txt that contains the headers to add, like this:

```
/*
MyWebSite Version <%= version %>
https://twitter.com/adamtuliper
Copyright 2015, licensing, etc
*/
```

Next, create a file named version.txt containing the current version number (there are plug-ins, like gulp-bump and grunt-bump, to increment version numbers, as well):

```
1.0.0
```

Now, install the gulp-header and gulp-concat plug-ins in your project root:

```
npm install gulp-concat gulp-header --save-dev
```

Alternatively, you can manually add them to the package.json file and let Visual Studio do the package restore for you.

Last, you just need gulpfile.js to tell Gulp what to do, as shown in **Figure 3**. If you don't want to concatenate all your scripts together and instead just add headers to every file, you can simply comment out the pipe(concat) line. Simple, right?

You then simply run the task via the following command and, voila, you've concatenated all the .js files, added a custom header and written the output to the ./scripts/processed folder:

```
gulp concat-header
```

The Task Runner Explorer

Visual Studio provides support for Gulp and Grunt via the Task Runner Explorer, which is included in Visual Studio 2015 and available as a Visual Studio Extension. You can find it in View | Other

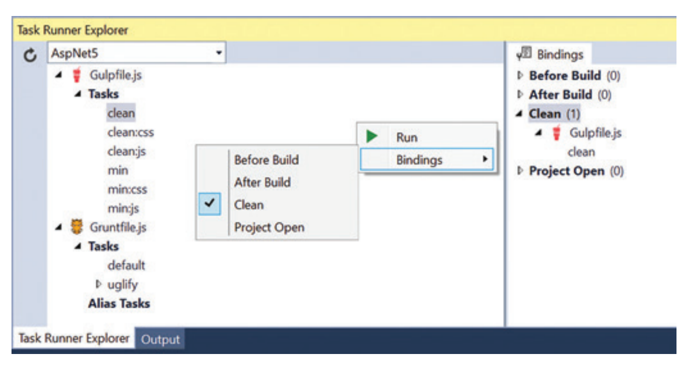
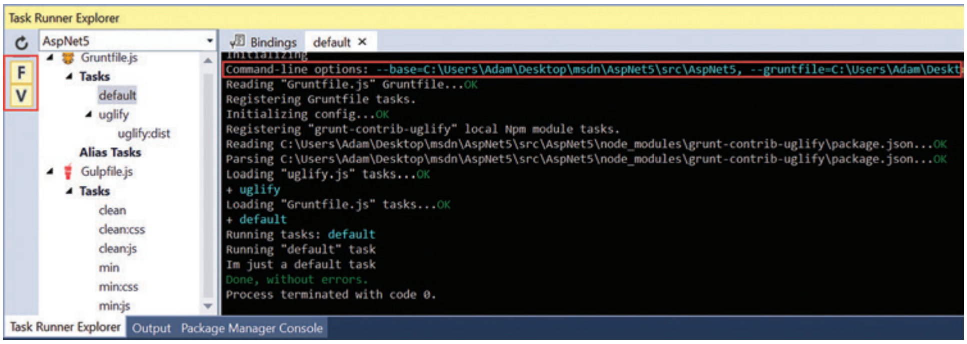


Figure 4 **The Task Runner Explorer Showing Both Grunt and Gulp Tasks with Options**

DynamicPDF Converter
DynamicPDF Rasterizer
DynamicPDF Core Suite
DynamicPDF Core Suite



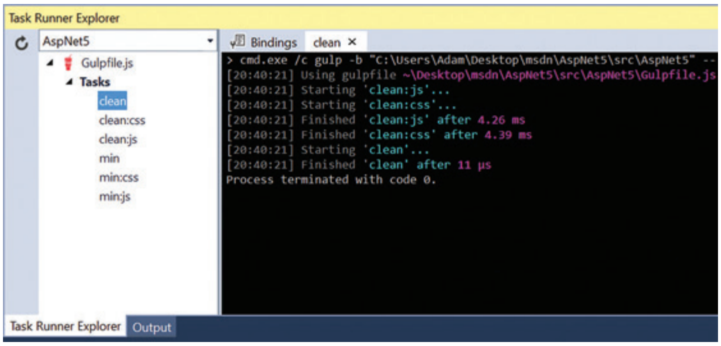
```
Initializing
Command-line options: --base=C:\Users\Adam\Desktop\msdn\AspNet5\src\AspNet5, --gruntfile=C:\Users\Adam\Deskt
Reading "Gruntfile.js" Gruntfile...OK
Registering Gruntfile tasks.
Initializing config...OK
Registering "grunt-contrib-uglify" local Npm module tasks.
Reading C:\Users\Adam\Desktop\msdn\AspNet5\src\AspNet5\node_modules\grunt-contrib-uglify\package.json...OK
Parsing C:\Users\Adam\Desktop\msdn\AspNet5\src\AspNet5\node_modules\grunt-contrib-uglify\package.json...OK
Loading "uglify.js" tasks...OK
+ uglify
Loading "Gruntfile.js" tasks...OK
+ default
Running tasks: default
Running "default" task
Im just a default task
Done, without errors.
Process terminated with code 0.
```

Figure 5 **Additional Grunt Options and the Command Line**

Windows | Task Runner Explorer. The Task Runner Explorer is pretty awesome because it will detect if you have a gulpfile.js or gruntfile.js in the project, parse the tasks and give you a UI to execute the tasks it finds, as shown in **Figure 4**. Moreover, you can define tasks to run when predefined actions happen in your project, which I'll cover next because ASP.NET 5 uses this functionality in its default templates. Simply right-click on a task to execute it or to bind it to a particular action, for example, to execute a task on Project Open.

The ASP.NET 5 templates included in Visual Studio 2015 use Gulp, and they install Gulp into your project's node_components folder so it's all ready for you to use in your project.

As **Figure 5** shows, Grunt provides options when you highlight a Grunt task that you won't see with Gulp, in particular the Force (to ignore warnings) and Verbose options (the F and V on the upper left). These are simply parameters passed to the Grunt command line. The great thing about the Task Runner Explorer is it shows you the commands it passes to the Grunt and Gulp command line (in the task output window), so there's no mystery as to what's going on behind the scenes.



```
> cmd.exe /c gulp -b "C:\Users\Adam\Desktop\msdn\AspNet5\src\AspNet5" --
[20:40:21] Using gulpfile ~\Desktop\msdn\AspNet5\src\AspNet5\Gulpfile.js
[20:40:21] Starting 'clean:js'...
[20:40:21] Starting 'clean:css'...
[20:40:21] Finished 'clean:js' after 4.26 ms
[20:40:21] Finished 'clean:css' after 4.39 ms
[20:40:21] Starting 'clean'...
[20:40:21] Finished 'clean' after 11 µs
Process terminated with code 0.
```

Figure 6 **Out-of-the-Box Tasks in the ASP.NET 5 Preview Templates**

Gulp and ASP.NET 5

The ASP.NET 5 templates included in Visual Studio 2015 use Gulp, and they install Gulp into your project's node_components folder so it's all ready for you to use in your project. You can still use Grunt in an ASP.NET 5 project, of course; you'll just need to remember to install it into your project via npm or by adding it to packages.json inside devDependencies and letting the auto package restore feature in Visual Studio do its thing. I want to stress: You can do all of this via the command line or inside of Visual Studio, whichever you prefer.

As of this writing, the current ASP.NET 5 templates include a couple of tasks to minify and concatenate .css and .js files. In previous versions of ASP.NET, these tasks were handled in compiled code at run time, which ideally isn't where or when these sorts of tasks should be done. As you can see in **Figure 6**, the tasks named clean and min call their css and js methods to minify those files or to clean previously minified files.

The following Gruntfile.js line shows another example of running multiple tasks at the same time:

```
gulp.task("clean", ["clean:js", "clean:css"]);
```

You have the option to bind Grunt and Gulp tasks to four different actions in Visual Studio. With MSBuild, it was common to define a pre-build and post-build command line to execute various tasks. With the Task Runner Explorer, you can define Before Build, After Build, Clean and Project Open events to execute these tasks. Doing so simply adds comments to the gulpfile.js or gruntfile.js files that don't affect the execution, but are looked for by Task Runner Explorer. To see the "clean" binding in the ASP.NET 5 gulpfile.js, take a look at this line at the top of the file:

```
// <binding Clean='clean' />
```

That's all that's required to hook into the event.

Wrapping Up

Both Grunt and Gulp are great additions to your Web arsenal. Both are well supported and have a vast ecosystem of plug-ins available. Every Web project can benefit from something they can provide. For more information, be sure to check out the following:

- The Grunt "Getting started" documentation (bit.ly/1dwDWq)
- Gulp recipes (bit.ly/1L8SkUC)
- Package Management and Workflow Automation: Machine Package Managers (bit.ly/1FLwGW8)
- Mastering Node.js Modules and Packages (bit.ly/1N8UKon)
- Adam's Garage (bit.ly/1NSAYxK) ■

Adam Tuliper *is a senior technical evangelist with Microsoft living in sunny SoCal. He is a Web dev, game dev, Pluralsight author and all-around tech lover. Find him on Twitter: @AdamTuliper or on his Adam's Garage blog at bit.ly/1NSAYxK.*

Thanks to the following Microsoft technical expert for reviewing this article: *Michael Palermo*

Azure Service Fabric and the Microservices Architecture

Cesar de la Torre, Kunal Deep Singh and Vaclav Turecek

Microservices is a hot buzzword at the moment. While there are many presentations and conference talks about the subject, a lot of developers remain confused. A common question we've heard: "Isn't this just another service-oriented architecture (SOA) or domain-driven design (DDD) approach?"

Certainly, many of the techniques used in the microservices approach derive from the experiences of developers in SOA and DDD. You can think of microservices as "SOA done right," with principles and patterns like autonomous services, Bounded-Context pattern and event-driven all having their roots in SOA and DDD.

In this article we address both microservices theory and implementation. We'll start with a short introduction to microservices, and then move on to the practical side and how you can build and deploy microservices with Azure Service Fabric. Finally, we'll show why this platform is a great fit when building microservices.

This article discusses:

- The microservices architecture approach and how it enables applications and services built for scale and agility
- The benefits of Microsoft Azure Service Fabric, the new Microsoft Platform as a Service that streamlines development and operation of microservices-based apps
- Understanding Azure Service Fabric programming models
- How to build stateless microservices with Azure Service Fabric, and an introduction to stateful Reliable Services and Actor Services

Technologies discussed:

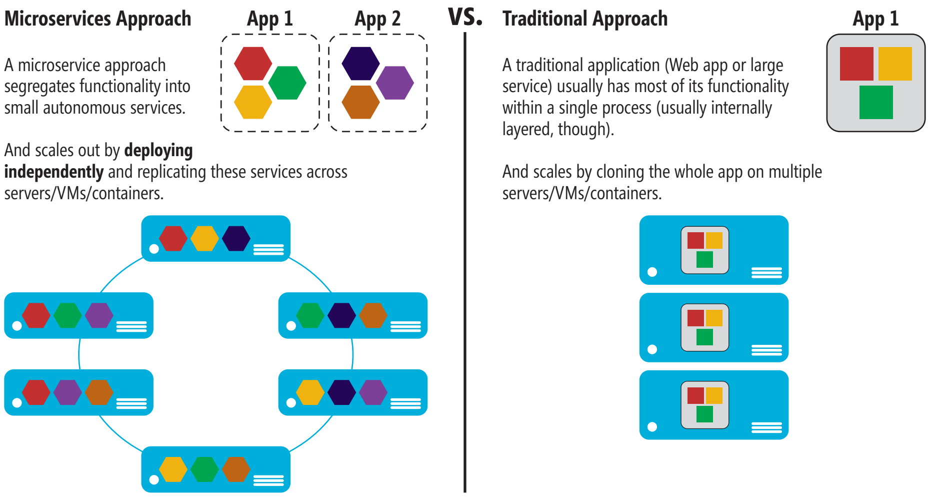
Microservices, Microsoft Azure, Azure Service Fabric, ASP.NET, MVC

As the name implies, microservices architecture is an approach to build a server application as a set of small services, each service running in its own process and communicating with each other via protocols such as HTTP and WebSockets. Each microservice implements specific, end-to-end domain and business capabilities within a certain Bounded-Context per service, and must be developed autonomously and deployed independently by automated mechanisms. Finally, each service should own its related domain data model and domain logic, and can employ different data storage technologies (SQL, No-SQL) and different programming languages per microservice.

Examples of microservices include protocol gateways, user profiles, shopping carts, inventory processing, purchase subsystem, payment processing, and queues and caches.

Why microservices? In a word, agility. Over the long term, microservices enable superior maintainability in large, complex and highly scalable systems by designing applications based on many independently deployable services that allow for granular release planning.

As an additional benefit, microservices can scale out independently. Instead of having giant monolithic application blocks that you must scale out at once, you can instead scale out specific microservices. That way, just the functional area that needs more processing power or network bandwidth to support demand can be scaled, rather than scaling out other areas of the application that really don't need it.



A microservice approach segregates functionality into small autonomous services.

And scales out by **deploying independently** and replicating these services across servers/VMs/containers.

A traditional application (Web app or large service) usually has most of its functionality within a single process (usually internally layered, though).

And scales by cloning the whole app on multiple servers/VMs/containers.

Figure 1 **Microservices Approach Compared to Traditional Server Application Approach**

Architecting fine-grained microservice applications enables continuous integration and continuous development practices, and accelerates delivery of new functions into the application. Fine-grain decomposition of applications also lets you run and test microservices in isolation, and to evolve microservices independently while maintaining rigorous contracts among them. As long as you don't break the contracts or interfaces, you can change any microservice implementation under the hood and add new functionality without breaking the other microservices that depend on it.

As **Figure 1** shows, with the microservices approach it's all about efficiency for agile changes and rapid iteration, because you're able to change specific, small portions of large, complex and scalable applications.

Data Sovereignty Per Microservice

An important rule in this approach is that each microservice must own its domain data and logic under an autonomous lifecycle, with independent deployment per microservice. This is really no different from how a full application owns its logic and data.

This approach means that the conceptual model of the domain will differ between sub-systems or microservices. Consider enterprise applications, where customer relationship management (CRM) applications, transactional purchase subsystems and customer support subsystems each call on unique customer entity attributes and data and employ a different Bounded-Context.

This principle is similar in DDD where each Bounded-Context, which is a pattern comparable to a subsystem/service, must own its domain-model (data and logic). Each DDD Bounded-Context would correlate to a different microservice.

On the other hand, the traditional (or monolithic) approach used in many applications is to have a single and centralized database (often a normalized SQL database) for the whole application and all its internal subsystems, as shown in **Figure 2**. This approach looks initially simpler and seems to enable re-use of entities in different subsystems to make everything consistent. But the reality is you end up with huge tables that serve many different subsystems and include attributes and columns that simply aren't needed in most cases. It's like trying to use the same physical map for hiking a short trail, taking a day-long car trip and learning geography.

Stateless or Stateful Microservices?

As mentioned earlier, each microservice must own its domain model. In the case of stateless microservices, the databases will be external, employing relational options like SQL Server or NoSQL options like MongoDB or Azure Document DB. Going further, the services themselves can be stateful, which means the data resides within the same microservice. This data can exist not just within the same server, but within the same microservice's process, in-memory and persisted on hard drive and replicated to other nodes.

Stateless is a perfectly valid approach and easier to implement than stateful microservices, as it is similar to traditional and well-known patterns. But stateless microservices impose latency between the process and data sources, while also presenting more moving pieces when improving performance via additional cache and queues. The result: you can end up with complex architectures with too many tiers.

Stateful microservices, on the other hand, can excel in advanced scenarios, as there is no latency between the domain logic and data. Heavy data processing, gaming back ends, databases as a service, and other low-latency scenarios all benefit from stateful services, which enable local state for faster access.

The drawback: Stateful services impose a level of complexity in order to

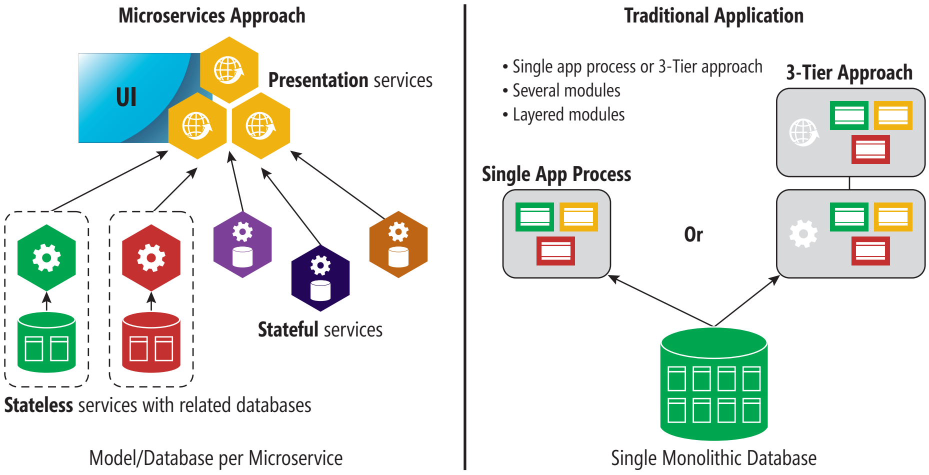


Figure 2 **Data Sovereignty Comparison: Microservices vs. Monolithic**

scale out. Functionality that would usually be implemented within the external database boundaries must be addressed for things such as data replication across stateful microservices replicas, data partitioning and so on. But this is precisely one of the areas where Service Fabric can help most—by simplifying the development and lifecycle of stateful microservices.

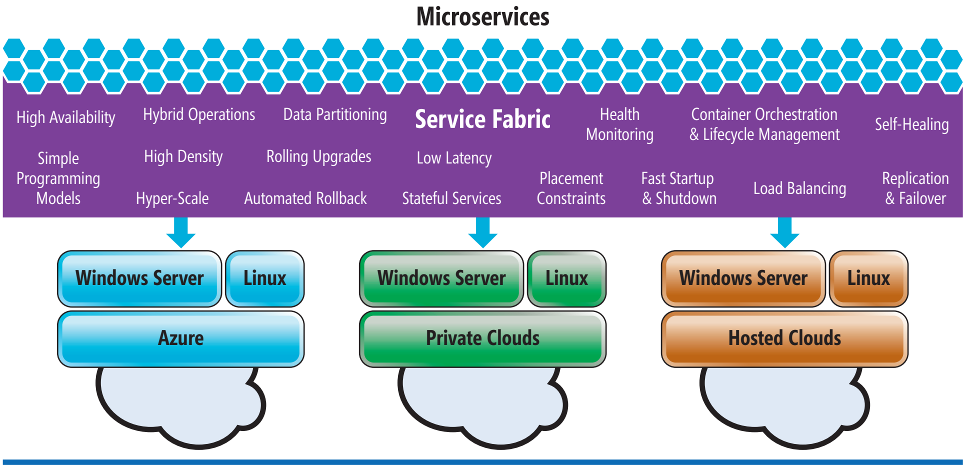


Figure 3 **Microsoft Azure Service Fabric**

Benefits of Azure Service Fabric

All the goodness that comes with the microservices approach comes with a drawback. Distributed computing and complex microservices deployments can be hard to manage if you do it by yourself. Service Fabric provides the plumbing needed to create, deploy, run, and manage microservices in an effective and efficient way.

What is Service Fabric? It's a distributed systems platform used to build hyper-scalable, reliable and easily managed applications for the cloud. Service Fabric addresses the significant challenges in developing and managing cloud applications. By using Service Fabric, developers and administrators can avoid having to solve complex infrastructure problems and focus instead on implementing mission-critical, demanding workloads knowing that they're scalable, reliable and manageable. Service Fabric represents Microsoft's next-generation middleware platform for building and managing these enterprise class, Tier-1 cloud scale services.

Service Fabric is a universal deployment environment; you're able to deploy any executable based on any language (Microsoft .NET Framework, Node.js, Java, C++) or even database runtimes like MongoDB.

Therefore, it's important to make clear that Azure Service Fabric is not limited to microservices-oriented applications. You can also use it to host and deploy traditional applications (Web apps or services) and get many benefits related to scalability, load balancing and fast deployment. Yet Azure Service Fabric is a new platform built from the ground up and especially designed for hyperscale and microservices-based systems, as shown in **Figure 3**.

Here are some of the benefits of Service Fabric:

- **Runs on Azure, on-premises or in any cloud.** A very important characteristic of Service Fabric is that you can run it on Azure, but also on-premises, on your own bare-metal servers or virtual machine (VMs), and even in other third-party hosted clouds. There's no lock-in to a specific cloud. You could even run a Service Fabric cluster in Amazon Web Services (AWS).
- **Supports Windows or Linux.** Currently (late 2015) Service Fabric supports Windows, but it will also support Linux and containers (Windows images and Docker images).
- **Fully vetted.** Service Fabric has been used for several years by Microsoft to power many of its cloud products.

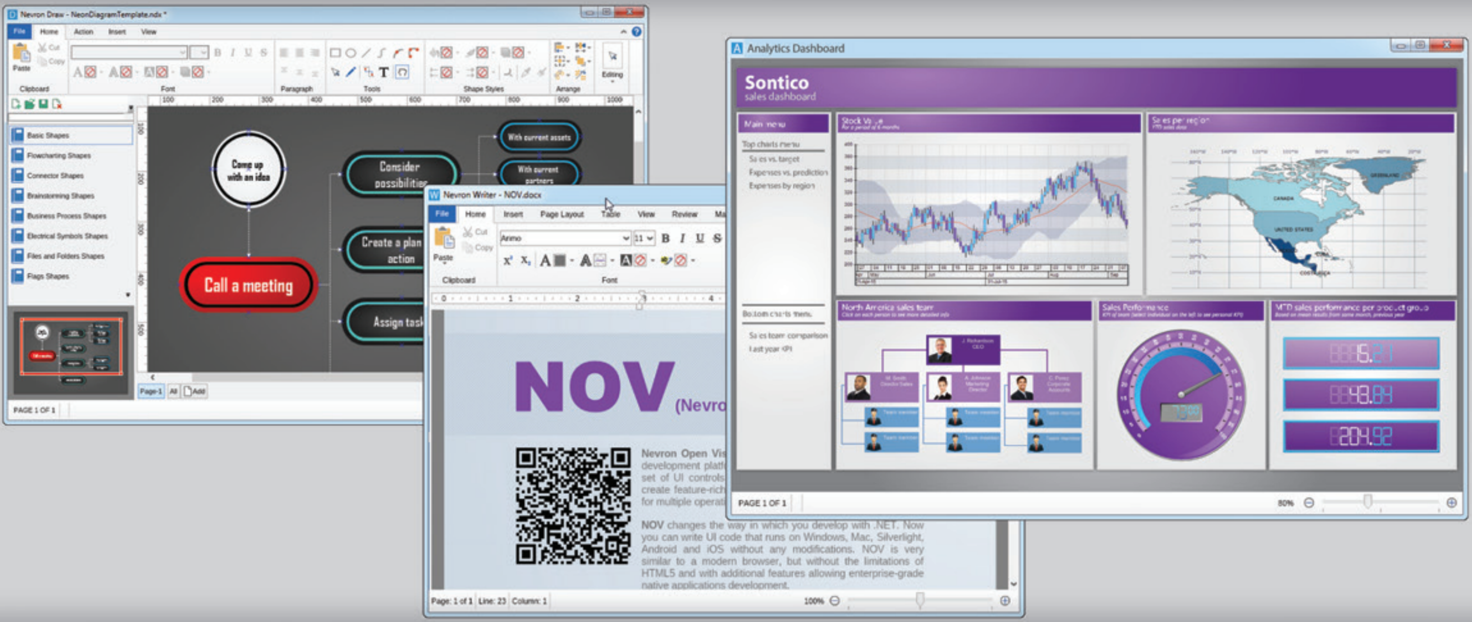
Service Fabric was born out of the need to build large-scale services within Microsoft. Taking products such as SQL Server and building them as services available in the cloud (Azure SQL Database) while being agile, reliable, scalable and cost-effective required a distributed technology that could meet all of these demands effectively. While the core technology that solved these complex scenarios was being built, it became apparent that SQL Server was not the only product that was making such a leap. Products such as Skype for Business had to solve similar problems on the path to becoming a microservices-based application. Service Fabric is the application platform that evolved out of these challenges and has been used in many large-scale services at Microsoft with varying architectures and requirements for running at scale. InTune, DocumentDB and the back end for Cortana and Skype for Business all are services that run on Service Fabric.

> What is Service Fabric? It's a distributed systems platform used to build hyper-scalable, reliable and easily managed applications for the cloud.

The experience supporting mission-critical systems has enabled Microsoft to design a platform that intrinsically understands the available infrastructure resources and needs of scalable applications. It enables the automatically updating, self-healing behavior that is essential to delivering highly available and durable services at hyper scale. Microsoft now makes this battle-hardened technology available for everyone.

Azure Service Fabric Programming Models

Service Fabric offers two high-level frameworks for building services: the Reliable Services APIs and the Reliable Actors APIs. While



Nevron Open Vision development platform set of UI controls create feature-rich for multiple operating

NOV changes the way in which you develop with .NET. Now you can write UI code that runs on Windows, Mac, Silverlight, Android and iOS without any modifications. NOV is very similar to a modern browser, but without the limitations of HTML5 and with additional features allowing enterprise-grade native applications development.

Figure 4 **Comparing Service Fabric Programming Models**

Reliable Actor APIs	Reliable Services APIs
Your scenario involves many small independent units/ objects of state and logic (live Internet of Things objects or gaming back-end scenarios are great examples)	You need to maintain logic and queries across multiple entity types and components
You work with a massive amount of single-threaded objects while still being able to scale and maintain consistency	You use reliable collections (like .NET reliable Dictionary and Queue) to store and manage your state/entities
You want the framework to manage the concurrency and granularity of state	You want to control the granularity and concurrency of your state
You want Service Fabric to manage the communication implementation for you	You want to decide on, manage and implement the communication protocols (Web API, WebSockets, Windows Communication Foundation and so on)
You want Service Fabric to manage the partitioning schema of stateful actor services so it's transparent for you	You want to control the partitioning scheme of your stateful service

both are built on the same Service Fabric core, they make different tradeoffs between simplicity and flexibility in terms of concurrency, partitioning and communication, as shown in **Figure 4**. It's useful to understand how both models work so that you can choose the framework better suited for your service. In many application scenarios, you can have a hybrid approach and use actors for certain microservices and employ reliable services to aggregate data generated by a number of actors. Therefore, a reliable service could orchestrate actor services in many common scenarios.

Building Stateless Microservices with Azure Service Fabric

A microservice in Azure Service Fabric can be almost any process in the server you want to create, whether it's the .NET Framework, Node.js, Java or C++. Currently only the .NET and C++ libraries of the Service Fabric API are available. Therefore, you need to implement microservices in the .NET Framework or in C++ for low-level implementations.

Figure 5 **Base Structure of Stateless Service in Azure Service Fabric**

```
using Microsoft.ServiceFabric.Services;
namespace MyApp.MyStatelessService
{
  public class MyStatelessService : StatelessService
  {
    //... Service implementation

    protected override async Task RunAsync(CancellationToken cancellationToken)
    {
      int iterations = 0;
      while (!cancellationToken.IsCancellationRequested)
      {
        ServiceEventSource.Current.ServiceMessage(this, "Working-{0}",
          iterations++);

        // Place to write your own background logic.
        // Example: Logic doing any polling task.
        // Logic here would be similar to what you usually implement in
        // Azure Worker Roles, Azure WebJobs or a traditional Windows Service.

         await Task.Delay(TimeSpan.FromSeconds(1), cancellationToken);
      }
      }
    }
}
```

As mentioned, a stateless microservice is one where there's a process (such as a front-end service or business-logic service), but there's literally no state persisted within the service, or the state that's present is lost when the process ends and doesn't require synchronization, replication, persistence or high availability. It could have a related external state, but persisted in external storage such as Azure SQL Database, Azure Storage, Azure DocumentDB or any other third-party storage engine (relational or NoSQL). Therefore, any existing service such as ASP.NET Web API service running on Cloud Services, Worker Role or Azure Web App would easily migrate to Service Fabric stateless services.

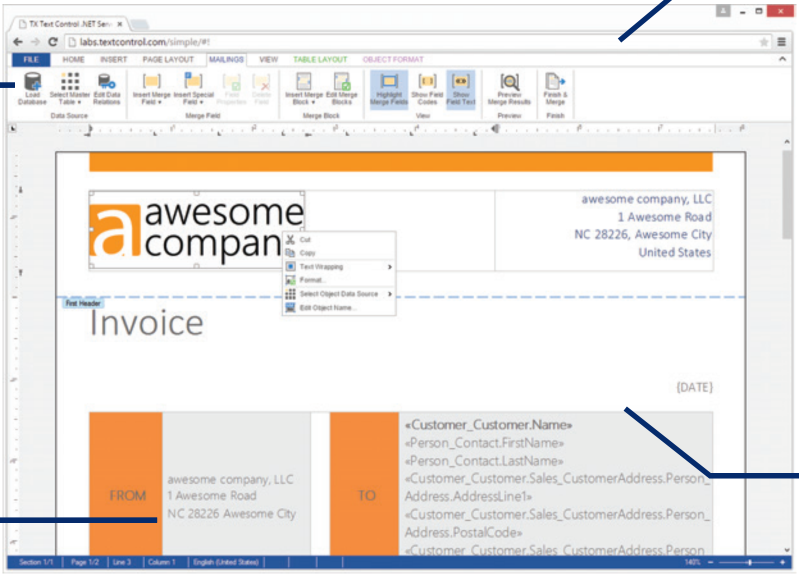
To set up your development environment, you need to have the Service Fabric SDK, which lets you run a local development cluster that's not an emulator, but runs the same bits as in Azure. Additionally, Visual Studio 2015 tooling integration makes development and debugging easier.

Before deploying and debugging services in your local machine, you need to create the cluster of nodes. You do that by running the Windows PowerShell script called DevClusterSetup.ps1, as explained in the "Install and Start a Local Cluster" section of the documentation page, "Prepare Your Development Environment," located at bit.ly/1Mfi0LB. You can refer to this page for a complete walk-through of the set up process.

Whether your service is stateful or stateless, Reliable Services provides a simple lifecycle that lets you quickly plug your code in and get started.

Stateless Service Base Class Any stateless Service Fabric service class has to derive from the Microsoft.ServiceFabric.Services.StatelessService class. This class provides API methods and context related to the Service Fabric cluster and the execution environment, and hooks into your service's lifecycle.

Whether your service is stateful or stateless, Reliable Services provides a simple lifecycle that lets you quickly plug your code in and get started. There's really just one or two methods that you need to implement to get your Service Fabric service up and running—typically RunAsync and CreateServiceReplicaListeners—which we'll explain when drilling down on the communication protocols.


labs.textcontrol.com/simple/#!
FILE
HOME
INSERT
PAGE LAYOUT
MAILINGS
VIEW
TABLE LAYOUT
OBJECT FORMAT
awesome company
awesome company, LLC
1 Awesome Road
NC 28226, Awesome City
United States
Invoice
{DATE}
FROM
TO
«Customer_Customer.Name»
«Person_Contact.FirstName»
«Person_Contact.LastName»

Note RunAsync as shown in **Figure 5**. This is where your service can "do work" in the background. The cancellation token that's provided is a signal for when that work should stop.

Communication Protocols in Azure Service Fabric Reliable Services Azure Service Fabric Reliable Services provides a pluggable communication model. You can use the transport of your choice, such as HTTP with ASP.NET Web API, WebSockets, Windows Communication Foundation, custom TCP protocols and so on.

You can implement your own code for your chosen communication protocol in your service or use any communication stack provided with Service Fabric, such as ServiceCommunication-Listener/ServiceProxy, which is the default communication stack provided by the Reliable Services Framework in Service Fabric. There will also be separate NuGet packages with additional communication stacks that you can plug into Service Fabric.

Using ASP.NET Web API in Service Fabric Microservices As mentioned before, Service Fabric lets you decide how you want your services to communicate, but one of the most common ways to create HTTP services in the .NET Framework is by using ASP.NET Web API.

Web API in Service Fabric is the same ASP.NET Web API you know and love. You use controllers, HttpRoutes and you can build REST services the same way by using the MapHttpRoute or attributes on the methods you want to map to Http routes. The difference when using Service Fabric is how you host a Web API application. First to take into account is that you cannot use IIS in Azure Service Fabric, as it uses just plain processes replicated across the cluster, so you have to self-host the HTTP listener in your code. In order to do that for Web API services, you have two choices:

- Use ASP.NET 5 Web API (code name "Project K") to self-host the HTTP listener within your Service Fabric microservices process.
- Use OWIN/Katana and ASP.NET 4.x Web API to self-host the HTTP listener within your Service Fabric microservices process.

Figure 6 **Adding the CreateServiceReplicaListeners Method to the Stateless Service Class**

```
using Microsoft.ServiceFabric.Services;
namespace MyApp.MyStatelessService
{
  public class MyStatelessService : StatelessService
  {
    //... Service implementation.

    protected override async Task RunAsync(CancellationToken cancellationToken)
    {
      // Place to write your own background logic here.
      // ...
    }

    protected override IEnumerable<ServiceReplicaListener>
      CreateServiceReplicaListeners()
      {
        return new List<ServiceReplicaListener>()
        {
          new ServiceReplicaListener(parameters =>
            new OwinCommunicationListener(
            "api", new Startup()))
        };
      }
    }
}
```

Running ASP.NET 5 on top of Azure Service Fabric is the best fit for building microservices because, as mentioned earlier, Service Fabric lets you deploy any number of services into each node/VM, allowing high microservices density per cluster. Additionally, that high density will get even better when Service Fabric eventually delivers .NET Core 5 and CoreCLR support in the future (underneath ASP.NET 5). It'll be "best of choice" to achieve super-high density of microservices because .NET Core 5 is a light framework with a very small memory footprint compared to the full .NET 4.x Framework.

Using "MVC-Type" Web Apps and Service Fabric Microservices ASP.NET MVC is a very popular framework for traditional Web sites, but you cannot use the "old" MVC framework (MVC 5 or older) with Service Fabric. This is due to the fact that in Service Fabric you need to self-host the HTTP listener in your process and that OWIN/Katana is not supported in ASP.NET 4.x MVC (It's supported only in ASP.NET 4.x Web API.) Therefore, the options you have to implement an "MVC-type" Web application on Service Fabric services are the following:

- Use MVC 6 in ASP.NET 5 to self-host the HTTP listener within your Service Fabric microservices process. It supports MVC because in ASP.NET 5, Web API and MVC are consolidated and are in fact the same framework with the same controllers with no dependency to IIS. This will be the preferred choice as soon as ASP.NET 5 is released to manufacture.
- Use ASP.NET 4.x Web API with OWIN/Katana to self-host the HTTP listener within your Service Fabric microservices process and simulate MVC controllers with Web API controllers and use HTML/JavaScript output rather than regular Web API content (JSON/XML). The documentation page at bit.ly/1UMdKIf demonstrates how to implement this technique.

Implement Self-Hosting with ASP.NET 4.x Web API and OWIN/Katana

For this approach, the Web API application you're used to working with doesn't change. It's no different from Web API applications you may have written in the past, and you should be able to simply move most of your application code right over. Hosting the application may be a little different from what you're used to if you've been hosting on IIS.

CreateServiceReplicaListeners Method in Service Class This is where the service defines the communication stacks it wants to use. The communication stack, such as Web API, is what defines the listening endpoints for the service, as well as how those messages that show up will interact with the rest of the service code.

Going back to the service class mentioned before in **Figure 4**, I now just have to add a new method called CreateServiceReplica-Listeners, which specifies at least one communication stack I want to use. In this case, it's the OwinCommunicationListener used by Web API, as shown in **Figure 6**.

It's important to highlight that you can have multiple communication listeners added to your service.

The OwinCommunicationListener class is boilerplate code that you'd reuse across all your microservices. In a similar way, you could be plugging in other protocols (WCF, WebSockets and so on).

If you're creating a Web API microservice, you still will have controllers with typical code no different to what you might be used to when implementing Web API services, like the following code:

```
// Typical Web API Service class implementation
public class CustomerController : ApiController
{
  //// Example - GET /customers/MSFT
  [HttpGet]
  [Route("customers/{customerKey}", Name = "GetCustomer")]
  public async Task<IHttpActionResult> GetCustomer(string customerKey)
  {
    // ... Stateless method implementation.
  }
}
```

Because this article is an end-to-end overview of Azure Service Fabric and microservices, we won't go through further steps to implement OWIN into your Service Fabric microservice. There's a simple example available on GitHub that you can download and analyze (bit.ly/1OW5Bmj): Web API and Service Fabric HelloWorld Example (ASP.NET 4.x).

Implement Self-Hosting with ASP.NET 5 Web API

Running ASP.NET 5 on top of Azure Service Fabric is the best fit for building microservices because Service Fabric lets you deploy any number of services into each node/VM, which allows for high microservices density per cluster. ASP.NET 5 also provides exciting features such as the following:

- Flexible hosting: You can now host your ASP.NET 5 application on IIS or in your own process. Hosting in your own process is fundamental in Azure Service Fabric.
- Web API, MVC and Web Pages: These are all merged together, simplifying the number of concepts.
- Full side-by-side support: ASP.NET 5 applications can now be installed on a machine without affecting any other applications on the machine that might be using a different .NET Framework version. This is a huge feature for IT management.
- Cross-platform support: ASP.NET 5 runs and is supported on Windows, Mac OS X and Linux.
- Cloud-ready: Features such as diagnostics, session state, cache and configuration are designed to work locally and in the cloud (like Azure) with no changes.

Looking forward, high-density capability will get even better when Service Fabric eventually delivers .NET Core 5 and CoreCLR support.

While support for .NET Core and CoreCLR remains in the works for Service Fabric, the benefit when that support comes online is clear. Running ASP.NET 5 on top of .NET Core 5 provides a lightweight .NET framework and CoreCLR runtime with a very small memory footprint compared to the traditional .NET 4.x. That combination will enable incredible microservices density, as shown in the "Microservice B" panel in **Figure 7**.

While .NET Core 5 and CoreCLR support is a future option, enterprises can prepare for it today by developing ASP.NET 5 services on top of .NET 4.x on Service Fabric. Doing so will make it easier to migrate to the CoreCLR in the future.

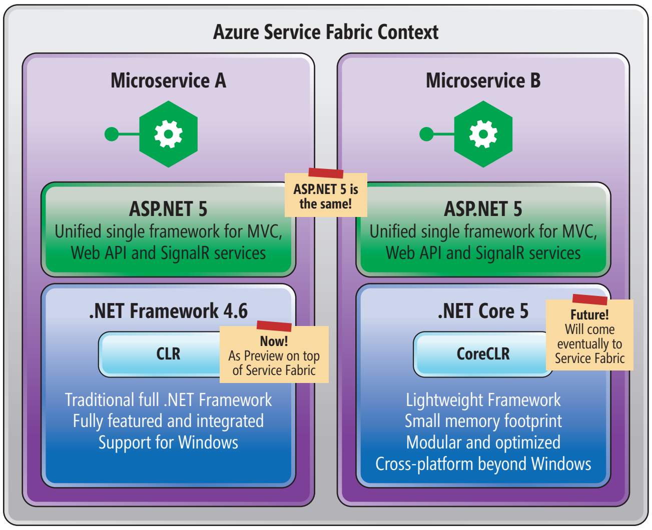

Azure Service Fabric Context
Microservice A
Microservice B
ASP.NET 5
Unified single framework for MVC, Web API and SignalR services
ASP.NET 5 is the same!
ASP.NET 5
Unified single framework for MVC, Web API and SignalR services
.NET Framework 4.6
CLR
Now! As Preview on top of Service Fabric
Traditional full .NET Framework
Fully featured and integrated
Support for Windows
.NET Core 5
CoreCLR
Future! Will come eventually to Service Fabric
Lightweight Framework
Small memory footprint
Modular and optimized
Cross-platform beyond Windows

Figure 7 **Comparing ASP.NET 5 Running on the CLR vs. CoreCLR Within the Service Fabric Context**

Operating and Deploying Hyper-Scale Microservices

The operations and deployment management provided by Service Fabric are other reasons why it's great for building and running in production hyper-scale microservices. You can focus on microservices development and let Service Fabric provide all the complex plumbing under the hood.

The high density of microservices is a great benefit if you care about lowering infrastructure cost. A Service Fabric cluster is composed of a pool of many VMs/servers (and containers in the future) called nodes in the cluster. In each node Service Fabric automatically deploys several services, so depending on the power of each server/VM you can have a super-high density of microservices across your cluster.

In Azure Service Fabric you have the ability to run hundreds of service instances on each machine or VM. That provides great savings for your Total Cost of Ownership (TCO). For the same

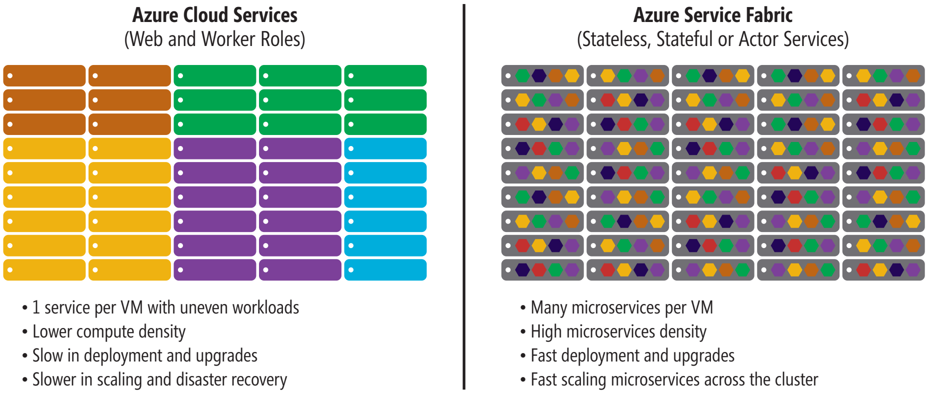
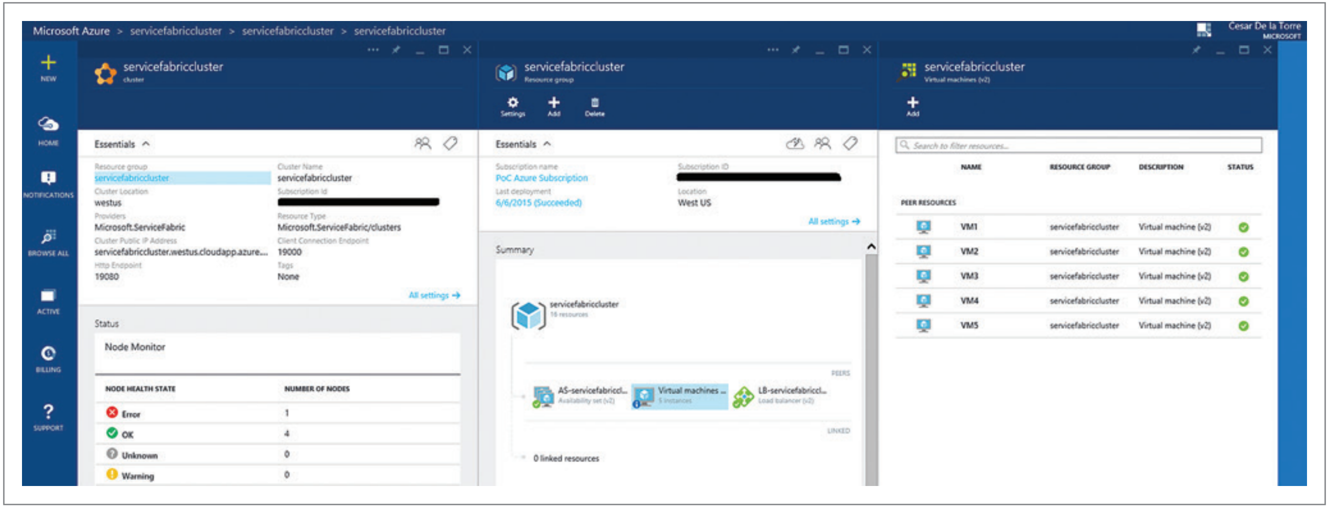

Azure Cloud Services
(Web and Worker Roles)
• 1 service per VM with uneven workloads
• Lower compute density
• Slow in deployment and upgrades
• Slower in scaling and disaster recovery
Azure Service Fabric
(Stateless, Stateful or Actor Services)
• Many microservices per VM
• High microservices density
• Fast deployment and upgrades
• Fast scaling microservices across the cluster

Figure 8 **Services Density Comparison—Azure Cloud Services vs. Service Fabric**


Microsoft Azure > servicefabriccluster > servicefabriccluster > servicefabriccluster
NEW
HOME
NOTIFICATIONS
BROWSE ALL
ACTIVE
BILLING
SUPPORT
Cesar De la Torre
MICROSOFT

servicefabriccluster
cluster

Essentials

Resource group: servicefabriccluster
Cluster Name: servicefabriccluster
Cluster Location: westus
Subscription Id: [redacted]
Providers: Microsoft.ServiceFabric
Resource Type: Microsoft.ServiceFabric/clusters
Cluster Public IP Address: servicefabriccluster.westus.cloudapp.azure....
Client Connection Endpoint: 19000
Http Endpoint: 19080
Tags: None

All settings →

Status

Node Monitor

NODE HEALTH STATE	NUMBER OF NODES
Error	1
OK	4
Unknown	0
Warning	0

servicefabriccluster
Resource group

Settings Add Delete

Essentials

Subscription name: PoC Azure Subscription
Subscription ID: [redacted]
Last deployment: 6/6/2015 (Succeeded)
Location: West US

All settings →

Summary

servicefabriccluster
16 resources

PEERS

AS-servicefabriccl... Availability set (v2)
Virtual machines ... 5 instances
LB-servicefabriccl... Load balancer (v2)

LINKED

0 linked resources

servicefabriccluster
Virtual machines (v2)

Add

Search to filter resources...

NAME	RESOURCE GROUP	DESCRIPTION	STATUS
PEER RESOURCES			
VM1	servicefabriccluster	Virtual machine (v2)	✓
VM2	servicefabriccluster	Virtual machine (v2)	✓
VM3	servicefabriccluster	Virtual machine (v2)	✓
VM4	servicefabriccluster	Virtual machine (v2)	✓
VM5	servicefabriccluster	Virtual machine (v2)	✓

Figure 9 **Service Fabric Cluster in the Azure Portal**

amount of services, you'll need less hardware. Therefore, high density is a big differentiator when comparing Azure Service Fabric with Azure Cloud Services, where you're limited to one VM per Service. If you were deploying microservices as a Web Role or Worker Role you probably had too many resources assigned to a single microservice, yielding slowdowns when deploying and managing. For instance, in **Figure 8** you can see one VM per instance of service in the form of Azure Web Role or Worker Role (color indicates type of service while each shape or box indicates a different VM and service instance). This approach doesn't help if you're looking to have high density of microservices, unless you use small VMs. Still, there's a limit and you won't be able to reach the same level of high density in either deployment compared to Azure Service Fabric.

By contrast, in Service Fabric you can deploy any number of microservices per node, so the density of services is far more efficient and cost is reduced. You can have tens, hundreds or even thousands of VMs/servers per cluster, with tens or hundreds of microservice instances and replicas per node/VM. And because each microservice is simply a process, the deployment and scaling is a lot faster than creating a new VM per service, as is the case with Azure cloud services.

Creating a Cluster in Azure's Cloud Before deploying microservices, you need to create the cluster of nodes in Azure or on-premises servers. When creating the cluster in the Azure cloud (like your production or staging environment), you can work directly from the Azure Portal or using Azure Resource Manager (ARM). In **Figure 9**, you see a Service Fabric cluster we created in our Azure Subscription by using the Azure Portal. Under the hood, the cluster creation process is based on Azure ARM and Azure Resource Groups. In this case, we created a cluster with five nodes/VMs that are also placed within an Azure Resource Group.

Deploying Applications/Services to the Cluster Once the cluster of nodes/VMs is up and running, you can deploy services. When deploying to production clusters you usually use a Windows PowerShell script to deploy your apps/services to the cluster in Azure, although for staging/test environments you can also deploy directly from Visual Studio.

When deploying to a local cluster on your development PC using Visual Studio 2015, you would usually deploy directly from the IDE by right-clicking on the Service Fabric application project and selecting Deploy. You can also use Windows PowerShell to deploy to the development cluster in your laptop, because the same Service Fabric bits run in development clusters as in Azure cloud-based clusters.

Day-to-day operations and manageability of deployments is necessary to keep services running smoothly in the long run, and application lifecycle management features have been built into Service Fabric, keeping the microservices-based approach in mind. The operations and management features available in Service Fabric include fast deployments, zero-downtime application upgrades, health monitoring of the services, and cluster scale up and scale down. Zero-downtime upgrades are possible because the upgrade tech in Service Fabric provides a built-in combination of rolling upgrades and automatic health checks that detect and roll back the changes in the upgrade if they destabilize the application. Management of Service Fabric clusters and applications is possible through Windows Powershell commands, providing all the power of the CLI and scripting, while also supporting visual tools including Visual Studio for ease of use.

Rolling upgrades are performed in stages. At each stage, the upgrade is applied to a subset of nodes in the cluster, called an upgrade domain. As a result, the application remains available throughout the upgrade. You can also have strong versioning and side-by-side support so you can deploy v1 and v2 versions of the same microservice and Service Fabric will redirect to one or the other, depending on the client's requests. Check the documentation at bit.ly/1kSupz8 for further details.

When debugging services within your local development cluster within your PC, Visual Studio makes it simple even when the services' processes are already running before you start debugging. The IDE automatically attaches to all the microservices processes related to your project, making it easy to get started and use the regular breakpoints in your Service Fabric microservices code. Just set the breakpoints in your code and hit F5. You don't need to find out to what process you need to attach Visual Studio.

Spreadsheets Made Easy.

Fastest Calculations

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

WIN Windows Forms | Silverlight | WPF WPF

Powerful Controls

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

Comprehensive Charting

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

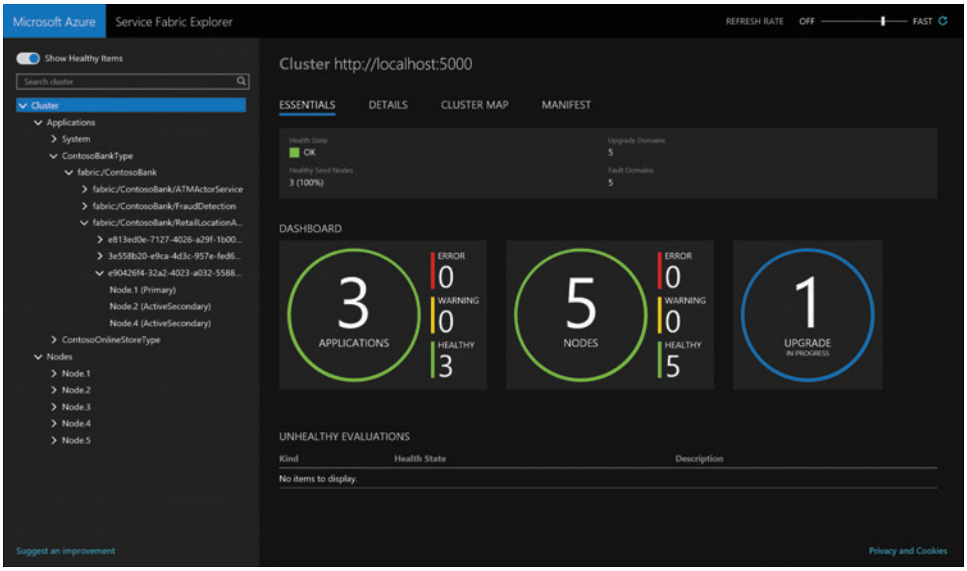
Scalable Reporting

Easily create richly formatted Excel reports without Excel from any ASP.NET, Windows Forms, WPF or Silverlight application.

Download your free fully functional evaluation at SpreadsheetGear.com

SpreadsheetGear

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



Show Healthy Items

Search cluster

- Cluster
  - Applications
    - System
    - ContosoBankType
      - fabric:/ContosoBank
        - fabric:/ContosoBank/ATMActorService
        - fabric:/ContosoBank/FraudDetection
        - fabric:/ContosoBank/RetailLocationA...
          - e813ed0e-7127-4026-a29f-1b00...
          - 3e558b20-e9ca-4d3c-957e-fed6...
          - e90426f4-32a2-4023-a032-5588...
            - Node.1 (Primary)
            - Node.2 (ActiveSecondary)
            - Node.4 (ActiveSecondary)
    - ContosoOnlineStoreType
  - Nodes
    - Node.1
    - Node.2
    - Node.3
    - Node.4
    - Node.5

Cluster http://localhost:5000

Health State: OK
Healthy Seed Nodes: 3 (100%)
Upgrade Domains: 5
Fault Domains: 5

3 APPLICATIONS	ERROR 0 WARNING 0 HEALTHY 3	
5 NODES	ERROR 0 WARNING 0 HEALTHY 5	
1 UPGRADE IN PROGRESS		

UNHEALTHY EVALUATIONS

Kind	Health State	Description
No items to display.		

Figure 10 **The Service Fabric Explorer**

Service Fabric Explorer Service Fabric Explorer, depicted in **Figure 10**, is a Web-based tool provided by the cluster to visualize the state of deployed applications, inspect the contents of individual nodes and perform various administrative actions. The Explorer tool is served from the same HTTP Gateway service that supports the Service Fabric REST APIs and can be reached by navigating to http://<your-cluster-endpoint>:19007/Explorer. In the case of a local cluster, the URL would be http://localhost:19007/Explorer.

For further details on Service Fabric Explorer, read the documentation page, "Visualizing Your Cluster Using Service Fabric Explorer" (bit.ly/1MUNyad).

The Service Fabric platform enables a range of features that allow you to focus on application development rather than implementing complex health and upgradable systems. Among them:

Scaling out Stateless Services The Service Fabric orchestration engine can automatically scale your Web app across more machines whenever new nodes are added to a cluster. When creating instances of a stateless service, you can specify the number of instances you want to create. Service Fabric will place that number of instances on nodes in the cluster accordingly, making sure not to create more than one instance on any one node. You can also instruct Service Fabric to always create an instance on every node by specifying "-1" for the instance count. This guarantees that whenever you add nodes to scale out your cluster, an instance of your stateless service will be created on the new nodes.

Automatic Resource Balancing Service Fabric provides service resource balancing (or service orchestration) that understands the total available resources in the cluster (like the compute from the VMs). It automatically moves the microservices depending on defined policies and constraints to best optimize how the created services are placed across the Vms (see **Figure 11**). This is focused on optimizing costs and performance.

Built-in Failover and Replication Machines in any datacenter or public clouds are prone to unplanned hardware failures. To guard against such scenarios Service Fabric provides built-in failover and replication, which means that despite hardware problems the availability of services is not affected. This is possible because Service Fabric can run and manage multiple instances of each service across machines and failure domains.

Placement Constraints You can instruct the cluster to do things like not place the front-end microservices in the same machines/nodes as the middle-tier microservices, and to avoid collocating certain types of microservices in the same nodes. This can be done to minimize known conflicts or to ensure priority access to resources for certain microservices.

Load Balancing of Requests Not to be confused with automatic resource balancing, load balancing requests are not handled by Service Fabric, but rather by the Azure Load Balancers or other platform external to Service Fabric.

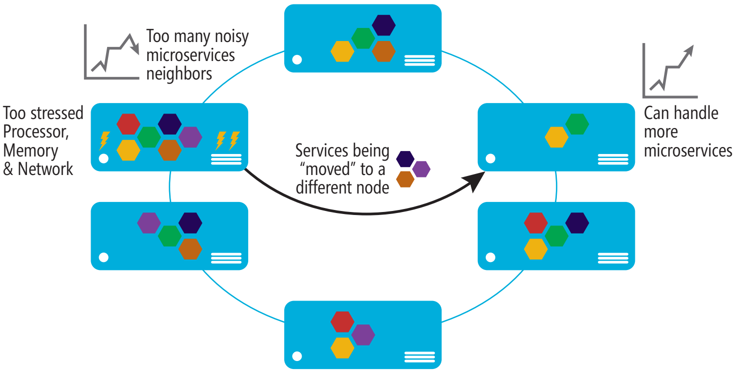

Too many noisy microservices neighbors
Too stressed Processor, Memory & Network
Services being "moved" to a different node
Can handle more microservices

Figure 11 **Cluster Showing Services Distribution Across the Nodes and Automatic Resource Balancing**

Health Allows you to monitor and diagnose your systems. Also used during application upgrades to provide safety checks so upgrades can be rolled back if the upgrade has a destabilizing effect. For more information, see the documentation page, "Introduction to Service Fabric Health Monitoring" (bit.ly/1jSvmHB).

Stateful Microservices in Azure Service Fabric

Support for stateful services is an exciting and important component of Azure Service Fabric. It's also complex and wide-ranging enough that exploring stateful services goes beyond the scope of this article, but we'll briefly explain what it is. Look for a follow-on article focused on this area of Service Fabric in a future issue.

A stateful microservice in Service Fabric collocates compute and data, with the state placed within the microservice itself (both in-memory and persisted on local disk). The reliability of state is achieved through local persistence and replication of data to other nodes/VMs. Basically, each data partition has several replica services associated with it. Each replica can be deployed in a different node/VM to provide high availability should the primary replica go down. This is similar to how Azure SQL Database works with database replicas, because it's based on Azure Service Fabric.

In complex and scalable applications, stateful services simplifies the architecture and reduces the number of components compared to a traditional, three-tier architecture that needs to use external cache and queues. When using stateful services, the benefits of a traditional external cache is now intrinsic in the stateful service. Additionally, external queues aren't needed because we can implement internal queues within the Service Fabric microservices.

When using stateful services, it automatically creates hot backup secondaries that can pick up the operations from the same point where a primary left off following a hardware failure. Many times services have to scale to continue to meet the demands of a growing user base, requiring the addition of hardware to the running environment. Service Fabric uses features such as partitioning so that services can be built in a way that they automatically spread over onto new hardware without requiring user intervention.

Stateful Azure Reliable Services provide a host of capabilities, including data partition support, replica support and leader election, replica service naming, and service address discovery from the gateway service. It also provides management of concurrency and granularity of state changes using transactions, the ability to maintain consistent state replication, and the use of reliable, distributed key/value collections (Reliable Dictionary and Reliable Queue). The good news is that Service Fabric handles the complexity and details of these topics for you, so you can focus on writing applications.

Reliable Actors Services in Azure Service Fabric

Azure Service Fabric Reliable Actors is an actor programming model for Service Fabric. It provides an asynchronous, single-threaded actor model. An actor represents a unit of state and computation. In some ways it's similar to the open source software project "Orleans" created by Microsoft Research. The important point is that Service Fabric Reliable Actors API is based on the underlying infrastructure provided by Service Fabric.

Implementing actor-instances for Internet of Things (IoT) devices is a good example of actor services usage. A Vehicle-Actor type in the form of a C# class would encapsulate the IoT vehicle domain logic plus its live states like GPS coordinates and other data. Then, we would have potentially millions of actor objects or instances of that mentioned class, distributed across many nodes in the cluster.

Of course, actors are not limited to live IoT objects—they could be used for any subject, but "live IoT objects" is a great didactic scenario.

Actors are distributed throughout the cluster to achieve high scalability and availability and are treated as in-memory objects in every cluster's VM. They are also persisted on local disk and replicated through the cluster.

The actors are isolated single-threaded objects that encapsulate both state and behavior. Every actor is an instance of an actor type, similar to the .NET code shown here:

```
// Actor definition (State+Behaviour) to run in the back end.
// Object instances of this Actor class will be running transparently
// in the service back end.

public class VehicleActor : Actor<Vehicle>, IVehicleActor
{
  public void UpdateGpsPosition(GpsCoordinates coord)
  {
    // Update coordinates and trigger any data processing
    // through the State property on the base class Actor<TState>.
    this.State.Position= coord;
  }
}
```

Next, the following code shows example client code to use the actor via a proxy object:

```
// Client .NET code
ActorId actorId = ActorId.NewId();
string applicationName = "fabric:/IoTVehiclesActorApp";
IVehicleActor vehicleActorProxy =
  ActorProxy.Create<IVehicleActor>(actorId, applicationName);
vehicleActorProxy.UpdateGpsPosition(new GpsCoordinates(40.748440, -73.984559));
```

All the communication infrastructure is taken care of under the hood by the Service Fabric Reliable Actors framework.

You could have millions of VehicleActor objects running on your cluster across many different nodes/VMs, and Service Fabric would be taking care of the required plumbing, including partitions and replicas where your millions of actors are spread and balanced.

The Actors client API provides communication between an actor instance and an actor client. To communicate with an actor, a client creates an actor proxy object that implements the actor interface. The client interacts with the actor by invoking methods on the proxy object.

For the scenarios where actors fit, it greatly simplifies the microservices implementation, especially when compared to stateful reliable services where you have more control but need to implement a lot of plumbing related to partitioning data, partition addressing, replicas and the like. If you don't need fine-grain control, then you can avoid the extra work by using Reliable Actors.

Wrapping Up

Microservices architectures require a commitment to decentralized approaches, disciplined architecture and design processes, new technologies like Azure Service Fabric, and a change in team dynamics toward small development teams and Agile principles applied per microservice. ■

Cesar de la Torre *is a senior program manager at Microsoft and lives in Redmond, Wash. His interests include Microsoft Azure and .NET development with approaches such as microservices architectures and domain-driven design, as well as mobile apps development surfacing the services.*

Kunal Deep Singh *is a senior program manager on the Microsoft Azure Service Fabric team. Prior to Azure he worked on multiple releases of Windows Phone, Silverlight and as a game developer for Xbox titles. He lives in Seattle, Wash.*

Vaclav Turecek *is a senior program manager at Microsoft. He works tirelessly with a very talented group of people to make Azure Service Fabric the best next-generation Platform as a Service.*

Thanks to the following Microsoft technical expert for co-authoring and reviewing this article: *Mark Fussell*

Babylon.js: Building a Basic Game for the Web

Raanan Weber

Babylon.js is a WebGL-based 3D engine that focuses mainly on game development and ease of use. As a 3D engine, it has the tools to create, display, and texture meshes in space, and to add light sources and cameras. Because it's game-focused, Babylon.js has some extra features that a regular 3D engine doesn't require. It has native support for collision detection, scene gravity, game-oriented cameras (for example, a follow-camera that tracks a moving object), as well as native support for Oculus Rift and other virtual reality (VR) devices. It has a physics engine plug-in system, native audio support, a user input-based action manager and much more. I'll explore all of these features in this tutorial.

About the Tutorial

In this tutorial I'll develop a simple bowling game. I'll create a bowling lane, add 10 pins and a bowling ball, and provide the ability to throw the ball. My game will surely not be ready to be released, but it will show how you can develop a game using the tools Babylon.js provides. I'll deliberately avoid using any external tools during development. The objects, cameras, textures and more will be created using JavaScript code only.

The Babylon.js version I'll be using during this tutorial is the latest stable release—2.1. David Catuhe and the Babylon development team are trying to keep the framework as backward-compatible as possible, so I can only assume that this tutorial will work properly on future releases, at least until the next major release. And I'm using Visual Studio 2015 Community Edition, but you can use any IDE you'd like.

The tutorial will be divided into two parts. In this article, I'll present an overview of the basic building blocks of Babylon.js. I'll create the meshes, texture them, add cameras and light sources, and enable simple user interaction. In part two, I'll show where Babylon.js really shines by adding collision and physics, audio, user actions, and special types of cameras.

This article discusses:

- Programming a simple game for the Web using Babylon.js and WebGL
- Creating meshes and adding textures to them
- Adding cameras and light sources
- Enabling simple user interaction

Technologies discussed:

Visual Studio 2015 Community Edition, Babylon.js

Code download available at:

msdn.com/magazine/msdnmag1215

Getting Started: A New Babylon.js Project

A simple Babylon.js project is a static Web site. Because I'm using Visual Studio, I'll use the local IIS server embedded in Visual Studio to host those static files. Visual Studio doesn't have a template for a static Web site, so a different approach is needed.

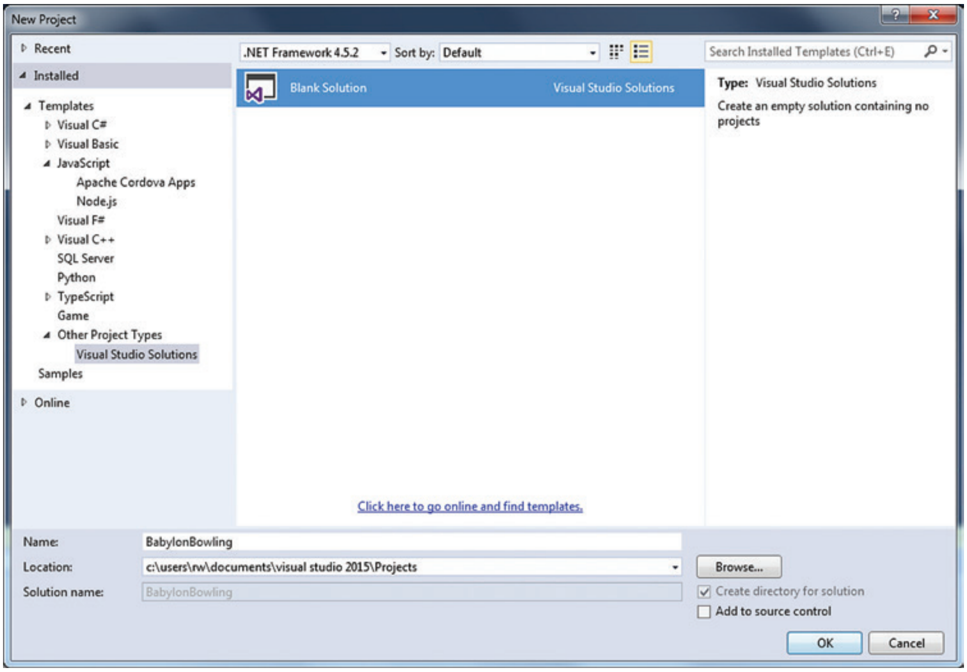
First, create a new blank solution. Open Visual Studio and go to File | New | Project. Select Other Project Types on the left pane and then Blank Solution, as shown in **Figure 1**. Give the solution a name (I used BabylonBowling) and click OK.

To create a new static Web site, first you need to create a new directory to host it. Right-click on the empty solution, then click on Open Folder in File Explorer, as shown in **Figure 2**.

Create a new directory for the project called BabylonBowling and close File Explorer.

Right-click on the solution and choose Add | Existing Website. Select the newly created folder (make sure you choose the folder you just created and not the solution's folder) and click Open. You should now have a blank Web site as the only project of this solution.

After creating the project, you need to add the framework and the framework's dependencies. There are a few ways to do that. The simplest way is to use the NuGet Package Manager.



Type: Visual Studio Solutions

Create an empty solution containing no projects

Name: BabylonBowling

Location: c:\users\rw\documents\visual studio 2015\Projects

Solution name: BabylonBowling

☑ Create directory for solution

□ Add to source control

Figure 1 **Creating a New Blank Solution in Visual Studio**

Right-click on the project and choose Manage NuGet packages. Click in the search field (the keyboard shortcut is Ctrl+E) and type babylon. You'll see a window similar to the one in **Figure 3**. Select BabylonJS. Make sure the Version selected is 2.1 (or the latest stable version) and click Install. Click OK in the Preview window that pops up (if one pops up) and Babylon.js will be installed into your empty project, including a demo scene.

For those using npm as a package manager, you can install Babylon.js using the command:

```
npm install babylonjs
```

After installing the package, you should have the following files in the scripts folder:

- babylon.js, a minified version of Babylon.js
- babylon.max.js, a debug version of Babylon.js
- Oimo.js, the Oimo JS physics engine that will be used in part two of this tutorial
- poly2tri.js, an optional triangulation library (github.com/r3mi/poly2tri.js)
- hand-minified.js, a pointer-events polyfill; this is missing when using npm, so install using the command:

```
npm install handjs
```

The NuGet installer also creates an index.html file and an index.js file.

An error in the NuGet package adds an unneeded line to web.config. Until it's fixed, double-click on this file and remove the line highlighted in **Figure 4** (in my solution, it's line 9).

You can now test the project. Open the project in your default browser using Ctrl+Shift+W or by clicking the Run button on the top navbar. If you see the 3D spaceship shown in **Figure 5**, your project is all set.

For those using different IDEs, just follow the "Creating Basic Scene" tutorial on the Babylon.js Documentation page to get to the current state. I still recommend using the npm to install the dependencies if not using NuGet. The tutorial can be found at **bit.ly/1MXT6WP**.

To start from scratch, delete the function createScene in index.js.

The Building Blocks

The first two elements I'll discuss are the engine and the scene.

The engine is the object responsible for communicating with the low-level WebGL API (WebGL 1 is based on OpenGL ES2, with very similar syntax). Instead of forcing you to write low-level WebGL code, the engine provides a higher-level and easier-to-understand API. It's also transparent to the developer. Except for the initial project setup, I won't directly use the engine at all. Certain lower-level functionality can be accomplished using the engine, but I won't be covering this.

The engine requires two parameters in order to initialize. The first is a canvas on which to draw. The canvas is an HTML element that's already in index.html. The second parameter is the state of anti-aliasing (on or off).

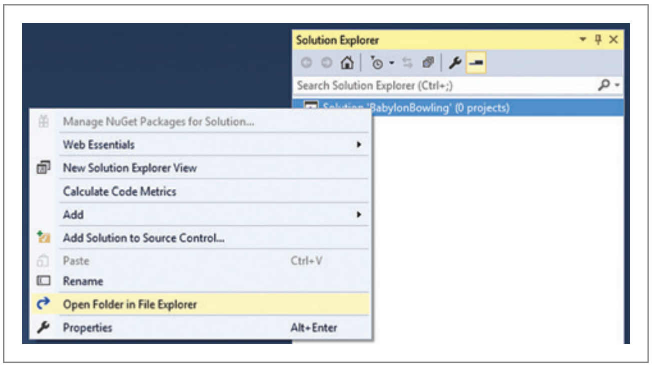
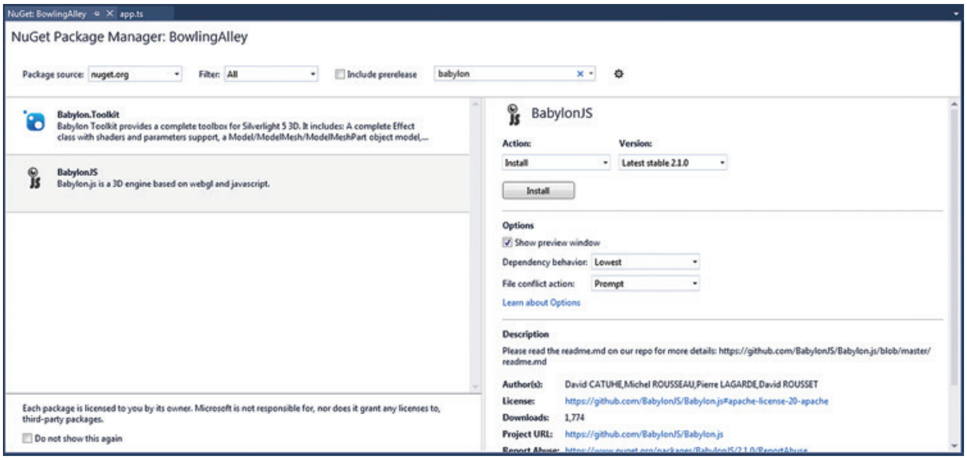

Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'BabylonBowling' (0 projects)
Manage NuGet Packages for Solution...
Web Essentials
New Solution Explorer View
Calculate Code Metrics
Add
Add Solution to Source Control...
Paste
Ctrl+V
Rename
Open Folder in File Explorer
Properties
Alt+Enter

Figure 2 **Opening the Solution's Folder in File Explorer**



NuGet: BowlingAlley ⊕ ✕ app.ts

NuGet Package Manager: BowlingAlley

Package source: nugetorg ▾ Filter: All ▾ □ Include prerelease babylon ✕ ▾

Babylon.Toolkit
Babylon Toolkit provides a complete toolbox for Silverlight 5 3D. It includes: A complete Effect class with shaders and parameters support, a Model/ModelMesh/ModelMeshPart object model,...

BabylonJS
Babylon.js is a 3D engine based on webgl and javascript.

Each package is licensed to you by its owner. Microsoft is not responsible for, nor does it grant any licenses to, third-party packages.

□ Do not show this again

BabylonJS

Action: Install ▾ Version: Latest stable 2.1.0 ▾

Install

Options

☑ Show preview window

Dependency behavior: Lowest ▾

File conflict action: Prompt ▾

Learn about Options

Description

Please read the readme.md on our repo for more details: https://github.com/BabylonJS/Babylon.js/blob/master/readme.md

Author(s): David CATUHE,Michel ROUSSEAU,Pierre LAGARDE,David ROUSSET

License: https://github.com/BabylonJS/Babylon.js#apache-license-20-apache

Downloads: 1,774

Project URL: https://github.com/BabylonJS/Babylon.js

Report Abuse: https://www.nuget.org/packages/BabylonJS/2.1.0/ReportAbuse

Figure 3 **Adding Babylon.js Using the NuGet Package Manager**

Open the currently blank index.js and add the following lines of code:

```
function init() {
  var engine = initEngine();
}
function initEngine() {
  // Get the canvas element from index.html
  var canvas = document.getElementById("renderCanvas");
  // Initialize the BABYLON 3D engine
  var engine = new BABYLON.Engine(canvas, true);
  return engine;
}
```

During development, I'll use the init function to add new functionality with each step. Each element I create will have its own function.

To understand what a scene represents, think of a Web site with different pages. A single Web page contains the text, images, event listeners, and all other resources needed to render that one, single page. Loading a different page loads different resources.

> Just as with a single Web page, the scene holds the needed resources to show one, single 3D "page."

Just as with a single Web page, the scene holds the needed resources to show one, single 3D "page." This scene can be very large and full of resources—meshes, cameras, lights, user actions and more. But in order to move from one page to another, a new scene is recommended. The scene is also responsible for rendering its own resources and communicating the needed information to the engine. The bowling game requires only a single scene. But if I planned to add new levels or a bonus game, I'd create them using new scenes.

To initialize the scene requires only the engine I created. Add the following to index.js:

```
function createScene(engine) {
  var scene = new BABYLON.Scene(engine);
  // Register a render loop to repeatedly render the scene
  engine.runRenderLoop(function () {
      scene.render();
  });
}
```

First, I create the scene object. The next lines are the only interaction I'll have with the engine. They tell the engine to render this specific scene every time the render loop is running.

Add this line of code to the end of the init function to actually create the scene:

```
var scene = createScene(engine);
```

Two more things before I continue. When you resize the browser window, the canvas is also being resized. The engine must also resize its internal width and height to keep the scene in perspective. Adding the following lines to the initEngine function right before the return engine; statement will keep the scene in perspective:

```
// Watch for browser/canvas resize events
window.addEventListener("resize", function () {
  engine.resize();
});
```

The second thing is to alter index.html to use the new init function. Open index.html and find the script tag containing the call to createScene. Change createScene to init, then save index.html and close it.

The scene has an extremely advanced debug layer that allows you to debug the scene being rendered. It shows the number of meshes being rendered and the current frames per second (FPS). It allows turning off features such as texturing and shadows and makes it easy to find lost meshes. Turning the debug layer on is simple:

```
scene.debugLayer.show();
```

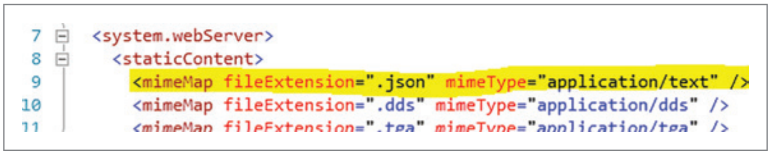
Do that, and you'll be able to debug your scene on your own.

I now have an engine and a scene, and I'm ready to start adding a camera, lighting and meshes to create my Bowling Alley scene.

Cameras

Babylon.js offers many types of cameras, each with its own specific purpose. Before choosing the camera for the game, let's review the most common types:

- *Free Camera* is a first-person-shooter camera. It can freely move throughout the scene, using the keyboard arrow keys, and the direction can be set using the mouse. Optional gravity and collision detection can be enabled, as well.
- *Arc Rotate Camera* is used to rotate around a specific target. Using the mouse, the keyboard or touch events, the user can view the object from all directions.
- *Touch Camera* is a free camera that uses touch events as input. It's suitable for all mobile platforms.
- *Follow Camera* automatically follows a specific target.



```
<system.webServer>
  <staticContent>
    <mimeMap fileExtension=".json" mimeType="application/text" />
    <mimeMap fileExtension=".dds" mimeType="application/dds" />
    <mimeMap fileExtension=".tga" mimeType="application/tga" />
```

Figure 4 **Delete the Highlighted Line from web.config**

Extreme Performance Linear Scalability

Cache data, reduce expensive database trips, and scale your apps to extreme transaction processing (XTP) with NCache.

In-Memory Distributed Cache

- Extremely fast & linearly scalable with 100% uptime
- Mirrored, Replicated, Partitioned, and Client Cache
- Entity Framework & NHibernate Second Level Cache

ASP.NET Optimization in Web Farms

- ASP.NET Session State storage
- ASP.NET View State cache
- ASP.NET Output Cache provider

Full Integration with Microsoft Visual Studio

- NuGet Package for NCache SDK
- Microsoft Certified for Windows Server 2012 R2

Open Source

Celebrating 10 Years of Market Leadership!

NCache™

FREE Download

sales@alachisoft.com US: +1 (925) 236 3830 www.alachisoft.com

Babylon.js supports WebVR and device-orientation cameras natively, which means you can easily address such devices as the Oculus Rift or Google Cardboard.

A new concept introduced in version 2.1 makes each camera type 3D-ready. This means that each camera type can be set to fit Oculus Rift-style stereoscopic view and red-blue glasses (anaglyph 3D). **Figure 6** shows shows the spaceship scene rendered with the anaglyph-free camera and **Figure 7** shows the scene with the stereoscopic camera.

For my bowling game, I'll use two types of cameras. The first is the player's main camera, which will set the position from which the bowling ball will be thrown. This is the exact purpose of the free camera. I also want to add a different view—when the ball is en route, I want to follow it until it hits the pins.

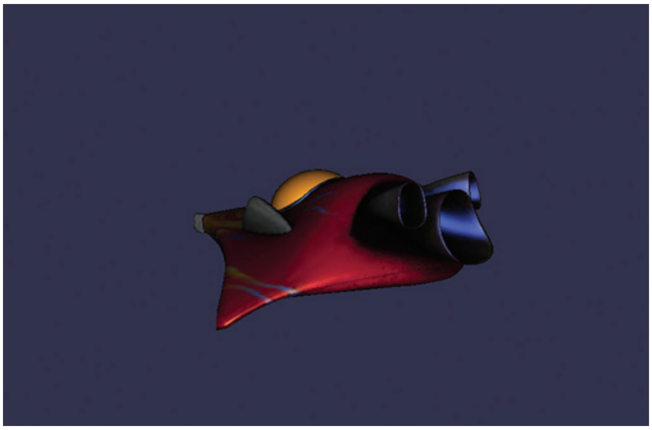


Figure 5 **The Babylon Default Spaceship**

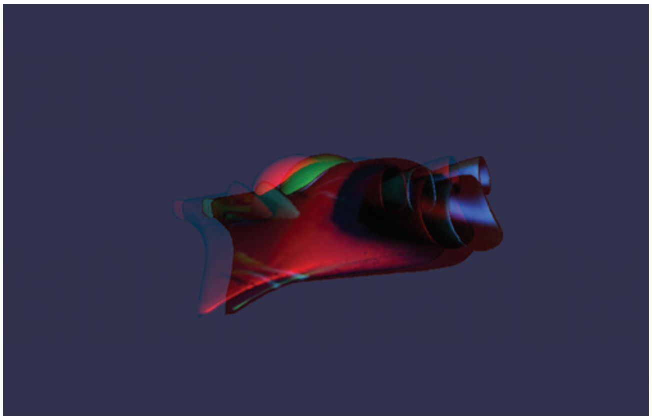


Figure 6 **Anaglyph 3D Camera**

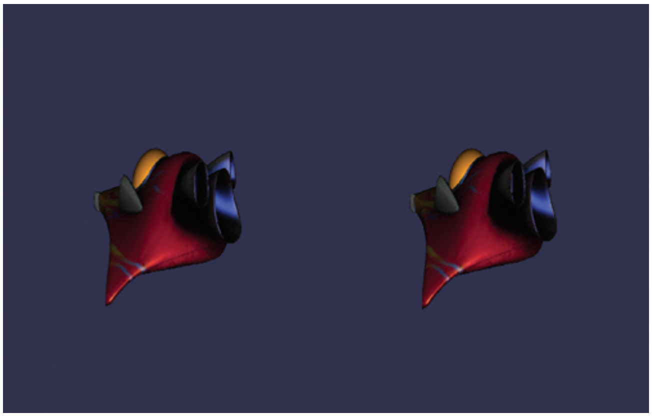


Figure 7 **Stereoscopic Camera**

First, I add the free camera. The createFreeCamera function takes the scene as a single variable:

```
function createFreeCamera(scene) {
  var camera = new BABYLON.FreeCamera(
    "player camera", BABYLON.Vector3.Zero(), scene);
  return camera;
}
```

I created a camera position at location 0,0,0 of my scene. Later, I'll extend this function (when necessary) to further configure the camera.

And, of course, don't forget to add it to the init function, at the end:

```
...
// Create the main player camera
var camera = createFreeCamera(scene);
// Attach the control from the canvas' user input
camera.attachControl(engine.getRenderingCanvas());
// Set the camera to be the main active camera
scene.activeCamera = camera;
```

The attachControl function registers the native JavaScript event listeners needed for the specific camera (such as those for mouse, touch or keyboard input). Setting the scene's active camera tells the scene that this camera should be used for rendering.

I'll add the second camera in part two, after enabling a ball throw.

Creating the Lane

A bowling lane is a relatively simple geometric structure. I'll start by setting some constants, which are the actual dimensions of a bowling lane in meters. The reason I use meters is the physics engine, which will be explained in the second part of this article.

You can see the constants in the project file. To calculate these values I used online-available information about bowling lanes.

Babylon.js supports WebVR and device-orientation cameras natively, which means you can easily address such devices as the Oculus Rift or Google Cardboard.

After setting the constants, I'm ready to start creating the meshes that will construct the bowling lane. My (2D) plan is shown in **Figure 8**. I'll start by creating the meshes. Texturing the meshes—giving them materials—will come afterward.

First, I'll create a global floor for the scene:

```
function createFloor(scene) {
  var floor = BABYLON.Mesh.CreateGround("floor", 100, 100, 1, scene,
false);
  return floor;
}
```

This creates a simple 100x100-meter floor. All meshes created using Babylon.js internal functions are created at position 0,0,0 of

Visual Studio MAGAZINE

READER'S CHOICE AWARD WINNER 2015

DevExpress®

Visual Studio MAGAZINE 1 READER'S 2015 CHOICE

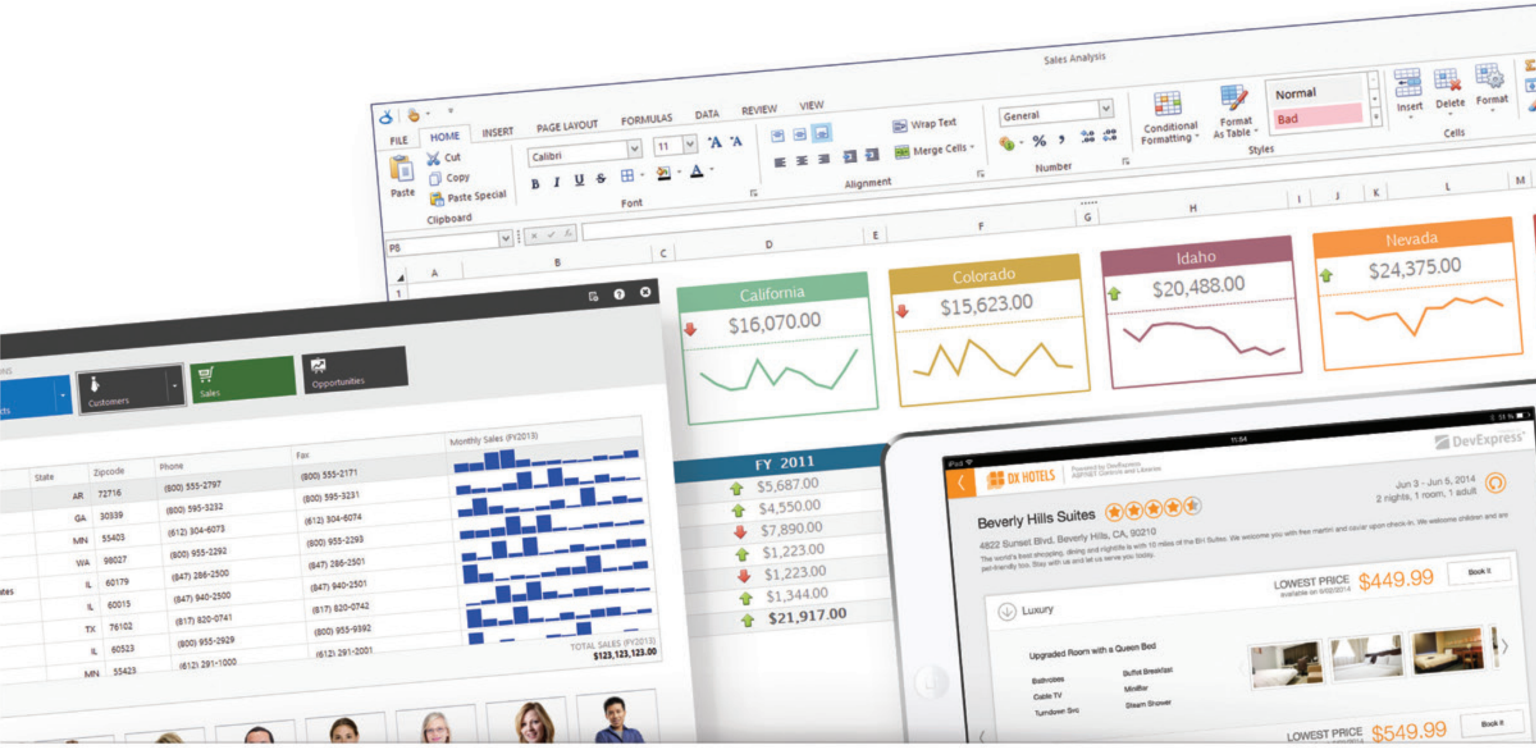
Thank You to VSM Subscribers!

With over 31 categories and 400 products to choose from, we are honored to have won the most awards in this year's VSM Readers' Choice.

"It's incredibly rewarding and humbling to have been chosen for a total of 18 gold and silver VSM Subscriber awards," said Julian Bucknall, CTO at DevExpress. "We want to thank all of our customers for the faith they've placed in our tools, their continued support and for taking the time to vote for our products."

If you've not had the opportunity to evaluate our award-winning tools and are ready to experience the DevExpress difference for yourself, download our free 30-day trial at **devexpress.com/try**.

WIN WPF ASP MVC HTML 5



To learn more, please visit our website → **www.devexpress.com**

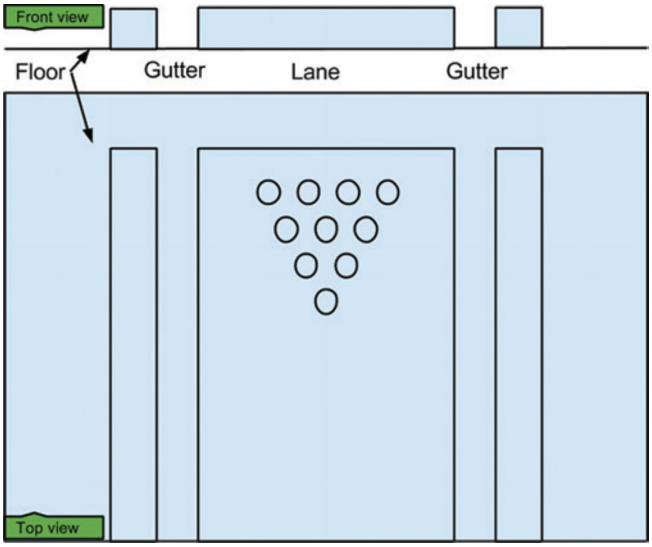


Figure 8 **A 2D Plan of the Bowling Lane Scene**

the scene and are centered. There are three important variables that transform the mesh and position it correctly in the scene:

- *mesh.position* is a vector of the mesh's position in space
- *mesh.scaling* is the mesh's scale factor in each of the axes
- *mesh.rotation* are the Euler angles (in radians) of the rotation in each axis; if you're more comfortable with quaternions (I know I am), you can use *mesh.rotationQuaternion* instead

After adding the function to my init function and starting the scene, I notice an interesting thing—I can't see the floor I added at all! The reason is that the camera and the floor were both created at the same point in space (0,0,0). Because the ground is completely flat, only viewing from above (or below) will show it on screen. Going back to the camera's initialization in the createCamera function, I change the second variable, which is the initial position of the camera, to a new vector—the camera will now be 1.7 units (meters, in this case) above the ground:

```
var camera = new BABYLON.FreeCamera(
  "cam", new BABYLON.Vector3(0,1.7,0), scene);
```

If you start the scene now, you'll see the floor stretching over half of the screen, as shown in **Figure 9**.

Try moving around with the arrows and the mouse to get the hang of controlling the free camera.

Figure 9 **Adding a Floor to the Scene**

You'll notice the floor is completely black. I'm missing light! I'll add a new function, createLight, which I'll extend later on.

```
function createLight(scene) {
  var light = new BABYLON.DirectionalLight(
    "dir01", new BABYLON.Vector3(-0.5, -1, -0.5), scene);
  // Set a position in order to enable shadows later
  light.position = new BABYLON.Vector3(20, 40, 20);
  return light;
}
```

Don't forget to add this line to init:

```
var light = createLight(scene);
```

Babylon.js has four types of lights:

- *Hemispheric:* Ambient light, predefined with ground color (the pixels that are facing down), sky color (the pixels that are facing up) and specular color.
- *Point:* A light emitted from a single point in all directions, like the sun.
- *Spot:* Just as the name suggests, a light from a single point with a specific direction and emission radius. This light can emit shadows.
- *Directional:* Light emitted in a specific direction from everywhere. The sun might be a point light, but a directional light simulates sunlight better. This light can also emit shadows, and it's the one I used in my example.

> An instance is an exact copy of a mesh, except for the transformation in space.

Now the floor will be white. Babylon.js assigns a default white material to each mesh with no assigned material.

The lane itself will be a box, laid on the floor I just created:

```
function createLane(scene) {
  var lane = BABYLON.Mesh.CreateBox("lane", 1, scene, false);
  lane.scaling = new BABYLON.Vector3(
    laneWidth, laneHeight, totalLaneLength);
  lane.position.y = laneHeight / 2; // New position due to mesh centering
  lane.position.z = totalLaneLength / 2;
  return lane;
}
```

Once again, don't forget to add this function to init.

You can see how I used the scaling parameter—I created a box sized 1x1x1 and changed its scaling to fit the constants I predefined.

Using the same technique, I created two boxes that will act as my gutter borders on both sides of the lane (this is what the ball will fall into if it's thrown in the wrong direction). Try creating them yourself, and take a look at the accompanying project to see how I did it.

Bowling Pins and Ball

What's now missing is the bowling pins and the ball. To create the pins, I'll use a cylinder, just a single cylinder that will be multiplied many times—10 to be exact. In Babylon.js, such multiplications are called *instances*. An instance is an exact copy of a mesh, except for its transformation in space. This means that an instance's geometry and texture can't be changed, but the position, scaling and rotation can. I personally never use the original object in the scene; if I want

Visual Studio
MAGAZINE

READER'S CHOICE AWARD WINNER 2015

LEADTOOLS®
THE WORLD LEADER IN IMAGING SDKs

Visual Studio MAGAZINE 1 READER'S 2015 CHOICE

LEADTOOLS Imaging SDKs: 25 Years of Excellence

Developers know the value of a quality toolkit, and LEADTOOLS delivers. With over 25 years of experience, LEAD's award-winning document, medical and multimedia imaging SDKs include a comprehensive collection of imaging-related technologies including viewers, conversion, OCR, Barcode, Forms Recognition, PDF, DICOM, PACS and HL7 with libraries for .NET, C++, HTML5/JS, Linux, iOS and mobile platforms.

LEADTOOLS literally puts millions of lines of code at your fingertips and cuts months, if not years off of your development life cycle. Whatever your programming needs, LEADTOOLS provides the best and most diverse imaging technology available.

```
function LoadImages(instances, xmlData) {
    Clear();
    _images = [];

    for (var instanceIndex = 0; instanceIndex < framesLength; instanceIndex++) {
        var loadImageElement = document.createElement('img');
        var frame = new ImageFrame();

        frame.FrameIndex = instanceIndex;
        if (multiFrame) {
            frame.FrameNumber = instanceIndex + 1;
            frame.Instance = instances[0];
            frame.DicomData = xmlData;
            frame.ImageInformation = (instanceInd
              _images[0].ImageInformation;
        }
        else {
            frame.FrameNumber = 1;
            frame.Instance = instances[instanceIn
            frame.DicomData = (instanceIndex == 0
            frame.ImageInformation = (instanceInd
              null;
        }

        frame.ImageElement = loadImageElement;

        loadImageElement.src = objectRetrievePro
    }
}
```

To learn more, please visit our website → **www.leadtools.com**

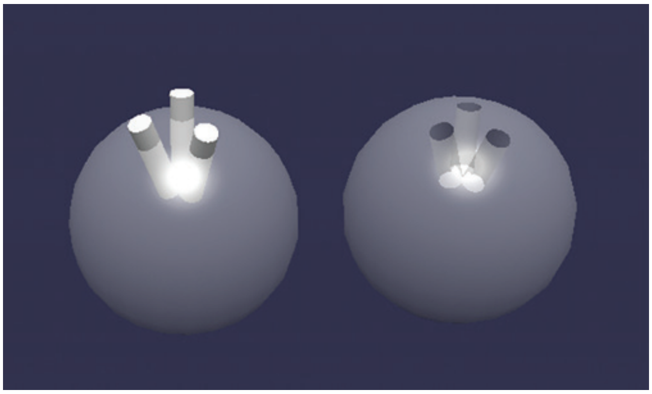


Figure 10 **Creating the Bowling Ball**

10 pins, I'll create one pin, disable it, and create 10 instances of it in 10 predefined locations:

```
function createPins(scene) {
  // Create the main pin
  var mainPin = BABYLON.Mesh.CreateCylinder(
    "pin", pinHeight, pinDiameter / 2, pinDiameter, 6, 1, scene);
  // Disable it so it won't show
  mainPin.setEnabled(false);
  return pinPositions.map(function (positionInSpace, idx) {
    var pin = new BABYLON.InstancedMesh("pin-" + idx, mainPin);
    pin.position = positionInSpace;
    return pin;
  });
}
```

The missing variables used in this function can be found in the project file, including the pinPositions, which is an array with the global positions of all 10 pins.

Now for the bowling ball. A bowling ball is a simple sphere, with 3 holes for the fingers. To create the sphere, I will use the CreateSphere function Babylon.js offers:

```
var sphere = BABYLON.Mesh.CreateSphere("sphere", 12, 0.22, scene);
```

Figure 11 **Creating a Bowling Ball Using CSG**

```
// The original sphere, from which the ball will be made
var sphere = BABYLON.Mesh.CreateSphere("sphere", 10.0, 10.0, scene);
sphere.material = new BABYLON.StandardMaterial("sphereMat", scene);
// Create pivot-parent-boxes to rotate the cylinders correctly
var box1 = BABYLON.Mesh.CreateBox("parentBox", 1, scene);
var box2 = box1.clone("parentBox");
var box3 = box1.clone("parentBox");
// Set rotation to each parent box
box2.rotate(BABYLON.Axis.X, 0.3);
box3.rotate(BABYLON.Axis.Z, 0.3);
box1.rotate(new BABYLON.Vector3(0.5, 0, 0.5), -0.2);
[box1, box2, box3].forEach(function (boxMesh) {
// Compute the world matrix so the CSG will get the rotation correctly
  boxMesh.computeWorldMatrix(true);
  // Make the boxes invisible
  boxMesh.isVisible = false;
});
// Create the 3 cylinders
var cylinder1 = BABYLON.Mesh.CreateCylinder(
  "cylinder", 4, 1, 1, 30, 1, scene, false);
cylinder1.position.y += 4;
cylinder1.parent = box1;
var cylinder2 = cylinder1.clone("cylinder", box2);
var cylinder3 = cylinder1.clone("cylinder", box3);
// Create the sphere's CSG object
var sphereCSG = BABYLON.CSG.FromMesh(sphere);
// Subtract all cylinders from the sphere's CSG object
[cylinder1, cylinder2, cylinder3].forEach(function (cylinderMesh) {
  sphereCSG.subtractInPlace(BABYLON.CSG.FromMesh(cylinderMesh));
});
// Create a new mesh from the sphere CSG
var ball = sphereCSG.toMesh("bowling ball", sphere.material, scene, false);
```

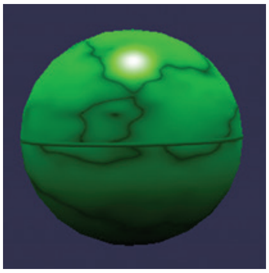


Figure 12 **Marble Texture Applied to Ball**

Now I need to drill the holes. To do that, I'll use a feature called constructive solid geometry (CSG) integrated into Babylon.js, which allows me to add or subtract meshes from existing meshes, or better yet, to add or subtract geometries from one another. What this means is that if two meshes intersect, I can "remove" one from the other and get an altered mesh. In my case, I want to create three round holes in a sphere. A cylinder will fit perfectly.

First, I create the cylinder I'll use for the first hole:

```
var cylinder = BABYLON.Mesh.CreateCylinder(
  "cylinder", 0.15, 0.02, 0.02, 8, 1, scene, false);
```

Next, I'll change the cylinder's position to intersect with the sphere:

```
cylinder.position.y += 0.15;
```

Then, I'll create CSG objects and use them to subtract the cylinder from the sphere:

```
var sphereCSG = BABYLON.CSG.FromMesh(sphere);
var cylinderCSG = BABYLON.CSG.FromMesh(cylinder);
sphereCSG.subtractInPlace(cylinderCSG);
var ball = sphereCSG.toMesh("test", sphere.material, scene, false);
```

Figure 10 shows what the sphere and cylinders look like, and right next to it the bowling ball that was created using CSG.

Decals are a way of "drawing" on top of an already-textured mesh.

Figure 11 and the playground at babylonjs-playground.com/#BIG0J show the entire code used to create the ball from scratch.

Textures

All of the meshes I created have the default white material. To make the scene more appealing, I should add other materials. The standard Babylon.js material (the default shader) has a lot of definitions to play with, which I won't be discussing here. (To learn more about the default Babylon.js shader you can try out the BabylonJS Material Editor at materialeditor.raananweber.com.) I will, however, discuss how I textured the lane and the bowling ball.

To texture the bowling ball I'll use another wonderful Babylon.js feature—procedural textures. Procedural textures aren't standard textures that use 2D images. They're programmatically created textures that are generated by the GPU (rather than the CPU), which has a positive performance impact on the scene. Babylon has many types of procedural textures—wood, bricks, fire, clouds, grass and more. The one I'm going to use is the marble texture.

Adding the following lines after creating the ball's mesh will make it a green marble ball like the one in **Figure 12**:

```
var marbleMaterial = new BABYLON.StandardMaterial("ball", scene);
var marbleTexture = new BABYLON.MarbleProceduralTexture(
  "marble", 512, scene);
marbleTexture.numberOfTilesHeight = 2;
marbleTexture.numberOfTilesWidth = 2;
marbleMaterial.ambientTexture = marbleTexture;
// Set the diffuse color to the wanted ball's color (green)
marbleMaterial.diffuseColor = BABYLON.Color3.Green();
ball.material = marbleMaterial;
```

Adding a wooden texture to the lane can be done using the standard material's diffuse texture. But that's not why I wanted to talk about the lane's material. If you look at a real bowling lane, you'll notice it has several sets of dots and a set of arrows or triangles on it. To simulate that, I could create a very large texture with all of them on it, but that might impact performance (due to a very large texture) or reduce the image quality.

I could also use decals, a new feature introduced in Babylon.js 2.1. Decals are a way of "drawing" on top of an already-textured mesh. They can be used, for example, to simulate gunshots on a wall or, as in my case, add decorations to a bowling lane. Decals are meshes, and are therefore textured using standard materials. **Figure 13** shows how I add the foul line and **Figure 14** shows what the lane looks like after adding the decals, as well as how the floor and gutters look after using procedural textures (bricks and grass) as materials.

User Input—the Babylon.js Action Manager

As a fully featured game engine, Babylon.js has a simple way to interact with user input. Let's say I want to change the ball's color using the C key. Every time I press the C, I want to set a random color to the ball:

```
// If the action manager wasn't initialized, create a new one
scene.actionManager = new BABYLON.ActionManager(scene);
// Register an action to generate a new color each time I press C
scene.actionManager.registerAction(
  new BABYLON.ExecuteCodeAction({ trigger:
  BABYLON.ActionManager.OnKeyUpTrigger, parameter: "c" },
  // The function to execute every time C is pressed
  function () {
    ball.material.diffuseColor =
      new BABYLON.Color3(Math.random(), Math.random(), Math.random());
  }
));
```

The Babylon.js Action Manager is a powerful tool for controlling actions according to triggers. A trigger can be mouse movement or clicks, mesh intersections or keyboard input. There are many triggers and actions from which to choose. Take a look at the Babylon.js Tutorial site (bit.ly/1MEkNRo) to see all of them.

Figure 13 **Adding the Foul Line Decal**

```
// Set the decal's position
var foulLinePosition = new BABYLON.Vector3(0, laneHeight, foulLaneDistance);
var decalSize = new BABYLON.Vector3(1,1,1);
// Create the decal (name, the mesh to add it to, position, up vector and
the size)
var foulLine = BABYLON.Mesh.CreateDecal("foulLine", lane,
foulLinePosition, BABYLON.Vector3.Up(), decalSize);
// Set the rendering group so it will render after the lane
foulLine.renderingGroupId = 1;
// Set the material
var foulMaterial = new BABYLON.StandardMaterial("foulMat", scene);
foulMaterial.diffuseTexture =
  new BABYLON.Texture("Assets/dots2-w-line.png", scene);
foulMaterial.diffuseTexture.hasAlpha = true;
foulLine.material = foulMaterial;

// If the action manager wasn't initialized, create a new one
scene.actionManager = new BABYLON.ActionManager(scene);
// Register an action to generate a new color each time I press "c"
scene.actionManager.registerAction(new BABYLON.ExecuteCodeAction({
trigger: BABYLON.ActionManager.OnKeyUpTrigger, parameter: "c" },
  // The function to execute every time "c" is pressed"
  function () {
    ball.material.diffuseColor =
      new BABYLON.Color3(Math.random(), Math.random(), Math.random());
  }
));
```

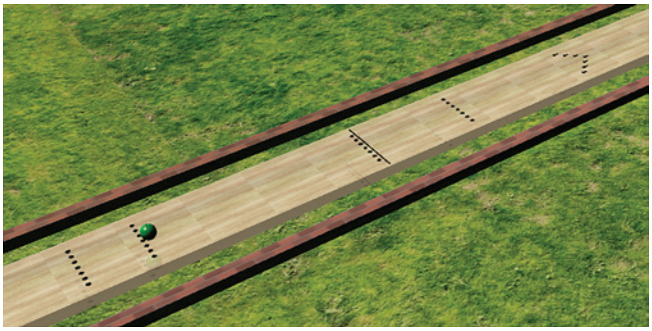


Figure 14 **Lane with Decals Added**

I'm going to use the action manager to control the ball and reset the scene. I'll add those actions in part two of the tutorial.

Using External Resources

I've created the meshes I needed using the internal functions of Babylon.js. Most of the time, this will not be enough. Babylon.js offers a lot of meshes—from spheres and boxes to complex ribbons—but it's hard to create complex models such as people, weapons for your Doom-for-the-Web, or a spaceship for your Space Invaders clone.

> The Babylon.js Action Manager is a powerful tool for controlling actions according to triggers.

Babylon.js offers exporter plug-ins to many known 3D tools, such as Blender and 3D-Studio. You can also export your models from the wonderful Clara.io and download a .babylon file.

Babylon.js uses its own file format that can contain an entire scene, including meshes, cameras, lights, animations, geometries and other information. So if you wish, you can use Blender only to model your entire scene. You can also import single meshes.

However, that's not part of this tutorial's scope. For this article, I just wanted to show how you can create a simple game using Babylon.js alone.

What's Next?

I've skipped many features integrated in Babylon.js. It has a huge number of features and I highly recommend you check the playground (babylonjs-playground.com), the Documentation page (doc.babylonjs.com), and the main Babylon.js page (babylonjs.com) to see the endless possibilities. If you have difficulties with any part of this tutorial, contact me or any of the Babylon.js heroes at the very active Babylon.js HTML5 Game Devs forum (bit.ly/1GmUyzq).

In the next article, I'll create the actual gameplay—adding physics and collision detection, the ability to throw the ball, sounds and more. ■

Raanan Weber *is an IT consultant, full stack developer, husband and father. In his spare time he contributes to Babylon.js and other open source projects. You can read his blog at blog.raananweber.com.*

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

LAS VEGAS 2016 CAMPAIGN FOR CODE

VISUAL STUDIO LIVE! is on a Campaign for Code in 2016, in support of developer education. First up on the campaign trail? Fabulous Las Vegas, where developers, software architects, engineers, designers and more will convene for five days of unbiased and cutting-edge education on the Microsoft Platform. Sharpen your skills in Visual Studio, ASP.NET, JavaScript, HTML5, Mobile, Database Analytics, and so much more.

EVENT PARTNERS

Microsoft Magenic

SUPPORTED BY

Visual Studio msdn magazine Visual Studio MAGAZINE

PRODUCED BY

1105 MEDIA YOUR GROWTH. OUR BUSINESS.

LAS VEGAS • MARCH 7-11, 2016

BALLY'S HOTEL & CASINO

Turn the page for more Event details

CONNECT WITH VISUAL STUDIO LIVE!

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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

DEVELOPMENT TOPICS INCLUDE:

- ALM/DevOps
- ASP.NET
- Cloud Computing
- Database & Analytics
- JavaScript / HTML5 Client
- Mobile Client
- Modern App Development
- UX/Design
- Visual Studio / .NET
- Windows Client

Be a responsible dev citizen. Register to code with us today!

REGISTER BY DECEMBER 18 AND SAVE $500!

Scan the QR code to register or for more event details.

USE PROMO CODE VSLDEC4

VSLIVE.COM/LASVEGAS

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

AGENDA AT-A-GLANCE

Bally's Hotel & Casino will play host to Visual Studio Live!, and is offering a special reduced room rate to conference attendees.

CONNECT WITH VISUAL STUDIO LIVE!

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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

Scan the QR code to register or for more event details.

ALM / DevOps	ASP.NET	Cloud Computing	Database and Analytics

START TIME	END TIME	Pre-Conference Workshops: Monday, March 7, 2016 *(Separate entry fee required)*	
7:30 AM	9:00 AM	Pre-Conference Workshop Registration - Coffee and Morning Pastries	
9:00 AM	1:00 PM	**M01** Workshop: Service Oriented Technologies - Designing, Developing, & Implementing WCF and the Web API - *Miguel Castro*	
1:00 PM	2:00 PM	Lunch @ Le Village Buffet, Paris Las Vegas	
2:00 PM	6:00 PM	**M01** Workshop Continues	
7:00 PM	9:00 PM	Dine-A-Round	

START TIME	END TIME	Day 1: Tuesday, March 8, 2016	
7:00 AM	8:00 AM	Registration - Coffee and Morning Pastries	
8:00 AM	9:00 AM	Keynote: To Be Announced	
9:15 AM	10:30 AM	**T01** This session is sequestered, details will be released soon ✪	**T02** Angular 101 - *Deborah Kurata*
10:45 AM	12:00 PM	**T06** This session is sequestered, details will be released soon ✪	**T07** Angular 2.0: A Comparison *- Deborah Kurata*
12:00 PM	1:00 PM	Lunch	
1:00 PM	1:30 PM	Dessert Break - Visit Exhibitors	
1:30 PM	2:45 PM	**T11** Native Mobile App Development for iOS, Android and Windows Using C# *- Marcel de Vries*	**T12** ASP.NET 5 in All its Glory *- Adam Tuliper*
3:00 PM	4:15 PM	**T16** Getting Started with Hybrid Mobile Apps with Visual Studio Tools for Cordova *- Brian Noyes*	**T17** MVC 6 - The New Unified Web Programming Model *- Marcel de Vries*
4:15 PM	5:30 PM	Welcome Reception	

START TIME	END TIME	Day 2: Wednesday, March 9, 2016	
7:30 AM	8:00 AM	Registration - Coffee and Morning Pastries	
8:00 AM	9:15 AM	**W01** Busy .NET Developer's Guide to Native iOS - *Ted Neward*	**W02** Getting Started with Aurelia *- Brian Noyes*
9:30 AM	10:45 AM	**W06** Busy Developer's Guide to Mobile HTML/JS Apps - *Ted Neward*	**W07** This session is sequestered, details will be released soon ✪
11:00 AM	12:00 AM	General Session: To Be Announced	
12:00 PM	1:00 PM	Birds-of-a-Feather Lunch	
1:00 PM	1:30 PM	Dessert Break - Visit Exhibitors - Exhibitor Raffle @ 1:15pm (Must be present to win)	
1:30 PM	2:45 PM	**W11** Optimizing Applications for Performance Using the Xamarin Platform *- Kevin Ford*	**W12** An Introduction to TypeScript *- Jason Bock*
3:00 PM	4:15 PM	**W16** Leverage Azure App Services Mobile Apps to Accelerate Your Mobile App Development - *Brian Noyes*	**W17** Take a Gulp, Make a Grunt, and Call me Bower - *Adam Tuliper*
4:30 PM	5:45 PM	**W21** This session is sequestered, details will be released soon ✪	**W22** JavaScript for the C# (and Java) Developer - *Philip Japikse*
7:00 PM	9:00 PM	Evening Out Event	

START TIME	END TIME	Day 3: Thursday, March 10, 2016	
7:30 AM	8:00 AM	Registration - Coffee and Morning Pastries	
8:00 AM	9:15 AM	**TH01** Building for the Internet of Things: Hardware, Sensors & the Cloud - *Nick Landry*	**TH02** Responsive Web Design with ASP.NET 5 - *Robert Boedigheimer*
9:30 AM	10:45 AM	**TH06** User Experience Case Studies - Good and Bad - *Billy Hollis*	**TH07** Tools for Modern Web Development *- Ben Hoelting*
11:00 AM	12:15 PM	**TH11** This session is sequestered, details will be released soon ✪	**TH12** SASS and CSS for Developers *- Robert Boedigheimer*
12:15 PM	1:45 PM	Lunch @ Le Village Buffet, Paris Las Vegas	
1:45 PM	3:00 PM	**TH16** Exposing an Extensibility API for your Applications - *Miguel Castro*	**TH17** Hack proofing your Web Applications - *Adam Tuliper*
3:15 PM	4:30 PM	**TH21** UWP Development for WPF and Silverlight Veterans - *Walt Ritscher*	**TH22** Increase Website Performance and Search with Lucene.Net Indexing *- Ben Hoelting*

START TIME	END TIME	Post-Conference Workshops: Friday, March 11, 2016 *(Separate entry fee required)*	
7:30 AM	8:00 AM	Post-Conference Workshop Registration - Coffee and Morning Pastries	
8:00 AM	12:00 PM	**F01** Workshop: Upgrading Your Skills to ASP.NET 5 - *Mark Michaelis*	
12:00 PM	1:00 PM	Lunch	
1:00 PM	5:00 PM	**F01** Workshop Continues	

Speakers and sessions subject to change

✪ DETAILS COMING SOON! These sessions have been sequestered by our conference chairs. Be sure to check vslive.com/lasvegas for session updates!

LAS VEGAS 2016 CAMPAIGN FOR CODE

Modern Apps LIVE!

MOBILE, CROSS-DEVICE & CLOUD DEVELOPMENT

Presented in partnership with **Magenic**

BONUS CONTENT! Modern Apps Live! is now a part of Visual Studio Live! Las Vegas at no additional cost!

JavaScript / HTML5 Client	Mobile Client	UX/Design	Visual Studio / .NET	Windows Client	Modern Apps Live!

Pre-Conference Workshops: Monday, March 7, 2016 *(Separate entry fee required)*		
Pre-Conference Workshop Registration - Coffee and Morning Pastries		
M02 Workshop: Agile DevOps - *Brian Randell*	**M03** Workshop: SQL Server for Developers - Leonard Lobel	**M04** Workshop: Sequestered, details will be released soon ★
Lunch @ Le Village Buffet, Paris Las Vegas		
M02 Workshop Continues	**M03** Workshop Continues	**M04** Workshop Continues
Dine-A-Round		
Day 1: Tuesday, March 8, 2016		
Registration - Coffee and Morning Pastries		
Keynote: To Be Announced		
T03 Introduction to Next Generation of Azure Compute - Service Fabric and Containers - *Vishwas Lele*	**T04** Developer Productivity in Visual Studio 2015 - *Robert Green*	**T05** Defining Modern App Development - *Rockford Lhotka*
T08 Docker and Azure - *Steve Lasker*	**T09** Building Windows 10 Line of Business Applications - *Robert Green*	**T10** Modern App Architecture - *Rockford Lhotka*
Lunch		
Dessert Break - Visit Exhibitors		
T13 This session is sequestered, details will be released soon ★	**T14** This session is sequestered, details will be released soon ★	**T15** ALM with Visual Studio Online and Git - *Brian Randell*
T18 This session is sequestered, details will be released soon ★	**T19** Geospatial Data Types in SQL Server - *Leonard Lobel*	**T20** DevOps and Modern Applications - *Dan Nordquist*
Welcome Reception		
Day 2: Wednesday, March 9, 2016		
Registration - Coffee and Morning Pastries		
W03 Exploring Microservices in a Microsoft Landscape - *Marcel de Vries*	**W04** This session is sequestered, details will be released soon ★	**W05** Reusing Logic Across Platforms - *Kevin Ford*
W08 Breaking Down Walls with Modern Identity - *Eric D. Boyd*	**W09** JSON and SQL Server, Finally Together - *Steve Hughes*	**W10** Coding for Quality and Maintainability - *Jason Bock*
General Session: To Be Announced		
Birds-of-a-Feather Lunch		
Dessert Break - Visit Exhibitors - Exhibitor Raffle @ 1:15pm (Must be present to win)		
W13 Real-world Azure DevOps - *Brian Randell*	**W14** Using Hive and Hive ODBC with HDInsight and Power BI - *Steve Hughes*	**W15** Start Thinking Like a Designer - *Anthony Handley*
W18 This session is sequestered, details will be released soon ★	**W19** Introduction to Spark for C# Developers - *James McCaffrey*	**W20** Applied UX: iOS, Android, Windows - *Anthony Handley*
W23 Managing Windows Azure with PowerShell - *Mark Michaelis*	**W24** Introduction to R for C# Programmers - *James McCaffrey*	**W25** Leveraging Azure Services - *Kevin Ford*
Evening Out Event		
Day 3: Thursday, March 10, 2016		
Registration - Coffee and Morning Pastries		
TH03 Unit Testing & Test-Driven Development (TDD) for Mere Mortals - *Benjamin Day*	**TH04** Effective Agile Software Requirements - *Richard Hundhausen*	**TH05** Building for the Modern Web with JavaScript Applications - *Allen Conway*
TH08 Unit Testing JavaScript - *Ben Dewey*	**TH09** Lessons Learned: Being Agile in a Waterfall World - *Philip Japikse*	**TH10** Building a Modern Android App with Xamarin - *Nick Landry*
TH13 This session is sequestered, details will be released soon ★	**TH14** Real World Scrum with Team Foundation Server 2015 & Visual Studio Online - *Benjamin Day*	**TH15** Building a Modern Windows 10 Universal App - *Brent Edwards*
Lunch @ Le Village Buffet, Paris Las Vegas		
TH18 Async Patterns for .NET Development - *Ben Dewey*	**TH19** DevOps vs. ALM Different Measures of Success - *Mike Douglas*	**TH20** This session is sequestered, details will be released soon ★
TH23 Improving Quality for Agile Projects Through Manual and Automated UI Testing - NO CODING REQUIRED! - *Mike Douglas*	**TH24** This session is sequestered, details will be released soon ★	**TH25** Analyzing Results with Power BI - *Steve Hughes*
Post-Conference Workshops: Friday, March 11, 2016 *(Separate entry fee required)*		
Post-Conference Workshop Registration - Coffee and Morning Pastries		
F02 Workshop: Building Business Apps on the Universal Windows Platform - *Billy Hollis*		**F03** Workshop: Modern Development Deep Dive - *Kevin Ford, Nick Landry, Brent Edwards, Allen Conway*
Lunch		
F02 Workshop Continues		**F03** Workshop Continues

VSLIVE.COM/LASVEGAS

Build a Cross-Platform UX with Xamarin.Forms

Keith Pijanowski

If you've decided to experiment with Xamarin then you've decided to start an exciting journey. Unlike other development tools, which tie you to a single platform, Xamarin gives you access to four different platforms. With Xamarin, you can use your C# skills to write your application for iOS, Android, Windows Phone and the Mac OS X. It's important to note that Xamarin delivers a fully native experience. You get native performance, complete API access on all platforms and a native UI. This article will focus on the three mobile platforms supported by Xamarin: iOS, Android and Windows Phone.

The benefits of using Xamarin are exciting for many reasons. One reason is that using a platform like Xamarin is just plain fun. Xamarin is a playground to learn all these platforms in more detail while at the same time using your C# skills. Instead of having to work in multiple workspaces (Xcode, Android Studio/Eclipse and Visual Studio) with multiple languages (C#, Java and Objective-C/Swift), you can work in one familiar environment (Visual Studio) with one language (C#).

This article discusses:

- Xamarin presents two main approaches to building native apps that share code: Traditional Xamarin and Xamarin.Forms
- Use the Traditional Xamarin approach when your app needs specialized interactions on each platform, and makes use of many platform-specific APIs
- Xamarin.Forms is a complete framework for building native apps for three platforms from a single, shared code base

Technologies discussed:

Xamarin.Forms, C#, Visual Studio, iOS, Android, Windows Phone

Code download available at:

msdn.com/magazine/msdnmag1215

With Xamarin, you can create a single solution that can render a version of your application on iOS, Android, Windows Phone and OS X.

There are also a few financial advantages to developing applications with Xamarin. With Xamarin, you can create a single solution that can render a version of your application on iOS, Android, Windows Phone and OS X. Code reuse and skill set reuse between these platforms represents significant cost savings and productivity gains. When it comes to monetizing your application, basic economics dictates that increasing the number of potential customers creates more opportunity for acquiring more paying customers. Collectively, the three mobile platforms supported by Xamarin represent more than 98 percent of all mobile devices across the globe.

Figure 1 **XAML for a Content Page**

```
<?xml version="1.0" encoding="UTF-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/XAML"
  x:Class="XamarinFormsTestHarness.FeedbackPage"
  Title="Feedback">

  <StackLayout Padding="10,10,10,10">

<Label Text="Name:" FontAttributes="None" FontSize="Medium" TextColor="#3498DB"/>
  <Entry Text="{Binding Name}" Placeholder="First and Last" Keyboard="Default"/>
  <Label Text="Email:" FontSize="Medium" TextColor="#3498DB" />
  <Entry Text="{Binding Email}" Placeholder="name@company.com" Keyboard="Email" />
  <Label Text="Feedback:" TextColor="#3498DB" FontSize="Medium"/>
  <Editor Text="{Binding Text}" HeightRequest="200" BackgroundColor="Gray"/>
    <Button x:Name="ButtonSubmitFeedback" Text="Submit"
      BackgroundColor="#3498DB"
      TextColor="White"
      Command="{Binding SaveFeedbackCommand}"/>

  </StackLayout>
</ContentPage>
```

Finally, if you're an enterprise building mobile line-of-business (LOB) applications, then having the ability to create a solution for all relevant platforms means that employees may use their own devices for work.

The Traditional Xamarin Approach

A Traditional Xamarin solution that supports all three mobile platforms is comprised of four projects at a minimum—a Portable Class Library (PCL) and a platform-specific project for each platform. The PCL (or a shared code project) contains the models, data access code and business logic. The code in the PCL can be referenced and reused from the other projects. However, PCLs contain no UI code. UI code is maintained in the platform-specific projects. It's also important to note that the goal of Traditional Xamarin is to provide feature parity with the native programming environments and the platforms they represent. Everything you can do in Objective-C, Swift or Java can be done in C# with Xamarin and Visual Studio. Any API you want to access on iOS, Android and Windows Phone you can; Xamarin has 100 percent native API access. This code is also in the platform-specific projects.

While the Traditional Xamarin approach will let you share significant amounts of code, there is a way to share even more code.

Enter Xamarin.Forms

Xamarin.Forms pushes the envelope further still when it comes to reusability. Specifically, Xamarin.Forms provides all the benefits of the traditional approach while also letting UI logic be reused across platforms. Xamarin.Forms solutions are still structured the same way as Traditional Xamarin, however, the PCL can now contain UI code. The platform-specific projects are still needed for project settings and to house images and other resources that are different across platforms.

It's no surprise the folks at Xamarin would attempt to do this. Even though iOS, Android and Windows Phone were conceived by different architects and grew up under different roofs, there is still a lot of commonality within their UIs. Users view content one page at a time. Also, many controls are similar across platforms. For example, text boxes, push buttons, radio buttons, labels, list views and image controls are relatively the same across platforms. In total, Xamarin.Forms comes with 40 controls, seven layouts and five page types for building native UXes.

The programming model adopted by Xamarin.Forms is XAML/C#. Pages, layouts and controls can be specified using XAML in much the same manner as pages created in a Windows Phone project. Pages, layouts and controls may also be created entirely in code. Both techniques will be shown in this article.

Xamarin.Forms pushes the envelope further still when it comes to reusability.

When using Xamarin.Forms what can be tricky is designing and implementing the UI that organizes your application's features and content. This has to be done in a way that maximizes code reuse while at the same time producing an application that is natural to use on each platform. To this end, Xamarin provides five page types to address common UI scenarios: Content pages for displaying basic content; Navigation pages for providing navigational features; Tabbed pages for creating pages that have tabs across the top or bottom of the screen; Master-Detail pages for presenting high-level data and detail data across two panes of information; and Carousel pages for creating pages that scroll content horizontally. This article will demonstrate the Xamarin.Forms solution to each of these scenarios by discussing each of the five page types. Page layouts and UI controls that come with Xamarin.Forms will also be shown along the way.

Displaying Content

Displaying basic content on a screen is done with the Content page. **Figure 1** shows the XAML for a page that's used to capture user feedback.

Figure 2 **Code for the Master Detail Page**

```
using System;
using Xamarin.Forms;

namespace XamarinFormsTestHarness
{

  public class MasterDetailControlPage : MasterDetailPage
  {

    public MasterDetailControlPage()
      {
      MenuPage menuPage = new MenuPage();

      menuPage.MenuListView.ItemSelected += (sender, e) =>
        MenuSelected(e.SelectedItem as MenuItem);

      // The Detail page must be set or the page will crash when rendered.
      this.Master = menuPage;
      this.Detail = new NavigationPage(new FeedbackPage());
      this.IsPresented = true;
      }
        ...
  }
}
```

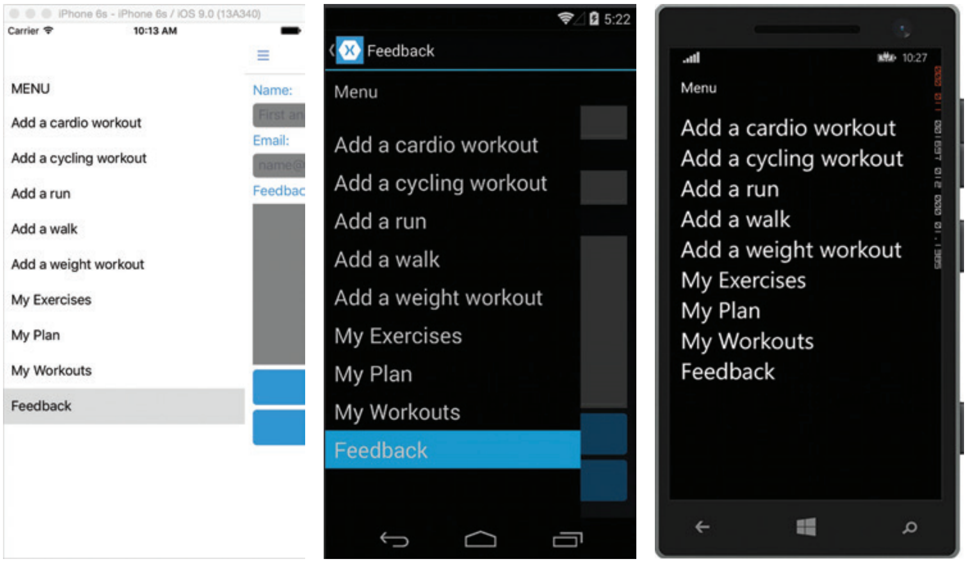


Figure 3 **Master Detail Page Rendered on iOS (Left), Android (Center) and Windows Phone (Right)**

The XAML in **Figure 1** uses a stack layout to organize basic controls such as Label, Entry, Editor and Button. As an example of Xamarin.Forms native capabilities, on iOS an Entry is rendered as a UITextField, on Android as an EditText and on Windows Phone as a TextBox.

A note of caution: Xamarin XAML is different from Windows Phone XAML. You cannot take the XAML from a Windows Phone page and drop it into a Xamarin Content page. For example, if you are used to using the Margin and Padding properties to make subtle tweaks to the positioning of your controls, then you will be unhappy to learn that these properties are not available in many of the Xamarin.Forms controls. Additionally, many of the controls and their properties have different names as compared to their Windows Phone counterparts. These discrepancies might seem like the symptom of a new product. In many cases this might be true, but also remember that Xamarin is not just for Windows Phone developers. The goal of Xamarin is to provide an environment for mobile developers of all backgrounds. If a certain control isn't named the same as it is in Window Phone, then chances are that the name Xamarin is using is a name that comes from either Android or iOS.

The Content page is the building block for all the other page types. Carousel pages, Tabbed pages, Navigation pages and Master Detail pages use content pages to build their UXes.

Master Detail Scenarios

Master Detail Scenarios are very common in mobile applications. Therefore, Xamarin built a specific page type to handle this scenario. A Master Detail page can be used to organize application functionality or application data. For example, in an application that manages customers and orders, a list of customers can be shown in a Master page. Once the user touches a specific customer, then all the orders for that customer can be shown in a details page. A Master Detail page can also be used to show the menu options of an application. In this scenario, the Master page displays a list of features available within the application and each detail page provides the appropriate feature. **Figure 2** shows the constructor code for a Master Detail page used as the main menu of a fitness application. **Figure 3** shows how the Master page is rendered on each platform. A Master page is rendered on iOS and Android as a popover menu that slides in from the left side of the screen. On Windows Phone it's rendered as a complete page. When users touch an option, they're taken to the appropriate page.

There are quite a few facts worth noting about the code in **Figure 2** to make it understandable. First, the Master Detail page is really a controller. It does not contain the UX of the Master page or any of the Detail pages. These pages are in separate files. Its job is to show the correct page based on the user's request. In **Figure 3** the Master page is a content page named MenuPage. MenuPage is instantiated and set into the Master property of the Master Detail page. The XAML for MenuPage isn't shown here for brevity. It can be found in the code download that accompanies this article. The Master Detail page shown in **Figure 2** also does not contain the UX for any of the Detail pages. They are also in separate files. The Detail pages are instantiated and set into the Detail property of the Master Detail page when the user selects a menu option.

The goal of Xamarin is to provide an environment for mobile developers of all backgrounds.

In the constructor of the Master Detail page both the Master page and the Detail page are set. The Master Detail page will bomb out when initially shown if both properties are not set.

It's also important to understand the IsPresented property. This property indicates whether the Master page is shown to the user. If it is set to True then the Master page is shown. If it is set to False then the current Detail page is shown.

Windows Phone treats the entire Master page experience as a single page view. In other words, when the user navigates to a detail view Windows Phone doesn't treat it as standard-page navigation. When the user taps the back button on a detail view, the user will be taken to the page shown prior to the initial navigation to the Master Detail page. This is a poor UX as the user was most likely expecting to be taken back to the Master view of the Master Detail page. To get around this problem Xamarin.Forms lets the OnBackButtonPressed event to be overridden. This event is invoked

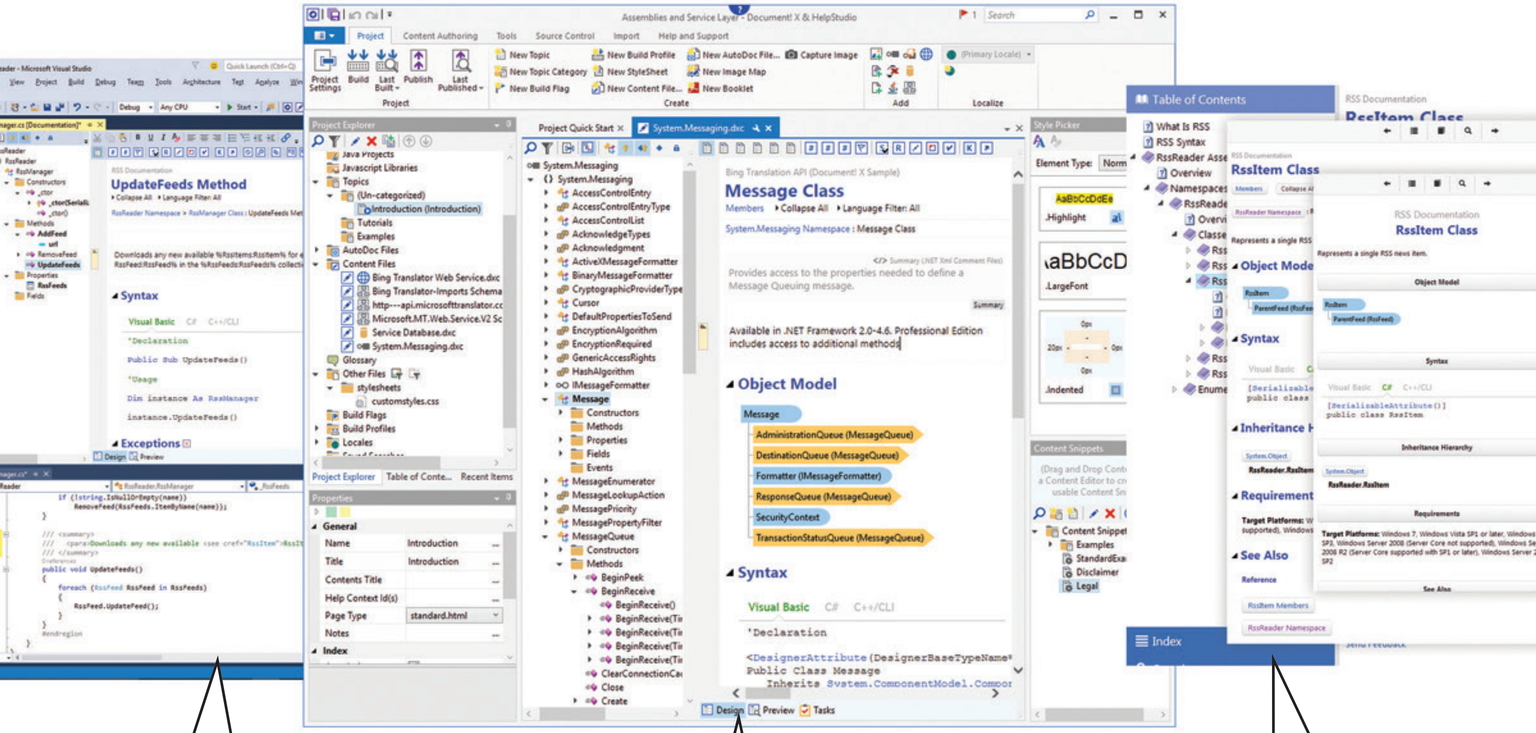


Figure 4 **Creating a Tabbed Page with the Children Property**

```
using Xamarin.Forms;

namespace XamarinFormsTestHarness
{
  class TabbedControlPage : TabbedPage
    {
    public TabbedControlPage()
      {
      this.Title = "My Workouts";
      this.Children.Add(new ThisWeek());
      this.Children.Add(new LastWeek());
      this.Children.Add(new ThisMonth());
      this.Children.Add(new LastMonth());
      this.Children.Add(new All());
      }
    }
}
```

on both Windows Phone and Android because all Windows Phone and Android devices have a back button. This event is not invoked if your app is running on iOS as iOS does not have a back button. On iOS the user can navigate back to the master page by tapping the master page's icon, which will be displayed in the upper left of all detail pages. It's important to always set the Master page's icon property, otherwise the user will be trapped on one of the application's detail pages with no way to get back to the Master page.

Application Navigation

The Navigation page manages page navigation for iOS and Android. NavigationPages are push/pop navigation. Push a new page to the top of the stack and then pop it off. The Windows Phone OS keeps track of all navigation and all Windows Phone devices have a hardware back key. Therefore, the user can always go back to the previous screen. So, on Windows Phone the Navigation page has no effect.

For iOS and Android the Navigation page provides a UX that shows the title of the current page and the title of the previous page as a link at the top of the page. The root page will only contain the title of the page. If users touch the title of the previous page they're taken back to the previous page.

The easiest way to use the Navigation page is to instantiate it in code and pass a page into its constructor. The following line of code is used to create the detail pages in **Figure 2**:

```
this.Detail = new NavigationPage(displayPage);
```

Navigation pages are especially important on iOS because the devices on which it runs don't have a hardware back button. Without the features provided by the Navigation page, users would not have the ability to go back one page on iOS.

Tabs

The UI of a tabbed page is rendered as a list of tabs that appear either at the top of the screen or at the bottom. On iOS, the list of tabs appears at the bottom of the screen, and the detail area is above. If there are more tabs than can fit on the screen, then the iOS rendering will provide a "More" tab that provides access to the options that could not fit on the screen. On Android the tabs appear across the top of the screen with the detail area below. The user can horizontally scroll the collection of tabs if the collection is too large to fit on the screen. On Windows Phone the tabbed page is rendered as a Pivot page.

The Content page is the building block for all the other page types.

There are two ways to create a tabbed page. First, application developers can set Content Page objects into the tabbed page's Children property, as shown in **Figure 4**. One tab will be created for each entry into the Children collection. The Title and Icon properties of each page are used to create the tabs. This technique is useful when each tab will have a different look and feel and the data in each tab is different. **Figure 5** shows how the tabbed page from **Figure 4** is rendered on each platform.

Another way to create a tabbed page is to assign a list of objects instantiated from the same class to the tabbed page's ItemsSource property. One tab will be created for each object in this list. A DataTemplate must then be set into the tabbed page's ItemTemplate property. The DataTemplate must be created from a Content page that uses data binding to get data from the corresponding object in the ItemSource property. It's important to bind this Content page's title property to a property in the bound object. The Title property will be used to create the tab label. The remainder of this content page may be used to bind to other properties from the bound object. This technique is best for tabbed pages where each tab will have the same UX showing different data.

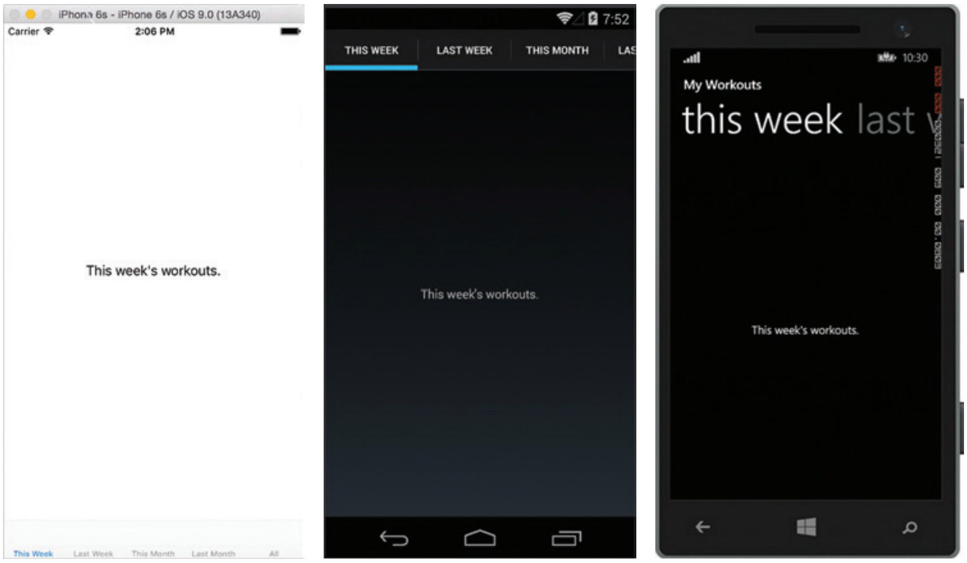


Figure 5 **The Tabbed Page Rendered on iOS (Left), Android (Center) and Windows Phone (Right)**

Visual Studio
MAGAZINE

READER'S CHOICE AWARD WINNER 2015

JetBRAINS

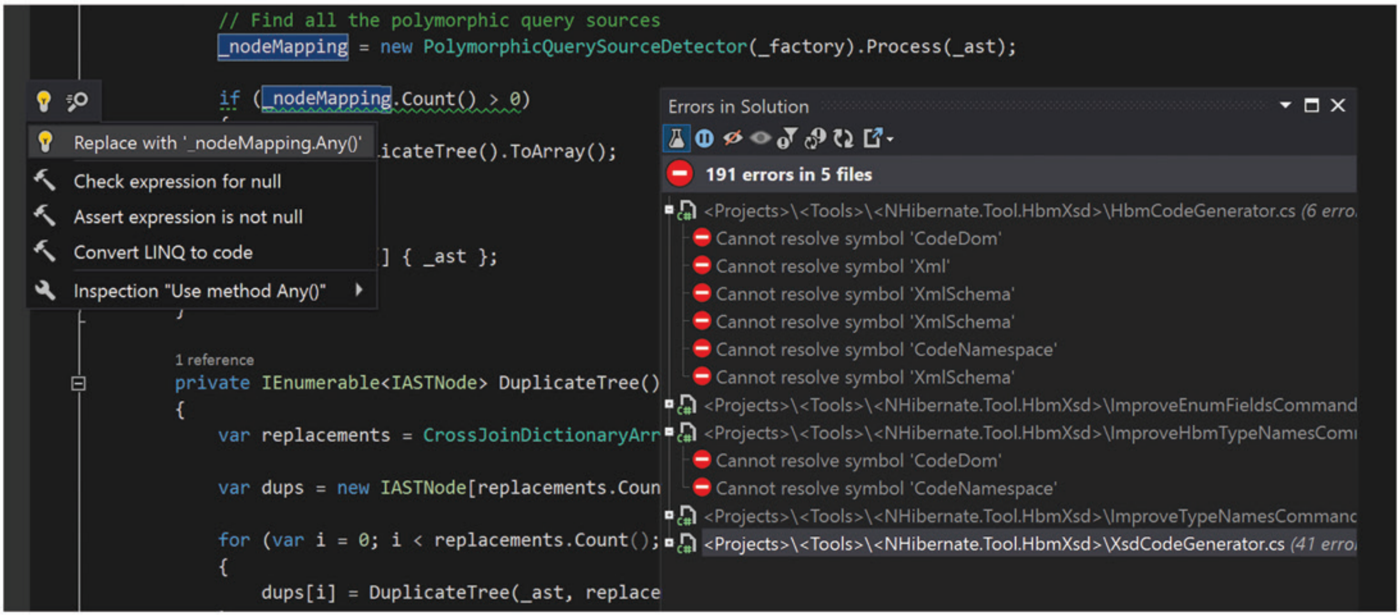
R# ReSharper

Visual Studio MAGAZINE
1
READER'S
2015
CHOICE

ReSharper: Renowned Visual Studio Productivity Tool

With unparalleled support for C#, VB.NET, XAML, JavaScript, TypeScript, XML, HTML, CSS, ASP.NET, ASP.NET MVC, NAnt and MSBuild scripts including comprehensive cross-language functionality, ReSharper will help any Visual Studio user write better code, easily examine and refactor existing code bases.

You can spend less time on routine, repetitive manual work and instead focus on the task at hand. A robust set of features for automatic error-checking and code correction cuts development time and increases your efficiency. You'll find that ReSharper quickly pays back its cost in increased developer productivity and improved code quality.


Replace with '_nodeMapping.Any()'
Check expression for null
Assert expression is not null
Convert LINQ to code
Inspection "Use method Any()"
Errors in Solution

```
        // Find all the polymorphic query sources
        _nodeMapping = new PolymorphicQuerySourceDetector(_factory).Process(_ast);

        if (_nodeMapping.Count() > 0)
                                 icateTree().ToArray();

                                 ] { _ast };

    1 reference
    private IEnumerable<IASTNode> DuplicateTree()
    {
        var replacements = CrossJoinDictionaryArr

        var dups = new IASTNode[replacements.Coun

        for (var i = 0; i < replacements.Count();
        {
            dups[i] = DuplicateTree(_ast, replace
```

```
191 errors in 5 files
<Projects>\<Tools>\<NHibernate.Tool.HbmXsd>\HbmCodeGenerator.cs (6 erro
    Cannot resolve symbol 'CodeDom'
    Cannot resolve symbol 'Xml'
    Cannot resolve symbol 'XmlSchema'
    Cannot resolve symbol 'XmlSchema'
    Cannot resolve symbol 'CodeNamespace'
    Cannot resolve symbol 'XmlSchema'
<Projects>\<Tools>\<NHibernate.Tool.HbmXsd>\ImproveEnumFieldsCommand
<Projects>\<Tools>\<NHibernate.Tool.HbmXsd>\ImproveHbmTypeNamesComi
    Cannot resolve symbol 'CodeDom'
    Cannot resolve symbol 'CodeNamespace'
<Projects>\<Tools>\<NHibernate.Tool.HbmXsd>\ImproveTypeNamesCommanc
<Projects>\<Tools>\<NHibernate.Tool.HbmXsd>\XsdCodeGenerator.cs (41 erro
```

To learn more, please visit our website → www.jetbrains.com/resharper

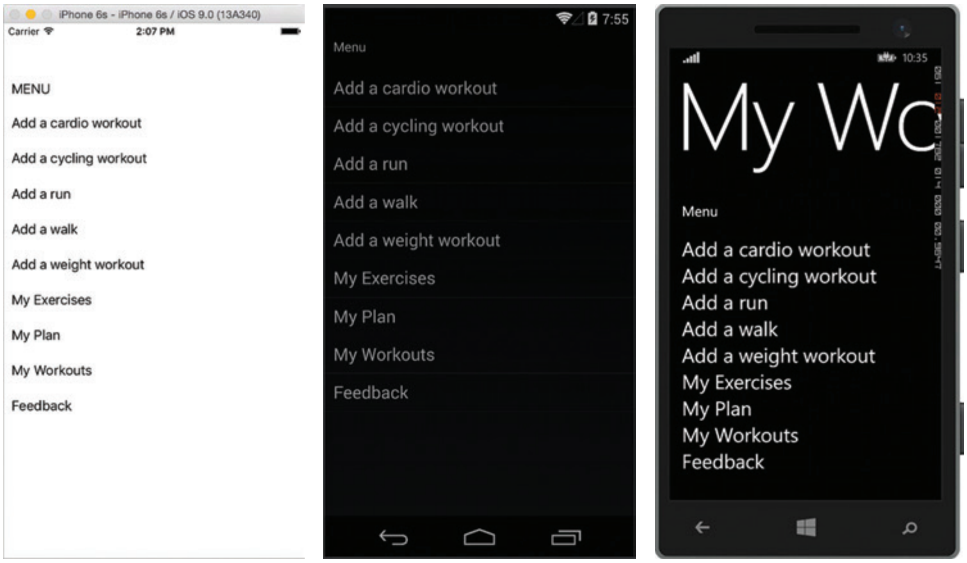


Figure 6 **Carousel Page Rendered on iOS (Left), Android (Center) and Windows Phone (Right)**

Carousels

The UI of a Carousel page lets the user swipe the screen from side to side in order to show different pages of content. Windows Phone developers will recognize the Carousel page as a Panorama. The Carousel page contains a Children property just like the tabbed page. To create a Carousel page, set Content page objects into the Children property of the Carousel page. Each Content page will be rendered as a screen of content. The Title property of each Content page isn't used in Carousel pages. (This is different from tabbed pages where each Title is used as the tab title.) Therefore, if you need a label for each screen, then you'll need to implement this manually as part of the Content page content. **Figure 6** shows how this page is rendered on each platform. Notice that on iOS and Android there's no visible clue informing the user of additional content to the left or right of the current view. Windows Phone does this by showing a portion of the page title. For this reason, be careful using a Carousel on iOS and Android. Users might miss important content in your application.

A well-designed Panorama page is like a magazine cover. It gives the user a quick glimpse at what's inside.

Designing for Three Platforms

If you're familiar with all three platforms then at this point you've probably realized that each of the Xamarin.Forms page types represents a UI scenario that's needed in one of the platforms and it was then creatively implemented in the other two. For example, Navigation pages that provide a link to the previous page are needed to provide back-key functionality on iOS because iOS doesn't have a back key. Navigation pages are also rendered on Android in a similar fashion because having a link to the previously visited page at the top of the current page is a popular UI structure on Android even though Android has a back key.

Similarly, Tabs are used quite often on both iOS and Android. Therefore, Xamarin.Forms needed to have this UI structure in the toolbox. On Windows Phone tabs are rendered as Pivot pages, which are an essential part of the OS.

Master Detail pages provide a popover panel, which is sometimes used on iOS and Android.

Finally, Windows Phone has the concept of a Panorama page. Panorama pages are best used as the homepage of an application. The idea of a Panorama page is to provide a high-level view of the application. A well-designed Panorama page is like a magazine cover. It gives the user a quick glimpse at what's inside. If done in a compelling manner, the Panorama pulls the user further into the application. The idea of horizontally swiping between screens of content isn't a familiar scenario in iOS and Android. Therefore, care must be taken when using a Carousel page in a Xamarin.Forms application.

As the Xamarin.Forms platform evolves, more page types will most likely be added. A Xamarin developer will have a design responsibility to look at each page type and determine if it makes sense for all the platforms his app is targeting.

Wrapping Up

Xamarin presents two main approaches to building native applications that share code: Traditional Xamarin and Xamarin.Forms. Picking the right approach depends on what you're building.

Use the Traditional Xamarin approach when your application needs specialized interactions on each platform and makes use of many platform-specific APIs. Traditional Xamarin also provides the power to build UIs that are customized for each platform.

Xamarin.Forms is a complete framework for building native applications using a single, shared code base. Consider Xamarin.Forms if you're prototyping, building a data entry application or your application requires little platform-specific functionality. Xamarin.Forms is also good for projects where sharing code is more important than a custom UI for each platform. ■

Keith Pijanowski *is an engineer, entrepreneur and businessman. He has more than 20 years of experience in the software industry and has worked for startups and large companies in roles that range from writing code to business development. Reach him at keithpij@msn.com or on Twitter: @keithpij.*

Thanks to the following technical expert for reviewing this article: *Pierce Boggan (Xamarin)*

Work With Relational Data, Not Complex APIs or Services

Whether you are a developer using ADO.NET, JDBC, OData, or MySQL, or a systems integrator working with SQL Server or Biztalk, or even an information worker familiar with ODBC or Excel – our products give you bi-directional access to live data through easy-to-use technologies that you are already familiar with. If you can connect to a database, then you will already know how to connect to Salesforce, SAP, SharePoint, Dynamics CRM, Google Apps, QuickBooks, and much more!

Give RSSBus a try today and see what mean:

visit us online at www.rssbus.com to learn more or download a free trial.

rssbus
INTEGRATION YOUR WAY

How Azure, Web API and Redis Helped Deliver Data Faster

Johnny Webb and Sean Iannuzzi

In today's world of real-time marketing, the spinning circle of death that too often accompanies waiting for a page to load or a search query to populate can quite literally mean the death of a sale. As more and more data records are created, stored and analyzed, the generation of rich, real-time data insights grows ever more challenging. The question becomes how to choose the right technology tools and architectures to best sift through millions of data records and deliver instantaneous results for a better customer experience.

This article will explore a recent use case in which we helped a client implement Redis and Web API. We'll discuss our implementation approach, the challenges we encountered and how we ultimately achieved the performance requirements.

Business Challenge

Several months ago, a Fortune 100 diversified insurance and financial services organization approached Harte Hanks, a global marketing company, with a new initiative. The company wanted to provide a better, richer customer experience during the online insurance quote process. To do this, it needed to enable fast and proactive access to the customer's policy data.

This article discusses:

- The business challenge posed by the client
- Requirements associated with the initiative
- Implementing the data layer
- Data loading and benchmarking
- Implementing the service layer

Technologies discussed:

Microsoft Azure, Web API, Redis

The objective was to allow customers to view and select one or more insurance policies through Internet self-service or agent-facing applications. Ultimately, this would lead to higher business value, greater customer satisfaction, higher quote-to-bind rates, and an improved agent commissioning process. In order to make this a reality, during the quote process the customers' stored details and data needed to be processed, matched and provided proactively, in a near instantaneous manner.

The client's database contained more than 50 million records, which meant with any traditional database a request of this nature typically would take multiple seconds to load. We were tasked with providing a Web service capable of executing queries on this large data source and delivering results to consumers within milliseconds.

For this task, we evaluated the implementation of Redis and Web API. The overall measure factors were:

- Transaction Requests
- Object Requests
- Responses Per Second
- Total Bandwidth of Data
- Total Throughput of the Site

Implementation: How We Did It

Building a Web service to expose data requires multiple layers of components providing specific functionality. As the demand for the Web service grows, the infrastructure supporting these layers must adapt and scale quickly. And though scalability is essential, availability is equally important and must always be considered in order to keep the consumer happy. Furthermore, exposed endpoints must be monitored and wrapped with specific authentication protocols to ensure the data is protected and delivered securely with the maximum performance. To help accommodate these demands

Visual Studio MAGAZINE

READER'S CHOICE AWARD WINNER 2015

SPARX SYSTEMS

Visual Studio MAGAZINE 3 READER'S 2015 CHOICE

Ultimate Design Platform

Expand your Vision – Build Tomorrow

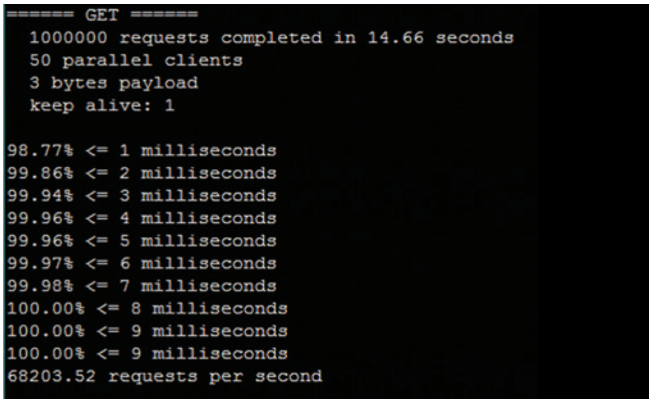
Enterprise Architect from Sparx Systems provides a rich set of visual modeling tools including software design & visualization capabilities that empowers developers and teams alike. A platform that tightly integrates development into the full project lifecycle from requirements through design, build, test and delivery. Leverage the visual power of open standards such as UML, SysML, BPMN, Archimate, TOGAF, UPDM, XSD. Wireframe your designs. Connect model and code and build visual models from your executing software with advanced recording and analysis capabilities.

An affordable, highly scalable, integrated platform for you and your team.

ENTERPRISE ARCHITECT

* Build Tomorrow

To learn more, please visit our website www.sparxsystems.com



```
====== GET ======
  1000000 requests completed in 14.66 seconds
  50 parallel clients
  3 bytes payload
  keep alive: 1

98.77% <= 1 milliseconds
99.86% <= 2 milliseconds
99.94% <= 3 milliseconds
99.96% <= 4 milliseconds
99.96% <= 5 milliseconds
99.97% <= 6 milliseconds
99.98% <= 7 milliseconds
100.00% <= 8 milliseconds
100.00% <= 9 milliseconds
100.00% <= 9 milliseconds
68203.52 requests per second
```

Figure 1 **The Redis Benchmark**

for our solution, we leveraged Microsoft Azure. Within Azure, we deployed a combination of resources including virtual machines, Web apps, virtual networking, resource groups and availability sets to build a fully functional, high-quality solution.

The Data Layer

Because the majority of our solutions are built using the Microsoft stack, it makes sense that we leverage Microsoft SQL Server most of the time. However, the SLA requirements for this solution specified a service response time of less than 1 second per request, at a rate of about 6,000 requests per hour, on a dataset with more than 50 million records. Because traditional databases like SQL Server store data to disk and are bottlenecked by IOPS, we couldn't guarantee that every query would pass that requirement. To further complicate the matter, the subset of data we needed to expose already belonged to a traditional database containing terabytes of unrelated data. For that reason, we began evaluating solutions that could support large datasets quickly. That's when we discovered Redis.

Redis is an in-memory data structure store that supports multiple datatypes. The essential server requirements include a Linux distribution, enough RAM to encapsulate your data and enough disk space for data persistence. Redis can also be configured as a cluster, but this is more suitable for solutions with terabytes of data. Because our dataset was less than 50GB, we decided to keep it simple and set up a single master/slave configuration. To host this configuration, we created two CentOS 7.1 virtual machines within a virtual network on Azure. Each VM contained 56GB of memory, a static IP address and an SSH endpoint with AES256 encryption. Because both VMs shared an available set, Azure provided a 99.95 percent SLA guaranteed uptime. As an added bonus, all the resources we created were appended to a resource group created specifically for this solution, allowing us to monitor and manage billing on a monthly basis. Within just a few minutes, our VMs were deployed and available to us.

Standing up our Redis servers was simple and accomplished well within an hour. After downloading and installing the latest stable release onto our newly created servers, first and foremost we modified the configuration file to allow what Redis calls append-only file (AOF) persistence. In short, this means that every command sent to our instance is stored to disk. If the server were to restart, all commands are re-executed, restoring the server to its original state. To eliminate a bloated AOF, a BGREWRITEAOF command is triggered occasionally, rewriting the file with the shortest sequence of commands needed to rebuild the current dataset in memory. Additionally, we configured a maximum memory use of 50GB, creating a buffer and preventing Redis from consuming all system memory if too much data was loaded. With that in mind, if our solution ever needed more memory, we could easily resize the VM using the Azure Portal and update the configuration later. Next, we configured the slave instance to acknowledge the master, creating data replication. If the master instance were to become unavailable, the slave instance would be ready to serve our clients. Finally, we restarted our Redis servers for the configurations to take effect. Here are the configurations we used:

```
appendonly yes
maxmemory 50000000000
slaveof 10.0.0.x 6379
# cluster-enabled no
```

Data Loading and Benchmarking

The data we needed to load into Redis from our traditional database contained approximately 4GB of customer metadata. This metadata was updated daily, so we needed to create a process to transform it into a Redis datatype and bulk load it in order to keep our solution current. To execute this, we created an automated process to extract the daily change sets into Redis protocol-formatted files and transferred them to the master server using the SSH endpoint available. Within the master instance, we created a bash script to bulk load the files using pipe mode. To emphasize, pipe mode was a key element of our solution, as we were able to load 28 million records within 5 minutes. Note, however, that pipe mode is not yet compatible with a cluster implementation. During our initial prototyping phase, we discovered that loading 28 million records into a cluster would take hours because the records were transferred individually. This was a significant factor in our decision to keep the design a simple master/slave implementation. The pipe mode command and response for a single instance looks like this:

```
$ cat data.txt | redis-cli -pipe

All data transferred. Waiting for the last reply...
Last reply received from server.
errors: 0, replies: 1000000
```

After pipe mode executed, the response indicated how many errors and replies were received. We parsed this response within the script, archived the file and sent an e-mail notification to the appropriate technical teams. For each error, the techs could evaluate the extract and quickly identify any data that would need to be reloaded. After completion of the daily processing script, we benchmarked our implementation using the Redis benchmark utility. **Figure 1** shows the performance results of our solution, reassuring us that it would meet our SLA requirements.

Service Layer

It was critical that our client be the only consumer of our solution. Luckily, Azure makes this easy with Access Control Service, allowing us to create an OAuth provider specifically for our solution. Once implemented, our services required a specific token per request, with a new token regenerated after 5 minutes. As a side note, the token should always persist until expiration. Requesting a new token for every service call would essentially bottleneck your solution or, worse, make it inaccessible if you exceed the request limit specified by Azure. We strongly recommend that anyone leveraging this feature read the Azure documentation before implementing it in a production environment.

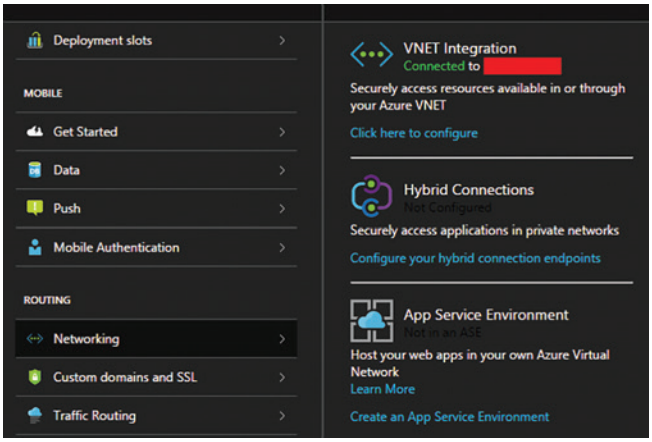
There are multiple C# Redis clients available, but we used StackExchange.Redis because we felt it was the most popular and mature. Because all of our data was stored in sets, we used the LRANGE command to query the data. To prevent a reconnection for each query, the connection used by the StackExchange.Redis was managed in a singleton pattern. The example in **Figure 2** demonstrates how we retrieve data.

In order to host our solution in Azure, we created a Web app. For maximum performance, it was critical that we deployed the Web app to the same region as the Redis instances. Once the app was deployed, we created a VNET connection to our Redis virtual network, allowing the service to obtain direct access to the data. This connection is demonstrated in **Figure 3**.

This Web app was configured to scale to 10 instances based on CPU usage. Because the traffic for our solution varied, the Web app would scale as necessary. As an added layer of security, an SSL certificate was applied to the app, along with IP filtering specifically for our client. Although the app was configured to scale automatically, it didn't do automatic failover. To maximize the availability of the solution to our client, we created a clone of the primary Web app, but placed it in a different region. Both apps were added to an Azure Traffic Manager, which we leveraged for automatic failover.

Figure 2 **Connecting to Redis and Retrieving Data**

```
private static Lazy<ConnectionMultiplexer> lazyConnection =
  new Lazy<ConnectionMultiplexer>(() => {
  return ConnectionMultiplexer.Connect(
    ConfigurationManager.AppSettings[Constants.AppSettings.RedisConnection]);
});

public static ConnectionMultiplexer Connection
{
  get
  {
    return lazyConnection.Value;
  }
}

public async static Task<MemberResponse> MemberLookup(MemberRequest request)
{
  MemberResponse response = new MemberResponse();

  try
  {
    if (request != null && request.customer != null && request.customer.Count() > 0)
    {
      foreach (var customer in request.customer)
      {
        Customer c = customer;
        RedisKey key = GetLookupKey(c);
        IDatabase db = Connection.GetDatabase();
        c.policies = await db.ListRangeAsync(key, 0, -1, CommandFlags.None);
        response.customer.Add(c);
      }

      response.success = true;
    }
    else
    {
      response.exceptions = new List<ServiceException>();
      response.exceptions.Add(Classes.ErrorCodes.Code_101_CustomerRequired);
      response.success = false;
        }
  }
  catch (Exception ex)
  {
    response.exceptions = new List<ServiceException>();
    response.exceptions.Add(Classes.ErrorCodes.Code_100_InternalError);
    response.success = false;
    Logging.LogException(ex);
  }

  response.executedOn =
    Utils.FormatEST(DateTime.UtcNow).ToString(Constants.DateTimeFormat);
  return response;
}
```



VNET Integration
Connected to [redacted]
Securely access resources available in or through your Azure VNET
Click here to configure

Hybrid Connections
Not Configured
Securely access applications in private networks
Configure your hybrid connection endpoints

App Service Environment
Not in an ASE
Host your web apps in your own Azure Virtual Network
Learn More
Create an App Service Environment

Figure 3 **VNET Integration Connection**

Finally, we purchased a custom domain and created a CNAME record pointing to our Traffic Manager URL, completing our implementation. In order to monitor the daily performance of our solution, we purchased an instance of New Relic directly from the Azure marketplace. By utilizing New Relic, we could quickly identify areas that needed improvement, as well as server availability.

Wrapping Up

By deploying this solution to Azure, we learned how to make different technology stacks work together quickly to provide a powerful, successful solution. Although deploying a solution to Azure doesn't eliminate server maintenance, if you follow the patterns in place, you will have success. Depending on the average duration of your solutions, you could save on hardware costs by moving all your solutions to the cloud.

By the end of the implementation, the average response time was 25 milliseconds for 50 concurrent users. Based on these results, the response time is extended roughly 10 milliseconds per 15 added concurrent users. ■

Sean Iannuzzi *is head of technology for Harte Hanks and has been in the technology industry for more than 20 years, playing a pivotal role in bridging the gap between technology and business visions for a plethora of social networking, Big Data, database solutions, cloud computing, e-commerce and financial applications of today. Iannuzzi has experience with more than 50 unique technology platforms, has achieved more than a dozen technical awards/certifications and specializes in driving technology direction and solutions to help achieve business objectives.*

Johnny Webb *is a software architect for Harte Hanks and has implemented high-quality solutions using cutting-edge technologies for well-known clients for the past 10 years. His expertise includes the full software development lifecycle, as well as the Microsoft .NET Framework, software architecture, cloud computing, Web development, mobile development, API development, SQL databases and NoSQL databases.*

Thanks to the following technical expert for reviewing this article:
Eric Riddle (Liquidhub)

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

JOIN US

March 7-11

May 16-19

June 13-16

SUPPORTED BY

Microsoft Visual Studio msdn magazine Visual Studio MAGAZINE

PRODUCED BY

1105 MEDIA INC
YOUR GROWTH. OUR BUSINESS.

CAMPAIGN FOR CODE 2016 ★ VISUAL STUDIO LIVE!

ON THE 2016 CAMPAIGN FOR CODE TRAIL!

October 3-6

August 8-12

December 5-9

September 26-29

vslive.com

TEST RUN

JAMES MCCAFFREY

Introduction to Spark for .NET Developers

Spark is an open source computing framework for Big Data, and it's becoming increasingly popular, especially in machine learning scenarios. In this article I'll describe how to install Spark on a machine running a Windows OS, and explain basic Spark functionality from a .NET developer's point of view.

The best way to see where this article is headed is to take a look at the demo interactive session shown in **Figure 1**. From a Windows command shell running in administrative mode, I started a Spark environment by issuing a spark-shell command.

The spark-shell command generates a Scala interpreter that runs in the shell and in turn issues a Scala prompt (scala>). Scala is a scripting language that's based on Java. There are other ways to interact with Spark, but using a Scala interpreter is the most common approach, in part because the Spark framework is written mostly in Scala. You can also interact with Spark by using Python language commands or by writing Java programs.

Notice the multiple warning messages in **Figure 1**. These messages are very common when running Spark because Spark has many optional components that, if not found, generate warnings. In general, warning messages can be ignored for simple scenarios.

The first command entered in the demo session is:

```
scala> val f = sc.textFile("README.md")
```

This can be loosely interpreted to mean, "Store into an immutable RDD object named f the contents of file README.md." Scala objects can be declared as val or var. Objects declared as val are immutable and can't change.

The Scala interpreter has a built-in Spark context object named sc, which is used to access Spark functionality. The textFile function loads the contents of a text file into a Spark data structure called a resilient distributed dataset (RDD). RDDs are the primary programming abstraction used in Spark. You can think of an RDD as somewhat similar to a .NET collection stored in RAM across several machines.

Text file README.md (the .md extension stands for markdown document) is located in the Spark root directory C:\spark_1_4_1. If your target file is located somewhere else, you can provide a full path such as C:\\Data\\ReadMeToo.txt.

The second command in the demo session is:

```
scala> val ff = f.filter(line => line.contains("Spark"))
```

This means, "Store into an immutable RDD object named ff only those lines from object f that have the word 'Spark' in them." The filter function accepts what's called a closure. You can think of a closure as something like an anonymous function. Here, the closure accepts a dummy string input parameter named line and returns true if line contains "Spark," false otherwise.

Because "line" is just a parameter name, I could've used any other name in the closure, for example:

```
ln => ln.contains("Spark")
```

Spark is case-sensitive, so the following would generate an error:

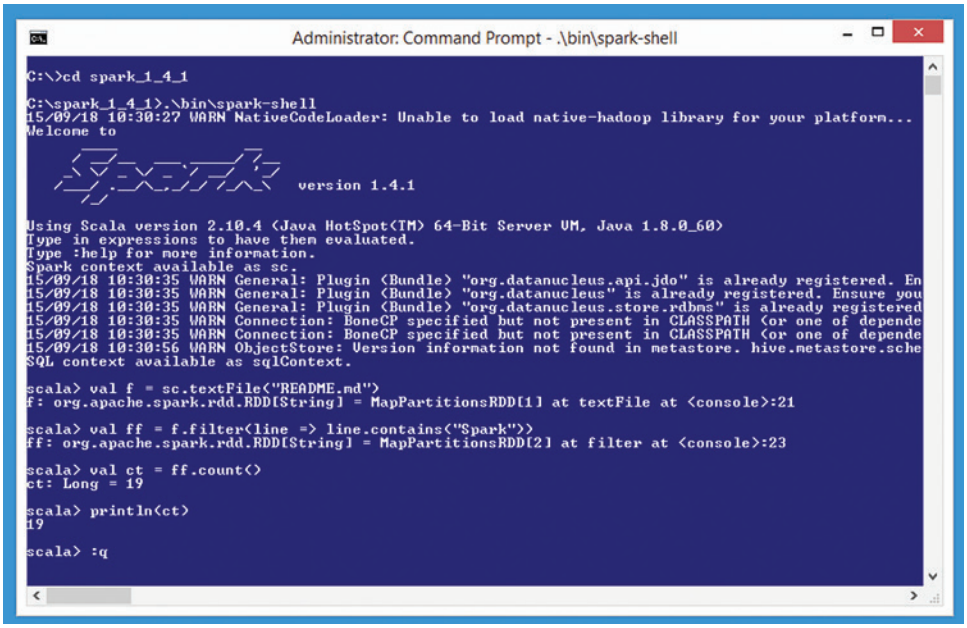
```
ln => ln.Contains("Spark")
```

Scala has some functional programming language characteristics, and it's possible to compose multiple commands. For example, the first two commands could be combined into one as:

```
val ff = sc.textFile("README.md").filter(line => lne.contains("Spark"))
```

The final three commands in the demo session are:

```
scala> val ct = ff.count()
scala> println(ct)
scala> :q
```



```
C:\>cd spark_1_4_1

C:\spark_1_4_1>.\bin\spark-shell
15/09/18 10:30:27 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform...
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 1.4.1
      /_/

Using Scala version 2.10.4 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_60)
Type in expressions to have them evaluated.
Type :help for more information.
Spark context available as sc.
15/09/18 10:30:35 WARN General: Plugin (Bundle) "org.datanucleus.api.jdo" is already registered. En
15/09/18 10:30:35 WARN General: Plugin (Bundle) "org.datanucleus" is already registered. Ensure you
15/09/18 10:30:35 WARN General: Plugin (Bundle) "org.datanucleus.store.rdbms" is already registered
15/09/18 10:30:35 WARN Connection: BoneCP specified but not present in CLASSPATH (or one of depende
15/09/18 10:30:35 WARN Connection: BoneCP specified but not present in CLASSPATH (or one of depende
15/09/18 10:30:56 WARN ObjectStore: Version information not found in metastore. hive.metastore.sche
SQL context available as sqlContext.

scala> val f = sc.textFile("README.md")
f: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[1] at textFile at <console>:21

scala> val ff = f.filter(line => line.contains("Spark"))
ff: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[2] at filter at <console>:23

scala> val ct = ff.count()
ct: Long = 19

scala> println(ct)
19

scala> :q
```

Figure 1 **Spark in Action**

Visual Studio MAGAZINE

READER'S CHOICE AWARD WINNER 2015

InstallAware

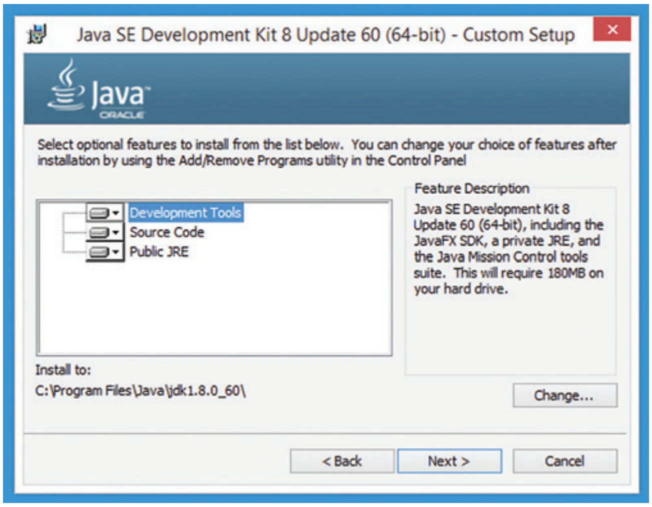
Visual Studio MAGAZINE 2 READER'S 2015 CHOICE

InstallAware X3: The One and Only

- 50% Faster and 174% Smaller than InstallShield(1)
- 50% Smaller than Microsoft(2)
- Pin to the New Windows 10 Start Menu and/or Taskbar(3)
- Automated, Multiple/Simultaneous Virtual Machine Testing(4)
- Fading/Sliding App-Savvy Wizard Transition FX(4)

1. As illustrated in the Installer Acid Test at www.installaware.com/news-installaware-x2-fastest-installer.htm.
2. InstallAware compresses better than even Microsoft's own already compressed sizes: www.installaware.com/web-deployment-superior-compression.htm.
3. Windows 10's recent updates prevented Taskbar pinning, now making InstallAware the only installer which can pin to the Taskbar, in addition to the Start Menu which was already an InstallAware exclusive.
4. Download the InstallAware setup at www.installaware.com and see for yourself how easy and powerful InstallAware is.

To learn more, please visit our website → www.installaware.com



Java SE Development Kit 8 Update 60 (64-bit) - Custom Setup

Select optional features to install from the list below. You can change your choice of features after installation by using the Add/Remove Programs utility in the Control Panel

Feature Description

Java SE Development Kit 8 Update 60 (64-bit), including the JavaFX SDK, a private JRE, and the Java Mission Control tools suite. This will require 180MB on your hard drive.

Install to:
C:\Program Files\Java\jdk1.8.0_60\

Figure 2 **The Default JDK Location**

The count function returns the number of items in an RDD, which in this case is the number of lines in file README.md that contain the word Spark. There are 19 such lines. To quit a Spark Scala session, you can type the :q command.

Installing Spark on a Windows Machine

There are four main steps for installing Spark on a Windows machine. First, you install a Java Development Kit (JDK) and the Java Runtime Environment (JRE). Second, you install the Scala language. Third, you install the Spark framework. And fourth, you configure the host machine system variables.

The Spark distribution comes in a compressed .tar format, so you'll need a utility to extract the Spark files. I recommend installing the open source 7-Zip program before you begin.

Although Spark and its components aren't formally supported on a wide range of Windows OS versions, I've successfully installed Spark on machines running Windows 7, 8, 10, and Server 2008 and 2012. The demo shown in **Figure 1** is running on a Windows 8.1 machine.

You install the JDK by running a self-extracting executable, which you can find by doing an Internet search. I used version jdk-8u60-windows-x64.exe.

When installing the 64-bit version of the JDK, the default installation directory is C:\Program Files\Java\jdkx.x.x_xx\, as shown in **Figure 2**. I recommend that you don't change the default location.

Installing the JDK also installs an associated JRE. After the installation finishes, the default Java parent directory will contain both a JDK directory and an associated JRE directory, as shown in **Figure 3**.

Note that your machine will likely also have a Java directory with one or more 32-bit JRE directories at C:\Program Files (x86). It's OK to have both 32-bit and 64-bit versions of JREs on your machine, but I recommend using only the 64-bit version of the Java JDK.

Installing Scala

The next step is to install the Scala language, but before you do so, you must go to the Spark download site (described in the next section of this article) and determine which version of Scala to install. The Scala version must be compatible with the Spark version you'll install in the following step.

Unfortunately, information about Scala-Spark version compatibility is scanty. When I was installing the Spark components (quite some time ago by the time you read this), the current version of Spark was 1.5.0, but I couldn't find any information about which version of Scala was compatible with that version of Spark. Therefore, I looked at the previous version of Spark, which was 1.4.1, and found some information on developer discussion Web sites that suggested that version 2.10.4 of Scala was likely compatible with version 1.4.1 of Spark.

Installing Scala is easy. The installation process simply involves running an .msi installer file.

The Scala installation wizard guides you through the process. Interestingly, the default installation directory for Scala is in 32-bit directory C:\Program Files (x86)\ rather than in 64-bit directory C:\Program Files\ (see **Figure 4**).

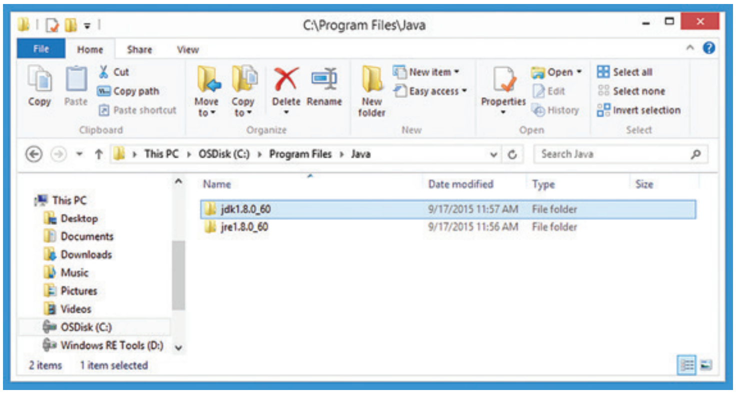
If you intend to interact with Spark by writing Java programs rather than by using Scala commands, you'll need to install an additional tool called the Scala Simple Build Tool (SBT). Interacting with Spark through compiled Java programs is much, much more difficult than using the interactive Scala.

Installing Spark

The next step is to install the Spark framework. First, though, be sure you have a utility program such as 7-Zip that can extract .tar format files. The Spark installation process is manual—you download a compressed folder to your local machine, extract the compressed files, and then copy the files to a root directory. This means that if you wish to uninstall Spark you simply delete the Spark files.

You can find the Spark site at spark.apache.org. The download page allows you to select a version and a package type. Spark is a computing framework and requires a distributed file system (DFS). By far the most common DFS used with the Spark framework is the Hadoop distributed file system (HDFS). For testing and experimentation purposes, such as the demo session in **Figure 1**, you can install Spark on a system that doesn't have a DFS. In this scenario, Spark will use the local file system.

If you haven't extracted .tar files before, you might find the process a bit confusing because you typically have to extract twice. First,



C:\Program Files\Java

This PC › OSDisk (C:) › Program Files › Java

Name	Date modified	Type	Size
jdk1.8.0_60	9/17/2015 11:57 AM	File folder	
jre1.8.0_60	9/17/2015 11:56 AM	File folder	

2 items 1 item selected

Figure 3 **Java JDK and JRE Installed to C:\Program Files\Java\**

READER'S CHOICE AWARD WINNER 2015

ActivePDF®
DRIVING THE DIGITAL BUSINESS

Visual Studio MAGAZINE 2 READER'S 2015 CHOICE

ActivePDF Toolkit™

Automate PDF Processing

10 year
Award Winner
by Visual Studio
Magazine
Readers Choice
Award

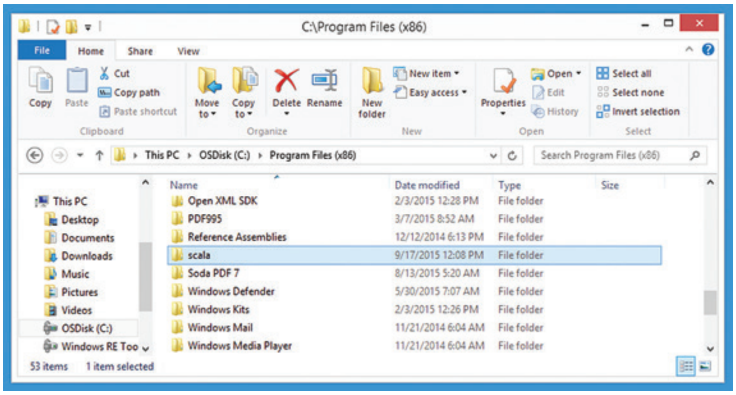
- Add XMP metadata content for document indexing and retrieval.
- Object stream compression for fast and efficient storage.
- AES 256 Encryption to protect confidential information.
- Stamp, stitch, append, form fill, merge, and more.

ActivePDF®
Get High Volume PDF Automation
Automate PDF processing & increase workflow productivity.
PDF
PDF SIMPLIFIED

Toolkit is a developer's library with hundreds of properties and methods that can knock out almost any PDF-related task. With simple, straight-forward .NET and COM interfaces, Toolkit hides the complexities of PDF, saving you days and even weeks of development time. Engineered in lightning-fast 32- and 64-bit native code, Toolkit pushes the envelope on performance, whether your need is just a few documents a day or thousands of pages a second. For 15+ years, ActivePDF has streamlined PDF automation to meet regulatory, business, and government compliance.

Get started Today with a Free Trial → www.ActivePDF.com

Toll Free US: 866 468 6733 | Outside US: +1 949 582 9002



Name	Date modified	Type	Size
Open XML SDK	2/3/2015 12:28 PM	File folder	
PDF995	3/7/2015 8:52 AM	File folder	
Reference Assemblies	12/12/2014 6:13 PM	File folder	
scala	9/17/2015 12:08 PM	File folder	
Soda PDF 7	8/13/2015 5:20 AM	File folder	
Windows Defender	5/30/2015 7:07 AM	File folder	
Windows Kits	2/3/2015 12:26 PM	File folder	
Windows Mail	11/21/2014 6:04 AM	File folder	
Windows Media Player	11/21/2014 6:04 AM	File folder	

Figure 4 **Scala Installs to C:\Program Files (x86)\scala\**

download the .tar file (mine was named spark-1.4.1-bin-hadoop2.6.tar) to any temporary directory (I used C:\Temp). Next, right-click on the .tar file and select "Extract files" from the context menu and extract to a new directory inside the temporary directory.

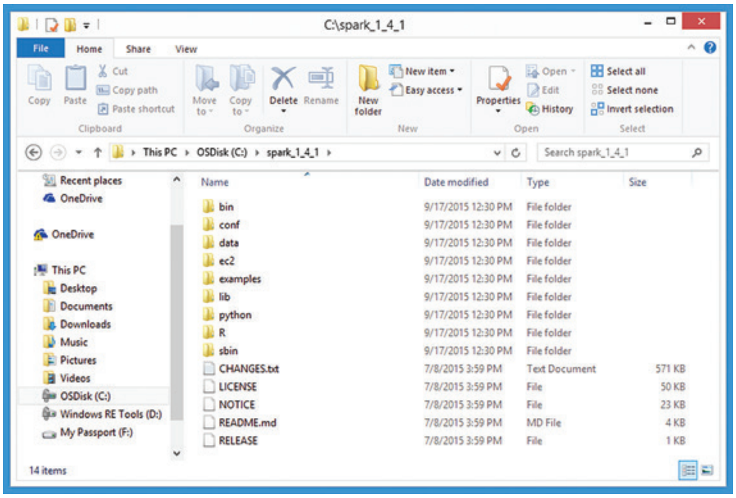
The first extraction process creates a new compressed file without any file extension (in my case spark-1.4.1-bin-hadoop2.6). Next, right-click on that new file and select "Extract files" again from the context menu and extract to a different directory. This second extraction will produce the Spark framework files.

Create a directory for the Spark framework files. A common convention is to create a directory named C:\spark_x_x_x, where the x values indicate the version. Using this convention, I created a C:\spark_1_4_1 directory and copied the extracted files into that directory, as shown in **Figure 5**.

Configuring Your Machine

After installing Java, Scala and Spark, the last step is to configure the host machine. This involves downloading a special utility file needed for Windows, setting three user-defined system environment variables, setting the system Path variable, and optionally modifying a Spark configuration file.

Running Spark on Windows requires that a special utility file named winutils.exe be in a local directory named C:\hadoop. You can find the file in several places by doing an Internet search. I created directory C:\hadoop and then found a copy of winutils.exe at http://public-repo-1.hortonworks.com/hdp-win-alpha/winutils.exe and downloaded the file into the directory.



Name	Date modified	Type	Size
bin	9/17/2015 12:30 PM	File folder	
conf	9/17/2015 12:30 PM	File folder	
data	9/17/2015 12:30 PM	File folder	
ec2	9/17/2015 12:30 PM	File folder	
examples	9/17/2015 12:30 PM	File folder	
lib	9/17/2015 12:30 PM	File folder	
python	9/17/2015 12:30 PM	File folder	
R	9/17/2015 12:30 PM	File folder	
sbin	9/17/2015 12:30 PM	File folder	
CHANGES.txt	7/8/2015 3:59 PM	Text Document	571 KB
LICENSE	7/8/2015 3:59 PM	File	50 KB
NOTICE	7/8/2015 3:59 PM	File	23 KB
README.md	7/8/2015 3:59 PM	MD File	4 KB
RELEASE	7/8/2015 3:59 PM	File	1 KB

Figure 5 **Manually Copy Extracted Spark Files to C:\spark_x_x_x\**

Next, create and set three user-defined system environment variables and modify the system Path variable. Go to Control Panel | System | Advanced System Settings | Advanced | Environment Variables. In the User Variables section, create three new variables with these names and values:

```
JAVA_HOME     C:\Program Files\Java\jdk1.8.0_60
SCALA_HOME    C:\Program Files (x86)\scala
HADOOP_HOME   C:\hadoop
```

Then, in the System Variables, edit the Path variable by adding the location of the Spark binaries, C:\spark_1_4_1\bin. Be careful; you *really* don't want to lose any values in the Path variable. Note that the Scala installation process will have already added the location of the Scala binaries for you (see **Figure 6**).

After you set up your system variables, I recommend modifying the Spark configuration file. Go to the root directory C:\spark_1_4_1\config and make a copy of file log4j.properties.template. Rename that copy by removing the .template extension. Edit the first configuration entry from log4j.rootCategory=INFO to log4j.rootCategory=WARN.

The idea is that by default Spark spews all kinds of informational messages. Changing the logging level from INFO to WARN greatly reduces the number of messages and makes interacting with Spark less messy.

The Hello World of Spark

The Hello World example of distributed computing is to calculate the number of different words in a data source. **Figure 7** shows the word-count example using Spark.

The Scala shell is sometimes called a read, evaluate, print loop (REPL) shell. You can clear the Scala REPL by typing CTRL+L. The first command in **Figure 7** loads the contents of file README.md into an RDD named f, as explained previously. In a realistic scenario, your data source could be a huge file spread across hundreds of machines, or could be in a distributed database such as Cassandra.

The next command is:

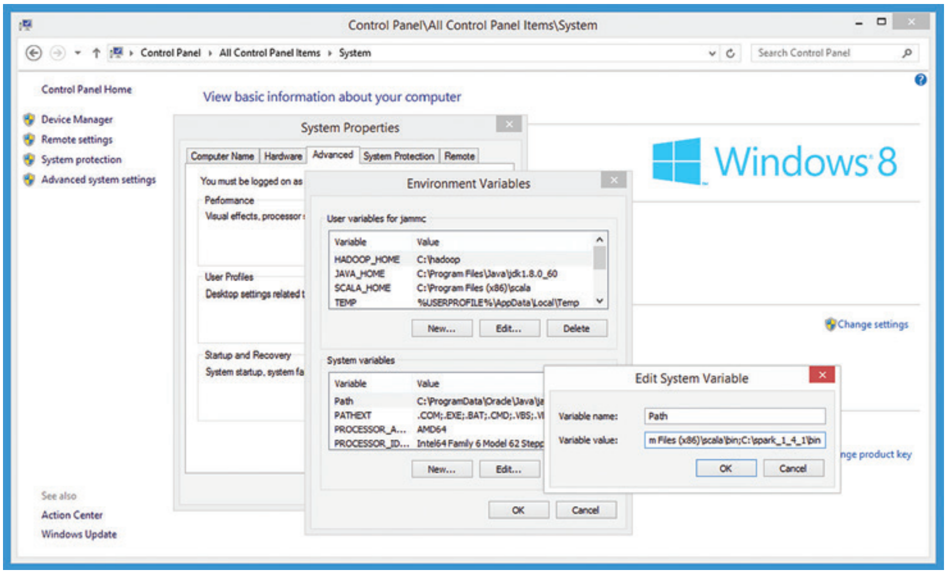
```
scala> val fm = f.flatMap(line => line.split(" "))
```

The flatMap function call splits each line in the f RDD object on blank space characters, so resulting RDD object fm will hold a collection of all the words in the file. From a developer's point of view, you can think of fm as something like a .NET List<string> collection.

The next command is:

```
scala> val m = fm.map(word => (word, 1))
```

The map function creates an RDD object that holds pairs of items, where each pair consists of a word and the integer value 1. You can see this more clearly if you issue an m.take(5) command. You'll see the first five words in file README.md, and a 1 value next to each word. From a developer's point of view, m is roughly a List<Pair> collection in which each Pair object consists of a string and an integer. The string (a word in README.md) is a key and the integer is a value, but unlike many key-value



User variables for jammc

Variable	Value
HADOOP_HOME	C:\hadoop
JAVA_HOME	C:\Program Files\Java\jdk1.8.0_60
SCALA_HOME	C:\Program Files (x86)\scala
TEMP	%USERPROFILE%\AppData\Local\Temp

System variables

Variable	Value
Path	C:\ProgramData\Oracle\Java\ja
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.V
PROCESSOR_A...	AMD64
PROCESSOR_ID...	Intel64 Family 6 Model 62 Stepp

Variable name: Path

Variable value: m Files (x86)\scala\bin;C:\spark_1_4_1\bin

Figure 6 **Configuring Your System**

pairs in the Microsoft .NET Framework, duplicate key values are allowed in Spark. RDD objects that hold key-value pairs are sometimes called pair RDDs to distinguish them from ordinary RDDs.

The next command is:

```
scala> val cts = m.reduceByKey((a,b) => a + b)
```

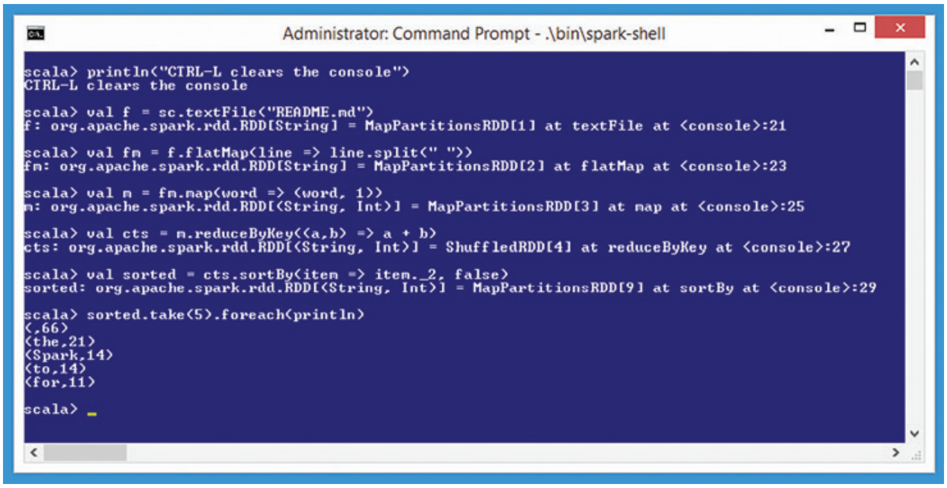
The reduceByKey function combines the items in object m by adding the integer values associated with equal key values. If you did a cts.take(10) you'd see 10 of the words in README.md followed by the number of times each word occurs in the file. You might also notice that the words in object cts aren't necessarily in any particular order.

The reduceByKey function accepts a closure. You can use an alternate Scala shortcut notation:

```
scala> val cts = m.reduceByKey(_ + _)
```

The underscore is a parameter wild card, so the syntax can be interpreted as "add whatever two values are received."

Notice that this word-count example uses the map function followed by the reduceByKey function. This is an example of the MapReduce paradigm.



```
scala> println("CTRL-L clears the console")
CTRL-L clears the console

scala> val f = sc.textFile("README.md")
f: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[1] at textFile at <console>:21

scala> val fm = f.flatMap(line => line.split(" "))
fm: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[2] at flatMap at <console>:23

scala> val m = fm.map(word => (word, 1))
m: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[3] at map at <console>:25

scala> val cts = m.reduceByKey((a,b) => a + b)
cts: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[4] at reduceByKey at <console>:27

scala> val sorted = cts.sortBy(item => item._2, false)
sorted: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[9] at sortBy at <console>:29

scala> sorted.take(5).foreach(println)
(,66)
(the,21)
(Spark,14)
(to,14)
(for,11)

scala> _
```

Figure 7 **Word Count Example Using Spark**

The next command is:

```
scala> val sorted =
  cts.sortBy(item => item._2, false)
```

This command sorts the item in the cts RDD, based on the second value (the integer count) of the items. The false argument means to sort in descending order, in other words from highest count to lowest. The Scala shortcut syntax form of the sort command would be:

```
scala> val sorted = cts.sortBy(_._2, false)
```

Because Scala has many functional language characteristics and uses a lot of symbols instead of keywords, it's possible to write Scala code that's very unintuitive.

The final command in the Hello World example is to display the results:

```
scala> sorted.take(5).foreach(println)
```

This means, "Fetch the first five objects in the RDD object named sorted, iterate over that collection, applying the println function to each item." The results are:

```
(,66)
(the,21)
(Spark,14)
(to,14)
(for,11)
```

This means there are 66 occurrences of the empty/null word in README.md, 21 occurrences of the word "the," 14 occurrences of "Spark" and so on.

Wrapping Up

The information presented in this article should get you up and running if you want to experiment with Spark on a Windows machine. Spark is a relatively new technology (created at UC Berkeley in 2009), but interest in Spark has increased dramatically over the past few months, at least among my colleagues.

In a 2014 competition among Big Data processing frameworks, Spark set a new performance record, easily beating the previous record set by a Hadoop system the year before. Because of its exceptional performance characteristics, Spark is particularly well-suited for use with machine learning systems. Spark supports an open source library of machine learning algorithms named MLib. ■

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 jammc@microsoft.com.*

Thanks to the following Microsoft technical experts for reviewing this article: *Gaz Iqbal and Umesh Madan*

THE WORKING PROGRAMMER — TED NEWARD

How To Be MEAN: Express Input

Welcome back, “Nodeists.” (I established that as the semi-official term of endearment for those who use Node.js on a regular basis. If you don’t care for it, drop an e-mail or a tweet with a better suggestion, bearing in mind my other two ideas were “Noderati” or “Nodeferatu.”)

In the previous installment, the application had grown to include some output capabilities, in the form of Web API endpoints for obtaining either the collection of persons (my resource for this application; I seem to be building some kind of people database) or the individual person via an arbitrary “id” given as part of the URL. It’s time to start processing input—the ability to put a new person into the system, remove a person from the system and update an existing person. In some ways, these are “just” new URL endpoints to the application, but there are a few new tricks I want to talk about along the way.

As I mentioned last time, those who are interested in seeing the latest-and-greatest of the code being written as part of this series can visit the Microsoft Azure site that holds the latest of this series’ code (msdn-mean.azurewebsites.net). It’s likely that the text here is out of sync with what’s on the site, given publication schedules, but if anything, the site will be ahead of what’s here, giving readers a look ahead at what’s to come next.

Speaking of the last column, as of the last installment, the code can display existing people in the database, but there’s no modification of them whatsoever yet. Because that’s usually a critical part of any online system, let’s add the “CUD” to the “R” to finish out the CRUD.

Another One Bites the Dust ...

The easiest one to implement first is the “D” in CRUD: Delete. Setting up a route in Express requires only that the code use delete instead of the get used last time:

```
app.delete('/persons/:personId', deletePerson);
```

Recall from my last column that the “:personId” is a parameter that will be available on the Express “req” (request) object in the associated function (deletePerson), and picked up by the personId middleware function also described last time, so that you can know which person to remove from the system.

Speaking of deletePerson, the implementation is relatively straightforward, using the lodash remove method to search the in-memory database and remove the individual described (see **Figure 1**).

The personId middleware will pick up :personId (whatever its value), find the appropriate person object in the database and put that as a person property on the incoming request (“req”) object, so if there’s no req.person, it means nobody by that ID was found in the database. When you send the response, however, instead of converting the result to JSON via the built-in JSON.stringify method from the previous articles, use the jsonp method of converting the data to a JSON format—or, to be more accurate, a JSONP (JSON with padding) format, which is widely considered to be the superior (and safer) way of sending JSON back to browsers. Notice how the method calls “chain”; that means you can use one fluent line of code to send back either a 200 with the JSON response consisting of a message that the person was deleted successfully, or a 404 with the message that the person’s ID wasn’t recognized.

> The easiest one to implement first is the “D” in CRUD: Delete.

Get into My Database

Next, you should probably support putting people into the database. Again, this is pretty straightforward routing: Conventionally, Web API advocates suggest using a POST to the resource-collection URL (/persons) as the means to insert into the database, so do that:

```
app.post('/persons', insertPerson);
```

Not particularly exciting. But a new wrinkle emerges: In order to insert the person into the system, you’re going to need to pick out the JSON for the person data out of the incoming request. One way to do that would be to grab the entire request body using the req.body parameter, but then you’re left with the onerous (and slightly dangerous) task of parsing the JSON into an object suitable for storage. Instead, this is where the Node.js community leans on its extensive collection of libraries, and sure enough, there’s a good

Figure 1 **Using the Lodash Remove Method**

```
var deletePerson = function(req, res) {
  if (req.person) {
    debug("Removing", req.person.firstName, req.person.lastName);
    _.remove(personData, function(it) {
      it.id === req.person.id;
    });
    debug("personData=", personData);
    var response = { message: "Deleted successfully" };
    res.status(200).jsonp(response);
  }
  else {
    var response = { message: "Unrecognized person identifier"};
    res.status(404).jsonp(response);
  }
};
```

INTEL® PROCESSOR AND COPROCESSOR OVERVIEW
This contract of employment is entered into
1. Commencement of Employment
2. Probation Period
3. Position and Section Employed

library out there that handles that, called body-parser. Install it ("npm install body-parser" in the same directory as package.json) and then you can reference it and tell the Express app object to use it:

```
// Load modules
  var express = require('express'),
  bodyParser = require('body-parser'),
  debug = require('debug')('app'),
  _ = require("lodash");

// Create express instance
var app = express();
app.use(bodyParser.json());
```

Recall that discussion earlier around :personId, about how Express allows you to create middleware functions that do a little bit of work silently as part of the pipeline? That's exactly what the body-parser library does—it installs a number of "hooks" (for lack of a better term) to process the body of the request in a variety of forms. Here, you're asking it to parse JSON, but it can also support URL encoding, parsing everything as a giant string (making it easier to parse CSV, for example), or grabbing everything "raw" into a Node.js Buffer (presumably because the incoming data is binary of some form). Because most of what we care about right now is handled with JSON, just using that is sufficient.

The insertPerson function, then, is actually really anticlimactic:

```
var insertPerson = function(req, res) {
  var person = req.body;
  debug("Received", person);
  person.id = personData.length + 1;
  personData.push(person);
  res.status(200).jsonp(person);
};
```

(OK, the unique-id generation code could use a lot of work, but remember the goal is to eventually end up using MongoDB, so let's not spend a ton of time worrying about that.)

When returning from insertPerson, the code sends back the complete Person object just inserted, including its new id field; this isn't a completely standard convention, but in my own code I've found it helpful. This way the client is aware of any additional validation/correction/server-side-augmentation of a submitted entity, such as the id field in this case.

How Do I POST Thee?

By the way, one of the interesting parts about building a Web API this way (as opposed to a more traditional Web app) is that some of the simple tricks for doing quick-and-dirty testing aren't available to you. For example, how do you quickly test to see if the insertPerson function's working? To be sure, there's a ton of support for automating testing in Node.js, and that's coming in a future column, but for now, it's easiest to use either a browser plug-in (like Chrome Postman) or, for the die-hard command-line fan, the cURL freeware utility that comes pre-installed on OS X and most Linux images. If you don't like either of those, there's a number of others, all of which stretch across the entire gamut of complexity, from the ad hoc to the automated, including one of my favorites, Runscope (runscope.com), a cloud-based system for automated testing of your API endpoints, for something running 24x7.

Figure 2 **Updating a Person Already in the Database**

```
var updatePerson = function(req, res) {
  if (req.person) {
    var originalPerson = req.person;
    var incomingPerson = req.body;
    var newPerson = _.merge(originalPerson, incomingPerson);

    res.status(200).jsonp(newPerson);
  }
  else {
    res.status(404).jsonp({ message: "Unrecognized person identifier" });
  }
};

// ...

app.put('/persons/:personId', updatePerson);
```

Regardless of what you use, if you don't like it, move on quickly, because there are literally thousands of them out there.

Keeping Up-to-Date with the Neighbors

Finally, the app needs to support updating a person already in the database; in many ways, this is a combination of the complexity of delete (finding the right person in the database) and insert (parsing the incoming JSON), but the middleware already installed handles most of that already, as shown in **Figure 2**.

The key thing in **Figure 2** is the use of the lodash method merge, which enumerates across the properties of the second parameter and either overwrites the property of the same name on the first parameter, or else adds it in. It's an easy way to copy the attributes of the second onto the first without destroying and recreating the first object (which is important in this case, because the first object is inside the personData array that serves as our impromptu database).

For the curious, the true branch of the if statement could be condensed down into one line of code:

```
var updatePerson = function(req, res) {
  if (req.person) {
    res.status(200).jsonp(_.merge(req.person, req.body));
  }
  else {
    res.status(404).jsonp({ message: "Unrecognized person identifier" });
  }
};
```

... but whether that's more or less readable is up to you to decide.

Wrapping Up

For anybody who's counting, the app.js file that began life in part two of this series is now exactly 100 lines of code (which includes about a half-dozen lines of comments), and now supports a full CRUD Web API against an in-memory database. That's not bad for a century's worth in your text editor. But there's more to do: The system needs to move away from using the in-memory database toward using MongoDB, and in order to do that without breaking all of the clients that are already using this highly prized Web API, there needs to be tests. Nobody wants to have to run tests over and over again by hand, so the next steps are to add automated tests to ensure the transition to MongoDB is seamless and simple from the client's perspective. In the meantime ... happy coding! ■

TED NEWARD *is the CTO of iTrellis, a Seattle-based polytechnology consulting firm. He has written more than 100 articles, is an F# MVP, INETA speaker, and has authored or co-authored a dozen books. Reach him at ted@tedneward.com if you're interested in having him come work with your team, or read his blog at tedneward.com.*

THANKS to the following technical expert for reviewing this article:
Shawn Wildermuth

Visual Studio
MAGAZINE

READER'S CHOICE AWARD WINNER 2015

ComponentSpace

Visual Studio MAGAZINE
3
READER'S
2015
CHOICE

SAML SSO Solutions

Fully functional, enterprise ready

Want your SAML single sign-on integration up and running within hours? ComponentSpace unique SSO solutions make it possible...

- ✔ Thorough, easy to understand product documentation
- ✔ Set-up and run fully functional SAML SSO examples in minutes
- ✔ Easy to understand API
- ✔ Unique lightweight components make it faster, easier and more cost effective

COST EFFECTIVE, EASY-TO-USE SAML SSO SOLUTIONS
SSO
FROM ANYWHERE TO ANYWHERE

Download your free 30-day trial → www.componentspace.com

Essential .NET

MARK MICHAELIS

Designing C# 7

By the time you read this, the C# 7 design team will have been discussing, planning, experimenting and programming for about a year. In this installment, I'll sample some of the ideas they've been exploring.

In reviewing, be mindful that at this time these are still ideas for what to include in C# 7. Some of the ideas the team has simply talked about, while others have made it as far as experimental implementations. Regardless, none of the concepts are set; many may never see the light of day; and even those that are further down the road could be cut in the final stages of language finalization.

Declaring Nullable and Non-Nullable Reference Types

Perhaps some of the most predominant ideas in the C# 7 discussion are those relating to further improvement in working with null—along the same lines as the null-conditional operator of C# 6.0. One of the simplest improvements might be complier or analyzer verification that accessing a nullable type instance would be prefaced with a check that the type is not, in fact, null.

On occasions when null isn't desirable on a reference type, what if you could avoid null entirely? The idea would be to declare the intent that a reference type would allow nulls (string?) or avoid null (string!). In theory, it might even be possible to assume that all reference type declarations in new code are, by default, non-nullable. However, as pointed out by my "Essential C# 6.0" book co-author, Eric Lippert, ensuring a reference type would never be null at compile time is prohibitively difficult (bit.ly/1Rd5ekS). Even so, what would be possible is to identify scenarios where a type could potentially be null and yet dereferenced without checking that it isn't. Or, scenarios where a type might be assigned null in spite of the declared intent that it not be null.

For even more benefit, the team is discussing the possibility of leveraging the non-nullable type declaration on a parameter such that it would automatically generate a null check (although this would likely have to be an opt-in decision to avoid any undesirable performance degradation unless it can be made at compile time).

(Ironically, C# 2.0 added nullable value types because there are numerous occasions—such as data retrieved from a database—where it's necessary to let an integer contain null. And now, in C# 7, the team is looking to support the opposite for reference types.)

One interesting consideration with reference type support for things non-nullable (for example, string! text) is what the implementation would be in Common Intermediate Language (CIL). The two most popular proposals are whether to map it to a NonNullable<T> type syntax or to leverage attributes as in [Nullable] string text. The latter is currently the preferred approach.

Tuples

Tuples is another feature under consideration for C# 7. This is a topic that has come up on multiple occasions for earlier versions of the language, but still hasn't quite made it to production. The idea is that it would be possible to declare types in sets such that a declaration can contain more than one value and, similarly, methods can return more than one value. Consider the following sample code to understand the concept:

```
public class Person
{
  public readonly (string firstName, int lastName) Names; // a tuple
  public Person((string FirstName, string LastName)) names, int Age)
  {
    Names = names;
  }
}
```

As the listing shows, with tuples support, you can declare a type as a tuple—having two (or more) values. This can be leveraged anywhere a data type can be leveraged—including as a field, parameter, variable declaration or even a method return. For example, the following code snippet would return a tuple from a method:

```
public (string FirstName, string LastName) GetNames(string! fullName)
{
  string[] names = fullName.Split(" ", 2);
  return (names[0], names[1]);
}
public void Main()
{
  // ...
  (string first, string last) = GetNames("Inigo Montoya");
  // ...
}
```

In this listing there's a method that returns a tuple and a variable declaration of first and last that the GetNames result is assigned to. Note that the assignment is based on the order within the tuple (not the names of the receiving variables). Considering some of the alternative approaches we have to use today—an array or collection, a custom type, out parameters—tuples are an attractive option.

There are numerous options that could go along with tuples. Here are a few under consideration:

- Tuples could have named or unnamed properties, as in:

  ```
  var name = ("Inigo", "Montoya")
  ```

  and:

  ```
  var name = (first: "John", last: "Doe")
  ```

- The result could be an anonymous type or explicit variables, as in:

```
var name = (first: "John", last: "Doe")
```

or:

```
(string first, string last) = GetNames("Inigo Montoya")
```

- You could potentially convert an array to a tuple, as in:

```
var names = new[]{ "Inigo", "Montoya" }
```

- You can access the individual tuple items by name, as in:

```
Console.WriteLine($”My name is { names.first } { names.last }.”);
```

- Data types could be inferred where they’re not identified explicitly (following the same approach used by anonymous types in general)

Although there are complications with tuples, for the most part they follow structures already well-established within the language, so they have pretty strong support for inclusion in C# 7.

Pattern Matching

Pattern matching is also a frequent topic within the C# 7 design team’s discussion. Perhaps one of the more understandable renderings of this would be expanded switch (and if) statements that supported expression patterns in the case statements, rather than just constants. (To correspond with the expanded case statement, the switch expression type wouldn’t be limited to types that have corresponding constant values, either). With pattern matching, you could query the switch expression for a pattern, such as whether the switch expression was a specific type, a type with a particular member, or even a type that matched a specific “pattern” or expression. For example, consider how obj might be of type Point with an x value greater than 2:

```
object obj;
// ...
switch(obj) {
  case 42:
    // ...
  case Color.Red:
    // ...
  case string s:
    // ...
  case Point(int x, 42) where (Y > 42):
    // ...
  case Point(490, 42): // fine
    // ...
  default:
    // ...
}
```

Interestingly, given expressions as case statements, it would also be necessary to allow expressions as arguments on goto case statements.

To support the case of type Point, there would need to be some type of member on Point that handled the pattern matching. In this case, what’s needed is a member that takes two arguments of type int. A member, for example, such as:

```
public static bool operator is (Point self out int x, out int y) {...}
```

Note that without the where expression, case Point(490, 42) could never be reached, causing the compiler to issue an error or warning.

One of the limiting factors of the switch statement is that it doesn’t return a value, but rather executes a code block. An added feature of pattern matching might be support for a switchexpression that returns a value, as in:

```
string text = match (e) { pattern => expression; ... ; default => expression }
```

Similarly, the is operator could support pattern matching, allowing not only a type check but possible support for a more generic query as to whether particular members on a type exist.

Records

In a continuation of the abbreviated "constructor" declaration syntax considered (but ultimately rejected) in C# 6.0, there is support for embedding the constructor declaration within the class definition, a concept known as "records." For example, consider the following declaration:

```
class Person(string Name, int Age);
```

This simple statement would automatically generate the following:

- A constructor:

  ```
  public Person(string Name, int Age)
  {
    this.Name = Name;
    this.Age = Age;
  }
  ```
- Read-only properties, thus creating an immutable type
- Equality implementations (such as GetHashCode, Equals, operator ==, operator != and so forth)
- A default implementation of ToString
- Pattern-matching support of the "is" operator

Although a significant amount of code is generated (considering only one short line of code created it all), the hope is that it might provide a correspondingly significant shortcut to manually coding what is essentially boilerplate implementations. Furthermore, all of the code can be thought of as "default" in that explicitly implementing any of it would take precedence and preclude generation of the same member.

One of the more problematic issues associated with records is how to handle serialization. Presumably leveraging records as data transfer objects (DTOs) is fairly typical and yet it isn't clear what, if anything, can be done to support the serialization of such records.

In association with the records is support for with expressions. With expressions allow the instantiation of a new object based on an existing object. Given the person object declaration, for example, you could create a new instance via the following with expression:

```
Person inigo = new Person("Inigo Montoya", 42);
Person humperdink = inigo with { Name = "Prince Humperdink" };
```

The generated code corresponding to the with expression would be something like:

```
Person humperdink = new Person(Name: "Prince Humperdink", Age: inigo.42 );
```

An alternative suggestion, however, is that rather than depending on the signature of the constructor for the with expression, it might be preferable to translate it to the invocation of a With method, as in:

```
Person humperdink = inigo.With(Name: "Prince Humperdink", Age: inigo.42);
```

Async Streams

To enhance support of async in C# 7, the notion of processing asynchronous sequences is intriguing. For example, given an IAsyncEnumerable, with a Current property and a Task<bool> MoveNextAsync method, you could iterate over the IAsyncEnumerable instance using foreach and have the compiler take care of invoking each member in the stream asynchronously—performing an await to find out if there's another element in the sequence (possibly a channel) to process. There are a number of caveats to this that have to be evaluated, the least of which is the potential LINQ bloat that could occur with all the LINQ standard query operators that return IAsyncEnumerable. Additionally, it's not certain how to expose CancellationToken support and even Task.ConfigureAwait.

C# on the Command Line

As a lover of how Windows PowerShell makes the Microsoft .NET Framework available in a command-line interface (CLI), one area I'm particularly intrigued by (possibly my favorite feature under consideration) is support for using C# on the command line; it's a concept more generically referred to as support for Read, Evaluate, Print, Loop (REPL). As one would hope, REPL support would be accompanied by C# scripting that doesn't require all the usual formality (such as class declaration) in trivial scenarios that don't need such ceremony. Without a compile step, REPL would require new directives for referencing assemblies and NuGet packages, along with importing additional files. The current proposal under discussion would support:

- #r to reference an additional assembly or NuGet package. A variation would be #r!, which would even allow access to internal members, albeit with some constraints. (This is intended for scenarios where you're accessing assemblies for which you have the source code.)
- #l to include entire directories (similar to F#).
- #load to import an additional C# script file, in the same way you'd add it to your project except now order is important. (Note that importing a .cs file might not be supported because namespaces aren't allowed in C# script.)
- #time to turn on performance diagnostics while executing.

You can expect the first version of C# REPL to be released with Visual Studio 2015 Update 1 (along with an updated Interactive Window that supports the same feature set). For more information check out Itl.tc/CSREPL, along with my column next month.

Wrapping Up

With a year's worth of material, there's far too much to explore all that the design team has been doing, and even with the ideas I touched on, there are a lot more details (both caveats and advantages) that need to be considered. Hopefully, however, you now have an idea of what the team is exploring and how they're looking to improve on the already brilliant C# language. If you'd like to review the C# 7 design notes directly, and possibly provide your own feedback, you can jump into the discussion at bit.ly/CSharp7DesignNotes.

Mark Michaelis *is founder of IntelliTect, where he serves as its chief technical architect and trainer. For nearly two decades he has been a Microsoft MVP, and a Microsoft Regional Director since 2007. Michaelis serves on several Microsoft software design review teams, including C#, Microsoft Azure, SharePoint and Visual Studio ALM. He speaks at developer conferences and has written numerous books including his most recent, "Essential C# 6.0 (5th Edition)" (itl.tc/EssentialCSharp). Contact him on Facebook at facebook.com/Mark.Michaelis, on his blog at IntelliTect.com/Mark, on Twitter: @markmichaelis or via e-mail at mark@IntelliTect.com.*

Thanks to the following Microsoft technical expert for reviewing this article: *Mads Torgerson*

What You Need to Know About Windows 10 App Development

I thought I'd offer some insight into Windows 10 app development to help you understand and use it more efficiently. While some of the features covered here apply to the average user, it's definitely a good idea for the developer to know about them, both as a user and for the benefit of the software the developer builds.

Visual Studio 2015

The IDE that most Microsoft developers use is Visual Studio. The first few things you'll notice that have changed about Visual Studio 2015 include the new, simplified installer, as well as the ability to sign in using multiple accounts. This is great for the many developers who are consultants and full-time employees who need to use the corporate network during the day and publish apps to the store in the evenings.

There are impressive amounts of third-party tools available in the Visual Studio installation. Xamarin is available out of the box, as is all the software you need for serious cross-platform development. You must select the option to include it in the installation, though. In addition to C# with Xamarin, there are options to install Java for Android and C-based languages for both iOS and Android.

As usual, Visual Studio editions and licensing almost seem to require a Ph.D. to understand or remember which features belong to which editions. Fortunately, you can learn more and compare the Visual Studio 2015 offerings at bit.ly/1COm2fP.

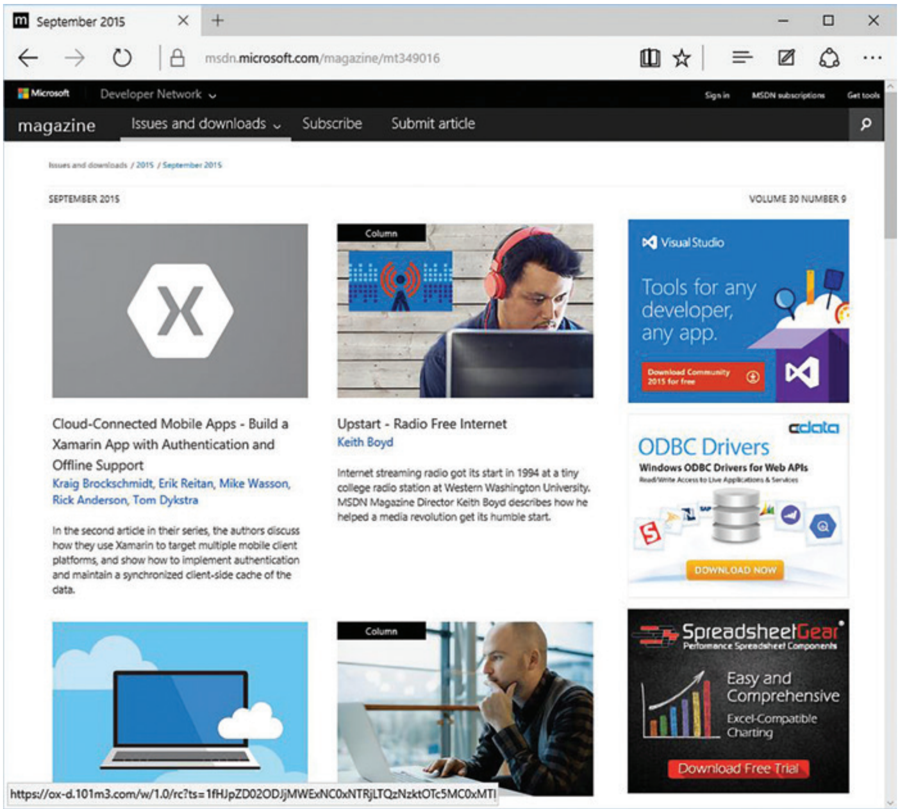
Every new release of Visual Studio brings with it a new set of templates. In ASP.NET, templates now enable loosely coupled Web site building with dependency injection available throughout the entire ASP.NET MVC 6 app. Apps deployed to the Microsoft Store focus heavily on the Universal Windows Platform (UWP) app concept to enable developers to build apps with a foundational common code base for all Windows OSes and devices that run them. Use C#, Visual Basic, JavaScript or C++ to create UWP apps. There's more on UWP apps later in this column.

Get an Edge up on the New Edge Browser

One of the most obvious and talked-about changes in Windows is the Edge browser, which sports a smooth and fast browsing experience. It's quite apparent after using Edge for even a short time that it's not your father's Internet Explorer. To start, there have been thousands of improvements to the browsing experience (bit.ly/1G49Cwe). The most obvious changes are the smooth face of Edge, a start page full of customizable content, and the overall look and feel. **Figure 1** shows the *MSDN Magazine* homepage in the Edge browser on Windows 10.

First and foremost, the Edge browser is about interoperability.

Every browser needs one or more engines to process the HTML, CSS and JavaScript that comprise Web pages today. Therefore, the Edge team designed a new HTML processing engine called EdgeHTML. Interoperability provides several benefits including the ability to create HTML that displays well on a variety of devices



September 2015
msdn.microsoft.com/magazine/mt349016
Microsoft
Developer Network
Sign in
MSDN subscriptions
Get tools
magazine
Issues and downloads
Subscribe
Submit article
Issues and downloads / 2015 / September 2015
SEPTEMBER 2015
VOLUME 30 NUMBER 9
Column
Visual Studio
Tools for any developer, any app.
Download Community 2015 for free
cdata
ODBC Drivers
Windows ODBC Drivers for Web APIs
Read/Write Access to Live Applications & Services
DOWNLOAD NOW
Column
SpreadsheetGear
Performance Spreadsheet Components
Easy and Comprehensive
Excel-Compatible Charting
Download Free Trial

Cloud-Connected Mobile Apps - Build a Xamarin App with Authentication and Offline Support
Kraig Brockschmidt, Erik Reitan, Mike Wasson, Rick Anderson, Tom Dykstra

In the second article in their series, the authors discuss how they use Xamarin to target multiple mobile client platforms, and show how to implement authentication and maintain a synchronized client-side cache of the data.

Upstart - Radio Free Internet
Keith Boyd

Internet streaming radio got its start in 1994 at a tiny college radio station at Western Washington University. MSDN Magazine Director Keith Boyd describes how he helped a media revolution get its humble start.

https://ox-d.101m3.com/w/1.0/rc?ts=1fHJpZD02ODJjMWExNC0xNTRjLTQzNzktOTc5MC0xMTI

Figure 1 **The Edge Browser with Its Sleek, Smooth Experience**

and form factors while seamlessly developing cross-platform Web apps. The team has implemented 45 new HTML standards in EdgeHTML (bit.ly/1G49Cwe).

You'll find the same Chakra engine from Internet Explorer. Chakra is extremely fast and performs well, so it makes sense to keep it and make some tweaks. Its speed is due to several factors, a major one being a technique called Graphics Processing Unit (GPU) offloading. Chakra offloads, or sends script over to the GPU for processing. This means that script runs on the GPU while HTML and friends run on the CPU where processing normally happens. Once the Internet Explorer team had implemented this feature, the other browsers quickly started implementing it, as well. You can find details about the performance of Chakra in Edge at the Windows Blog (bit.ly/1XONpt0).

You can dig into the Edge Developer Guide at bit.ly/1jwFYec, where you'll find a full breakdown of how the F12 tools were revamped with efficient use in mind. Some of the new and exciting features are the ability to set XHR breakpoints and view pages in the DOM Explorer.

Universal Windows Platform

There is a significant amount of Web traffic on smartphones and tablets, though there are many desktop power Web surfers, too. Web sites and apps absolutely must support multiple devices and form factors nowadays.

Windows 10 is the true universal Windows OS family. Now you can build and maintain one code base, one package and one-submission-to-one-store for all Windows 10 devices. That means everything from phones, tablets and laptops to desktops, ultrabooks and servers. You name it, UWP apps can run on it.

Windows Notifications Are News to Me

Who doesn't love the fact that your phone, computer, and other devices can alert and remind you about everything and anything, regardless of whether it's important to you? Your neighbor's friend's sister's 36th birthday? Got it! Now you can remind your users about all sorts of things related to your app. That's if the user allows it. Some users turn off notifications. For those who like them, they can turn on Tips about Windows, as well as system-wide or app-specific notifications in the system settings.

There's a new Action Center for your app's notifications, and its icon is located in the bottom right of the screen in the Windows notification area (aka the system tray). Clicking on the notifications icon shows a modern flyout window containing touch-friendly tiles below a list of awaiting messages. Notifications from your app will display in the notification area, if the user approves, of course.

The code to create notifications essentially remains the same as it has before, where the notification displays and its overall aesthetics are controlled by Windows. Of course, you may customize the look and feel somewhat, by selecting one of many pre-defined notification templates. There are so many to choose from, there should be no problem finding one to meet your needs:

```
ToastTemplateType toastTemplate = ToastTemplateType.ToastImageAndText01;
XmlDocument toastXml =
  ToastNotificationManager.GetTemplateContent(toastTemplate);
```

For more information, see the MSDN Library article, "Working with Tiles, Badges, and Toast Notifications (XAML)," at bit.ly/1LPogJw.

Talk to Cortana

Windows Phone introduced Cortana to the world. Cortana is the Microsoft speech-enabled digital assistant that lets you use voice commands to perform a variety of tasks such as scheduling appointments, getting directions and fetching the latest news and weather. Cortana can assist you in many daily activities. Such valuable and helpful software deserves an SDK, and in Windows 10, there are new features such as background voice commands and continuous dictation. You can even enable text-to-speech (TTS) capabilities with the Speech SDK. Using voice commands and speech recognition technology is an excellent way to build a higher-quality product with more than just a visual UI.

To build voice-enabled Windows apps, you create and register Voice Definition Files (.vcd) that list the commands, words and phrases available in your app, just as you would have in earlier versions of Windows and Visual Studio. Then you can write your app in C#, JavaScript or any language you want and let Cortana translate those commands into spoken form. The code is quite simple and looks something like that in **Figure 2**, which overrides the OnActivated event to detect which command has been issued so the app can perform an action.

Figure 2 **The Contents of a .vcd File and Accompanying C# Code**

```
<?xml version="1.0" encoding="utf-8"?>
<VoiceCommands xmlns="http://schemas.microsoft.com/voicecommands/1.0">
  <CommandSet xml:lang="en-us">
    <CommandPrefix> Options </CommandPrefix>
    <Example> Show Options</Example>
    <Command Name="showOptions">
      <Example> Show options </Example>
      <ListenFor> [Show] {optionViews} </ListenFor>
      <Feedback> Showing {optionViews} </Feedback>
      <Navigate Target="/options.xaml"/>
    </Command>
    <PhraseList Label="optionViews">
      <Item> today's specials </Item>
      <Item> best sellers </Item>
    </PhraseList>
  </CommandSet>
  <!-- Other CommandSets for other languages -->
</VoiceCommands>

protected override void OnActivated(IActivatedEventArgs args)
  {
    if (args.Kind ==
      Windows.ApplicationModel.Activation.ActivationKind.VoiceCommand)
    {
      var commandArgs =
        args as Windows.ApplicationModel.Activation.VoiceCommandActivatedEventArgs;
        Windows.Media.SpeechRecognition.SpeechRecognitionResult
      speechRecognitionResult =
        commandArgs.Result;

      string voiceCommandName = speechRecognitionResult.RulePath[0];
      string textSpoken = speechRecognitionResult.Text;
      string navigationTarget =
        speechRecognitionResult.SemanticInterpretation.
        Properties["NavigationTarget"][0];

      switch (voiceCommandName)
      {
        case "showOptions":
          // EventReminder(textSpoken, navigationTarget);
          break;
        // default:
        // There is no match for the voice command name
      }
    }
  }
```

The Windows Store

The new Windows Store has something for everyone. For businesses, the Windows Store enables administrators to showcase apps for their employees. They can even distribute select apps from the Windows Store to deploy private line-of-business apps. Additionally, purchase orders are now an accepted payment method. In Windows 10, the store now offers subscriptions as an additional monetization option.

As part of the Store updates, the Windows 10 Microsoft Advertising SDK now offers support for video ads. Some marketing experts say that video and multimedia sell more than text, so now you can test that hypothesis in your app. Fortunately, because Microsoft announced installation tracking as a new feature, you have a way to do so.

Continuum

All these UWP app developments are not just about the cross-device experience, though. Continuum on Windows 10 detects when you want to switch between usage modes in multi-function devices. Consider Francine Flyer, a user on an airplane who has completed some work in desktop mode on a Surface, and now wants to watch a movie. Francine Flyer effortlessly switches between desktop and tablet mode by simply detaching the keyboard. Windows will notice and ask if she wants to switch to a more touch-friendly scenario. Flyer taps "Yes," Windows moves into a touch-friendly mode and she enjoys her movie without a pesky keyboard or mouse getting in the way. As you can see, Continuum is great for Surfaces, hybrid or convertible laptops/tablets and any kind of multi-function device. Even if you have a laptop that has a touchscreen without convertible capabilities, it will still benefit when switching between modes. More than just the large devices can benefit from Continuum. Continuum for phones will let the user use your apps like desktop apps when connecting his phone to a wireless keyboard, mouse and screen. With so many phablet-sized devices around, Continuum will certainly be put to good use.

A New Start (Menu)

Definitely the most obvious change and probably the one most argued about is the Windows Start Menu. There are a few UI "troublemakers," so to speak, in computing history, and the Windows Start Menu is certainly one of them. With every new version of Windows comes people who love any changes to the Start Menu, but just as many who don't. When the modern experience was introduced in Windows 8, many hailed the new design, but many clung to the classic paradigms and resisted change. Now, in Windows 10, there are some very big changes to the Start Menu.

As it is now, moving the tiles to the Start Menu is a much better design than before, plus the workflow between the desktop and start page was a tiny bit turbulent. Combine that with Continuum, and we now have an enhanced and better Start Menu, with a Start Page only where it should be–on touch devices such as tablets, phones and so on. If you prefer keyboard shortcuts, remember that the Windows keyboard key has been around for a while. You can start typing the name of your app, or what you want to do, then Windows will find the app or perform the action you want.

Adapt to an Adaptive UX

Adaptive development on the Windows family of OSes is similar conceptually to responsive design for the Web. However, adaptive development targets entire families of devices, while responsive development targets screen-size ranges. In the last few years, it's been impossible to keep up with the explosion of different devices on the market. There are some for whom buying a new smartphone causes analysis paralysis. Can you imagine if they had to develop software for all those devices? Fortunately, Windows 10 determines what device is hosting the app at run time, or how the user is using the app, and adjusts the UX accordingly. This means elements such as flyouts or other controls may be automatically resized, or a larger or smaller font applied, depending on the resolution. Before designing an adaptive solution, be sure to consult the MSDN Library article, "Device Primer for Universal Windows Platform (UWP) Apps," at bit.ly/1MpspVh.

One Windows Platform

One of the showcase features for Windows 10 is the advent of the UWP app. A UWP app is one that you can deploy to all Windows OSes, all from a single code base! Usually this strategy only works for the back end and logic; however, it's a much smoother process for building the UI, as well. That's because rather than targeting different OSes and writing multiple versions of the same UI code, you target entire device families, so there's less hassle in building a UI. However, if you only want to target one specific OS, you may do so. Otherwise, you will construct the UI so that it works well within a minimum and maximum size that device families tend to fit with Continuum. There are several API and control changes in XAML, as well as in the Windows Library for JavaScript (WinJS). XAML, in particular, sports a new Calendar control, along with new adaptive panel controls in which to put your calendar.

As a developer, you take advantage of a single solution in Visual Studio to deal with because of the underlying code base called the One Windows Platform. This single solution model enables the adaptive controls and technology to adjust to the various device families with little to no code.

Wrapping Up

Windows keeps reimagining the UX landscape to bring users what they want. There's many new and exciting features you can build into your app, such as changes in the APIs and controls. It's easy to forget speech is input, too, so don't forget to consider Cortana for your next app. From the Windows Store to Visual Studio, Windows 10 delivers on a smooth development and UX. ■

Rachel Appel *is a consultant, author, mentor and former Microsoft employee with more than 20 years of experience in the IT industry. She speaks at top industry conferences such as Visual Studio Live!, DevConnections, MIX and more. Her expertise lies within developing solutions that align business and technology focusing on the Microsoft dev stack and open Web. For more about Appel, visit her Web site at rachelappel.com.*

Thanks to the following Microsoft technical expert for reviewing this article: *Frank La Vigne*

DON'T GET ME STARTED

DAVID S. PLATT

Refactoring Higher Education

I've been teaching software development to students for a quarter of a century now, starting in 1991 with the 16-bit Windows SDK in C. The subject matter has evolved continuously over the years—Win32, COM, Microsoft .NET Framework, Microsoft Azure and so on. But my teaching methods, and those of my colleagues, remained mostly unchanged until the Great Recession seven years ago, when training was the first item cut from development budgets. That economic squeeze forced us to reevaluate not only our existing methods, but our entire role in the industry.

Teaching software development to programmers originally centered on transmitting specific facts in a certain order: first call this function, then call that one. RegisterWindowClass. CreateWindow. And my all-time favorite, CoMarshalInterThreadInterfaceInStream. My students were happy for a live instructor to pick those nuggets out of the 1,000-page manual, because the alternative was reading the whole thing themselves.

> Teaching software development to programmers originally centered on transmitting specific facts in a certain order: first call this function, then call that one.

But because this data transfer is almost entirely one-way, technology advanced to the point that students didn't need a live human being to impart it. Videos, such as those from Microsoft Virtual Academy or Pluralsight, could fulfill much of this function at a far lower cost. That approach scaled better than live instruction, as movies scale better than live theater.

Good, effective education still needs live instructors, but in different places doing different things. We aren't broadcasting "what" or "how" we used to. Instead, we're reserving our expensive contact hours for "why," or, "how does this relate to other things," or, "where do we go from here," along with the ever-popular, "WTF was that?"

So I've changed my instruction process for live classes. It's less about spewing a one-size-fits-all set of function calls into the void. It's more about helping my students splice the new technology into their business logic—hmm, let's look at your code, where should the module boundaries be and why? What sorts of tests should we have? How should we figure out who our users are and what they need? It's an entirely different type of teaching and learning.

I have good company in making this shift. Harvard Medical School is making similar adjustments to its curriculum, starting with this year's freshman class. Today's future doctors aren't subject to the "sage-on-a-stage" spouting, "Gentlemen, here are the 12 cranial nerves: Olfactory, Optic, Oculomotor … [and so on]." Instead, students memorize these basic facts from instructional videos outside of class. Professors then spend their contact time working through real-life problem analysis, such as: "A patient's left eyebrow is up, like Mr. Spock expressing doubt, and it won't come down. Which cranial nerve would you look at first, and why? How would you start doing that? What else would you look for, and why? Mr. Smith, you look confused. Surely you haven't forgotten your cranial nerves, have you? Ms. Jones, please help him out." You can read the whole story at bit.ly/1S7UR2G.

By the way, that article states that the "videos are between five and eight minutes … because students' attention spans don't last much longer." Does that scare you as much as it does me? I wonder if we geeks today have longer or shorter attention spans.

I use the hands-on model in my User eXperience class. It's not about implementing this feature or that one (a color gradient, say). It's about deciding what ought to be implemented in order to make users happier and more productive. I rarely teach an in-house class unless the client has an actual design project to work on during the course labs. I encourage the students in my public classes to bring their own projects, and if they don't, I assign them one.

As I wrote in my very first Don't Get Me Started column (msdn.com/magazine/ee309884), a bad computer program demands that its users become more like a computer. But the very best programs use computers to do what computers do well, clearing the track for humans to do what only humans can.

Education needs to evolve in a similar way, and the best human instructors are starting to realize this. It's one thing to put a specific fact into a student's head. It's an entirely different thing, a much more satisfying thing, to help them put that fact to work. ■

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.*

Develop For **Any** Device In **Any** Platform

A **COMPLETE COLLECTION** OF POWERFUL, MODULAR .NET UI CONTROLS FOR **MOBILE**, **WEB**, AND **DESKTOP**

NEW RELEASE: more speed, more power, more flexibility than ever!

Newly-Added Controls

FlexReport
Fast, flexible reporting tool for modern business.

Financial Chart
A full service control for creating stock market applications directly out-of-the-box.

FlexGrid for UWP
Our legendary datagrid control, redesigned for modern, touch-based enterprise apps. (Beta)

Wide Range of Controls

Grids & Data Management

Data Visualization

Reporting & Documentation

Scheduling

WinForms | WPF | ASP.NET Web Forms | ASP.NET MVC | WinRT | LightSwitch | Silverlight | ActiveX

ComponentOne Studio

Download your free 30-day trial at
www.ComponentOne.com

Visual Studio Microsoft Partner

Arduino
Succinctly
by Marko Švaljek
FakeItE
Succinctly
by Michael McCarthy
Spark
Succinctly
by Marko Švaljek
Syncfusion | Technology Resource Portal
ASP.NET
Multitenant
Applications
Succinctly
y Ricardo Peres
T4
Succinctly
y Nick Harrison