THE MICROSOFT JOURNAL FOR DEVELOPERS

JUNE 2017 VOL 32 NO 6

msdn magazine

Boost IoT Development.....12

DevExpress®

Your Next Great Desktop App Starts Here

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

Download your free 30-day trial
an experience the DevExpress difference today.

devexpress.com/try

THE MICROSOFT JOURNAL FOR DEVELOPERS

JUNE 2017 VOL 32 NO 6

msdn magazine

Boost IoT Development.....12

COLUMNS

Microsoft


TRAFFIC BY REGION
SPEND
CONVERSIONS
VISITS
UNIQUE VISITORS
2.3M
CONVERSION RATE
10%
CAMPAIGN HEALTH
20%
TOTAL
200K
CONVERSIONS
SALES DASHBOARD
MONTH TO DATE 9/1 - 9/16
$303,876
Actual
$340K
Forecast
REGION | DIVISION
REGION TARGET FORECAST
Americas $873,030
Asia $580,200
Africa $359,876
Australia $200,387
Europe $100,876
TOTAL $2,114,369
LANDSCAPE VIEW FOR MORE DETAILS
SALES HISTORY
upcoming
Starting 4/12 Nova No
Starting 4/15 Haddad
Starting 5/10 Hancock
tasks by resource
Prototype Interactions
Design Research
Oversee Prototyping
Bind Data to Server
Review Prototypes
Review Prototypes
Projected End 3/13
Projected End 3/13
Projected End 3/13
Projected End 3/12
Projected End 3/10
Projected End 3/13
90 hours
55 hours
67 hours
60 hours
35 hours
10 hours
hours worked
3/2 - 3/6 (this week)
Johnson
Lasa Ap
Fivel Ma
Nova No
Haddad
Hancock
today 3/3
current
0.56 (-2.12%)
1Y 2Y 5Y 10Y MAX
Stock Price
CHART
Daily Range
Annual Range
Open
Volume
Adobe Systems Inc
2013-07-26 $47.64
Microsoft Corporation
2013-07-26 $31.62
Intel Corporation
2013-07-26 $23.26
Jun 2013
Jul 2013
Aug 2013
Sales Volume

msdn magazine

JUNE 2017 VOLUME 32 NUMBER 6

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

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

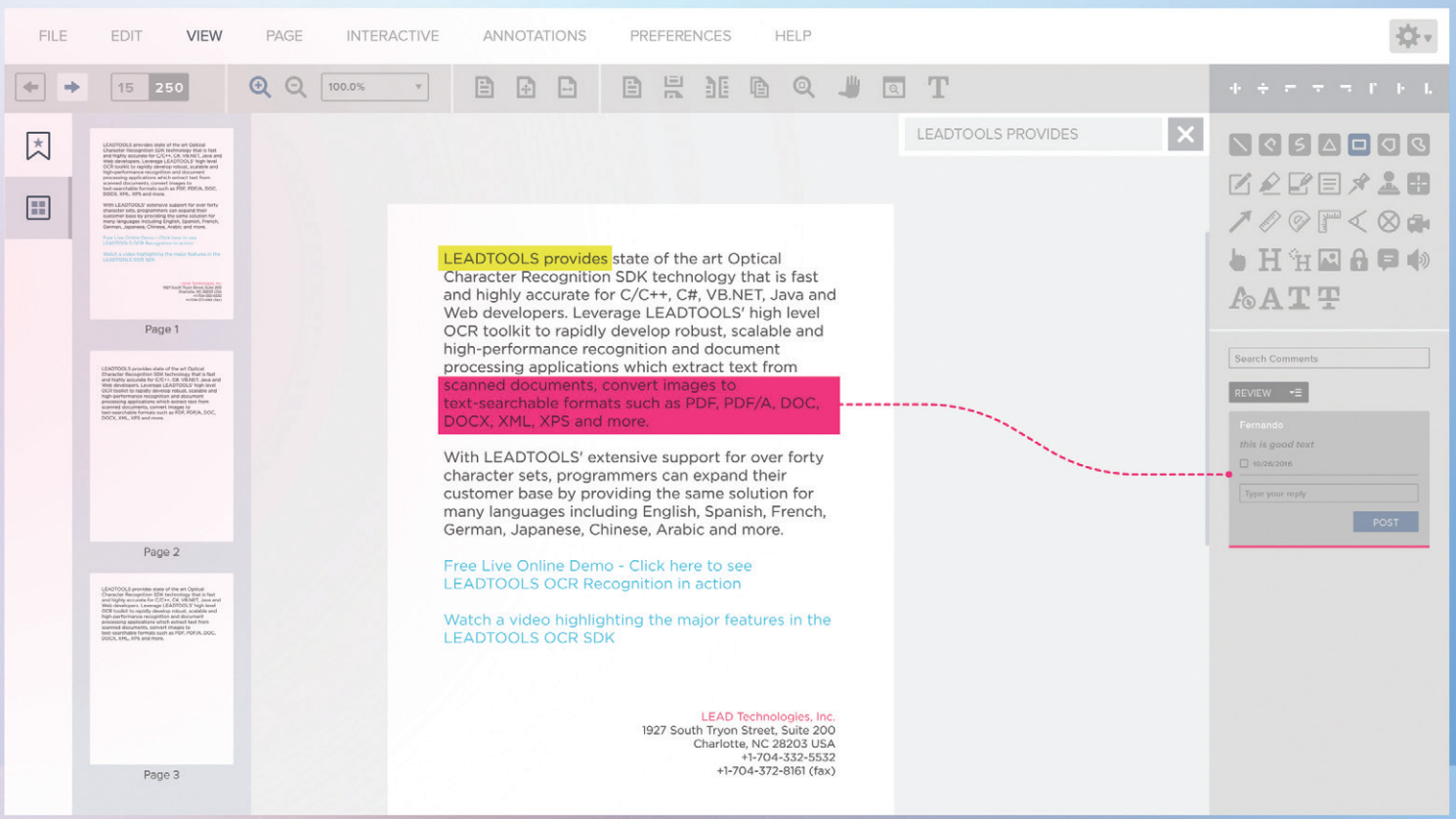
1906 2006 100 AMERICAN BUSINESS MEDIA

WPA Western Publications Association

LEADTOOLS®

THE WORLD LEADER IN IMAGING SDKs

The leading **Document Viewer SDK** just got better.



With only a few lines of code, developers can use the **LEADTOOLS** HTML5/JavaScript Document Viewer SDK to add rich document viewing features to any project, including text search, annotation, memory-efficient paging, inertial scrolling, and vector display.

- CREATE, ORGANIZE & EXPORT A DOCUMENT FROM MULTIPLE FILES
- PDF, DOC, DOCX, RTF, HTML, SVG, and XPS
- ANNOTATIONS & MARKUP
- DRAG AND DROP INTERFACE
- SEARCHABLE TEXT

SDKs for WEB | SDKs for MOBILE | SDKs for DESKTOP | SDKs for SERVER

.NET Windows API Java WinRT Linux iOS macOS Android JavaScript

DOWNLOAD OUR 60 DAY EVALUATION
WWW.LEADTOOLS.COM

LEAD TECHNOLOGIES INCORPORATED

SALES@LEADTOOLS.COM
800.637.1840

Editor's Note

MICHAEL DESMOND

Grokking Git

Back in the August 2016 issue, *MSDN Magazine* published a feature by Jonathan Waldman titled "Commit to Git: Source Control in Visual Studio 2015" (msdn.com/magazine/mt767697). That article went on to be among the 10-most-visited feature articles for all of 2016, based on first-month page views, and was the most-trafficked DevOps-themed article we've ever published.

Clearly, Git struck a nerve with readers. So when Waldman pitched a two-part exploration of Git internals for Visual Studio developers, it didn't take me long to say yes. Working closely with Microsoft experts Kraig Brockschmidt and Ralph Squillace, Waldman's first part of the series, "Git Internals for Visual Studio Developers," explores the version control system (VCS) and how it differs from the traditional, centralized VCSes familiar to most Windows developers.

Along the way, Waldman realized that developers who understand the directed acyclic graph (DAG) used to store commit objects in Git are in much better position to grasp advanced Git operations and concepts.

Waldman began using Git as his VCS with Visual Studio 2013 and, he says, "grew to love working against a decentralized repo." But he faced a learning curve as he experimented with Git and pieced together guidance and resources. When Microsoft released enhanced Git support with Visual Studio 2015, Waldman dove right in, eventually writing his Commit to Git feature article in *MSDN Magazine* based on his experience.

"There's been a dearth of good material about Git for the Microsoft technology professional," he says, noting that most Git tutorials and instruction rely on the Unix shell command-line interface. "I think *MSDN Magazine* readers enjoyed finally seeing material about Git that leverages the Windows-based Visual Studio IDE and PowerShell/Command Prompt command-line interfaces."

Along the way, Waldman realized that developers who understand the directed acyclic graph (DAG) used to store commit objects in Git are in much better position to grasp advanced Git operations and concepts. He learned also that developers often misunderstand what a Git branch is and how it works. In fact, a large part of the challenge in grokking Git boils down to vocabulary, where familiar terms like check in/check out, branch and merge represent approaches different from those in a centralized VCS.

"I think it's essential to first understand the repo from Git's perspective, using a purely repo-centric vocabulary. Once you do that, you realize that Git simply stores and manages commit objects on a [DAG], and that's all," Waldman says. "Everything else has to do with user-initiated actions that compare or manage those objects. And when it comes to describing those actions, it is critical for team members to adopt a lean Git vocabulary with clear definitions and then use it formally and consistently."

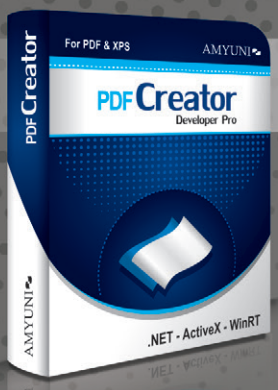
Waldman's two-part exploration this month and next aim to break down these barriers. This month's article introduces the Git DAG and explores commit objects and branching. Next month, Waldman plans to extend his coverage to what he describes as "foundational knowledge" about Git.

"I plan to explore exactly what happens when you check in and check out code—with particular emphasis on Git's three-tree architecture," Waldman says. "I'll also cover how Git detects code changes, and provide additional details about branching and merging."

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

Switch to Amyuni PDF

AMYUNI



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

NEW v5.5

- 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

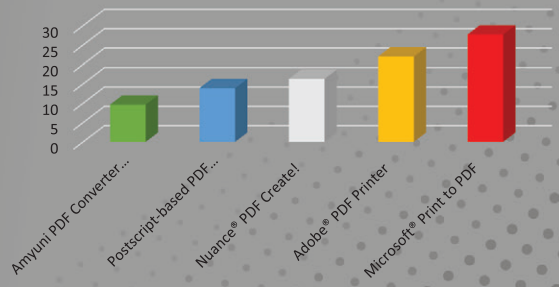
Other Developer Components from Amyuni®

- 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

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

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

Stop Stalling: 8 Steps to Better Productivity

Procrastination is not a victimless crime. The psychic costs of putting off tasks over time can add up. And the more we put them off—even simple tasks like bug fixes or unit tests—the greater the burden becomes. Every time we think, "I should really do that," we add to the burden.

Some simple strategies can help you get your work done, and move you from living in regret to living in freedom.

1. Win elsewhere. If I haven't been able to tackle a problem head-on for a while, it means that—psychologically—it's making me deeply uncomfortable. It's too hot to handle directly. So, I set a time—say one week—and in that one week I just do other, less-overwhelming tasks. This way, I build my "get-things-done" muscle, because, so far, I've just been building my "get-stuck" muscle. Once I build this momentum, I switch to the postponed task.

2. Outsource it. My time's valuable. Even if I'm working for someone else, I like to put a dollar figure on my time—let's say $500/hour. I then quantify all of the projects I do: this one's a $2,000 project, that one's a $10,000 bug fix or this one's a $10 design item.

> To get over "coder's block," just doing the tiniest action imaginable—maybe simply writing one line of code—is all that's needed!

So, if I'm going to spend five hours on a task, I make sure I feel like it'll deliver at least a $2,500 value ($500/hr x 5 hours). Otherwise, I outsource, delegate, reassign or swap my task with someone else, or I just eliminate it. See next step.

3. Eliminate it. Sometimes tasks stay unfinished because at a deep level, I know it's pointless. If I've been stuck for a long time, I ask myself: "How does this task move me forward on my top-two goals right now?"

Often, it doesn't. I want to do it because I set an artificial goal for myself to do it, or maybe because I'd wanted to impress somebody. In such cases, I just eliminate the task altogether.

4. Promise no more. Often, I've ended up in this position because I said "yes" without thinking. If I keep making this mistake, I'll end up with a plethora of hanging projects. So, whenever somebody asks me to take on a new project, I start saying no in creative ways:

- "Let me think about it."
- "I'll check with my manager first."
- "I don't think that'll work for me right now. I have deadlines to meet." (This statement is almost always true, because I usually have at least one deadline I'm working toward.)

5. Inaction is work. Doing nothing is a valid choice. I've been making that choice all along, without realizing it. I can decide to make that my permanent choice for this project, and consider it done by dropping it!

6. Hop a little. To get over "coder's block," just doing the tiniest action imaginable—maybe simply writing one line of code—is all that's needed! This chips away at conditioned resistance to challenging work, and channels our innate talent for high concentration.

The power of tiny actions is revealed in the strong hold that distractions have over us. We may decide to take a peek at a news headline (a tiny action), and before you know it, two hours have passed!

7. Maximize playfulness. To quote one of my favorite authors, "The remedy for weakness is not brooding over weakness, but thinking of strength. Teach [others] of the strength that is already within them." All of us have a natural talent for high concentration and excellent productivity, just waiting to be tapped.

This is a game with two players: me, and my mind. If I feel like my mind and I are conspiring together *positively*, I can be more successful. I do this by making this a fun game for us and introducing artificial challenges into the situation:

- What's the absolute *least* amount of work I could do for this project and still make progress?
- How much of this could I do in 20 minutes instead of 20 hours?

8. Batch. I group together tasks, and I do them at once. Then, over time, I can trickle out the results to others. Phases of lower productivity can't exist without other phases of hyper-focused productivity. With batching, you can milk the productive moments for all they're worth, then not worry as much about the less-productive phases. ■

KRISHNAN RANGACHARI *helps brilliant high-performers have amazing careers. Tell him your story at RadicalShifts.com.*

We didn't invent the Internet...

...but our components help **you** power the apps that bring it to business.

TOOLS • COMPONENTS • ENTERPRISE ADAPTERS

- **E-Business**
  AS2, AS4, EDI/X12, OFTP …
- **Credit Card Processing**
  Authorize.Net, ACH, 3-D Secure …
- **Shipping & Tracking**
  FedEx, UPS, USPS …
- **Accounting & Banking**
  QuickBooks, OFX, SAP …
- **Internet Business**
  Amazon, PayPal, Google …
- **Internet Protocols**
  FTP, SMTP, IMAP, POP, WebDav …
- **Secure Connectivity**
  SSH, SFTP, SSL, Certificates …
- **Encryption & Certificates**
  X.509, OpenPGP, SHA, S/MIME …
- **Network Management**
  SNMP, MIB, LDAP, Monitoring …
- **Compression**
  Zip, Gzip, Jar, AES, 7Zip …

Our **Red Carpet Subscription** includes all product lines + updates for one year.

The Market Leader in Internet Communications, Security, & E-Business Components

Each day, as you click around the Web or use any connected application, chances are that directly or indirectly some bits are flowing through applications that use our components, on a server, on a device, or right on your desktop. It's your code and our code working together to move data, information, and business. We give you the most robust suite of components for adding Internet Communications, Security, and E-Business Connectivity to any application, on any platform, anywhere, and you do the rest. For more than 20 years, we have had one goal: to provide the very best connectivity solutions for our professional developer customers. With more than 100,000 developers worldwide using our software and millions of installations in almost every Fortune 500 and Global 2000 company, our business is to connect business, one application at a time.

connectivity powered by

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

Microsoft .NET | Java | C++ | Delphi | C++ Builder | ActiveX | Linux/Unix | macOS | Android | iOS | PHP | Xamarin | Node.js

JULIE LERMAN

Visual Studio Code: Create a Database IDE with the MSSQL Extension

While Visual Studio is still my go-to IDE for heavy-duty .NET development, I've fallen head over heels with Visual Studio Code (VS Code), whether I'm working in C# or JavaScript or Node and whether I'm working in Windows or in macOS. Of course, there are many other languages and frameworks supported by VS Code and its myriad extensions, but these are my current toolsets. I'm not alone in my love for VS Code. During the Visual Studio 2017 launch in March 2017, Microsoft announced that VS Code had 1.3 million active monthly users. What's really great is that, as with any cross-platform tool, each member of your team can pick their favorite OS, yet all can still use the same coding tools.

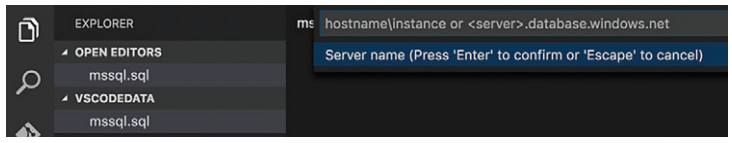


Figure 1 **Connecting to a SQL Database with the mssql Extension**

And then there's data—there's always data involved. I spent years and years working only with SQL Server and coding Windows applications. But in the past few years, I've expanded my horizons—not only with a new IDE (VS Code), but also new databases and new platforms (as I type this on my MacBook Pro).

My foray into VS Code began as part of my wanderings into Node.js, which you've witnessed in this very column. That was originally on Windows. But because VS Code is cross-platform (built with Electron), I eventually found myself going back and forth, working on the same code, sometimes in Windows, sometimes in macOS with GitHub as the common denominator. Thanks to the C# extension and the cross-platform nature of .NET Core, I eventually reached beyond Node.js to write .NET Core apps with EF Core in both environments. When in Visual Studio, I rely heavily on the built-in SQL Server Data Tools and the SQL CE/SQLite Toolbox extension to explore most of the data my apps are creating. But with VS Code, I need something external to explore the data. When using Windows and the super lightweight VS Code, it never felt right to open up SQL Server Management Studio, which is anything but lightweight. For other databases, whether on my Windows or Mac boxes, I've also been using JetBrains DataGrip (jetbrains.com/datagrip), a cross-platform database tool that supports a slew of databases.

But as the universe of extensions for VS Code has grown (to just short of 3,000 as I'm writing this in late April 2017), a number of extensions for interacting with data stores have come to the table. Two that I've already worked with are the mssql extension (bit.ly/2gb2ICf) from the SQL Server team, and the vscode-database extension for SQLite and PostgreSQL (bit.ly/2mh8nYF). These allow you to write and execute SQL against your databases. You can see demos of both in my Pluralsight course, "Entity Framework Core: Getting Started." There are other data-related extensions, as well—for example, to interact with Azure Data Lake (also from Microsoft), Redis and SalesForce data. Some of these are still in preview, including the mssql extension.

In this article, I'm going to show you some of the basics of using the mssql extension. Originally I planned to write about both mssql and vscode-database, but mssql is so rich in features, I had a lot of fun exploring it and easily filled up my allotment of words for this column. It makes no difference whether you're on Windows, macOS or Linux to use mssql.

The mssql extension lets you interact with a variety of SQL Server databases: Microsoft SQL Server, Azure SQL Database and SQL Data Warehouse. As I happen to be sitting in front of a MacBook, I'll connect to an external SQL Server. You may be aware that SQL Server now runs on Linux—is that not amazing? That means you could spin up a Docker container that runs SQL Server. I won't do that for this article, but I've written a blog post about that, which you can read at bit.ly/2qaev9r. What I will do, however, is connect to an Azure SQL Server database in the cloud. This is a powerful reminder that you don't need to be a Windows developer or admin or a C# developer to take advantage of SQL Server. You can

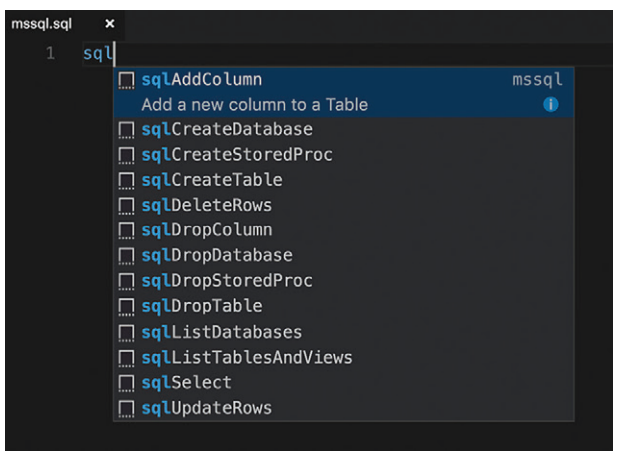


Figure 2 **TSQL Snippets Provided by the mssql extension**

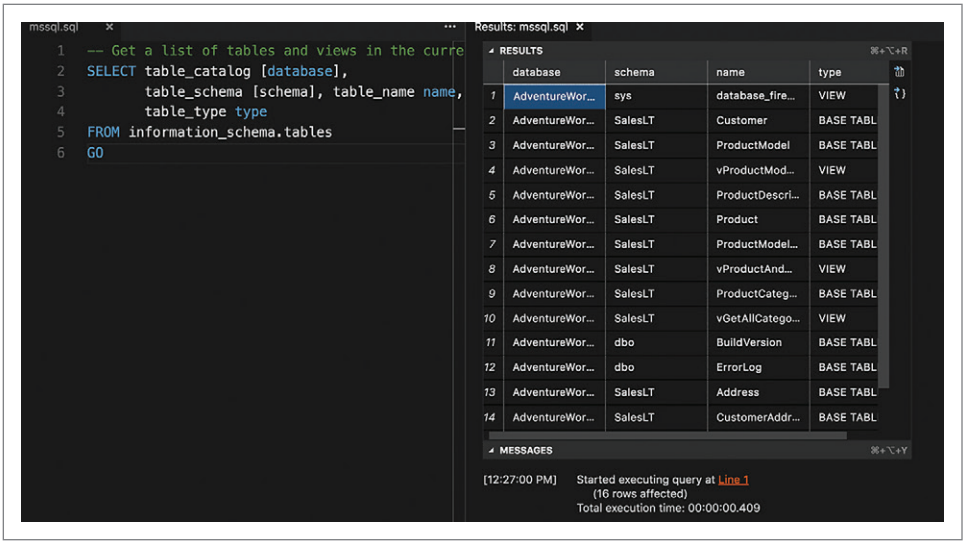


Figure 3 **Results of a Schema Query**

manage it on the Azure portal and connect to it from any type of app. So, imagine your app is PHP and you're using the PHP extension for VS Code and you're talking to a SQL Server database on Azure.

I took advantage of the free Azure account and credits that come with a Visual Studio subscription to create an Azure SQL database based on the AdventureWorks sample. The documentation to do this is at bit.ly/2o2IDTy.

Back in VS Code, you can install the mssql extension by clicking on the extension icon in the VS Code Activity Bar, filtering on mssql and then clicking its Install button. You'll be prompted to reload VS Code when it's ready, but keep in mind that it will delay installing part of the extension until you use one of its commands. Note that for macOS, you'll need to install OpenSSL. Check the mssql documentation link (provided earlier) for details.

In addition to the query execution engine that's installed, mssql places a slew of commands in the VS Code command palette. It makes sense to start by connecting to your database, although other functions will prompt you to connect if you haven't already.

Open the palette with F1 (or Ctrl or Command+Shift+P if you have one of those funny keyboards without function keys) and type MS SQL to filter on all of the mssql commands. If you don't have any other extensions that provide commands with the SQL keyword, just SQL will do the trick. I love that you can even get to help files with the Getting Started Guide command. You can also store different connections and then connect easily with the Manage Connection Profiles feature.

The easiest way to interact with mssql is by having a file open for editing and ensuring that VS Code knows you're editing SQL. You can do this whether you have a folder or project open thanks to the New query command. That will create a SQLQuery.sql file and the sql extension will cause VS Code to switch to the mssql editor. The current editor is noted in the lower right-hand corner of VS Code, and this changes based on file extensions to give you proper IntelliSense and other relevant features provided by the extension. You can click on what's displayed to change it if needed. The mssql editor will not only give you help writing TSQL, it also knows how to execute queries and perform other tasks defined by the extension.

With a SQL file open in the editor, selecting MS SQL: Connect from the Command Palette will display a list of existing connection profiles you've already created and let you create a new one. Select Create Connection Profile and you'll then be prompted for each key element of the connection string. For example, the first prompt asks for the Server name, as shown in **Figure 1**. My SQL Azure database is on a server named thedatafarmsqlerver.database.windows.net, so that's what I'll enter.

Next, you'll be prompted to enter the database name, your login and password, then an optional profile name. By the way, Manage Connection Profiles can also lead you to this point as it has a Create menu option.

Once you've filled out the connection information and are successfully connected, the profile will get saved in the VS Code settings file. With version 0.3, if the connection fails, the profile won't get stored; but that experience is on deck to change. You can look at the stored profile by going to Preferences and Settings from the VS Code menu or via the Ctrl or Command+, (comma) keystroke combo. Here's an example of a connection profile:

```
"mssql.connections": [
  {
    "authenticationType": "SqlLogin",
    "server": "thedatafarmsqlserver.database.windows.net",
    "database": "AdventureWorksSample",
    "user": "me",
    "password": "mypassword",
    "savePassword": true,
    "profileName": "AzureAWSample"}
  }
]
```

With profiles stored in the settings, you have the option to select AzureAWSample or other stored profiles when you want to connect.

Once connected, you can start writing and executing TSQL. The easiest way to do that is to have a file with the SQL extension open in the editor. As I mentioned, this forces the SQL editor features to kick in, and one of its wonderful features is the built-in TSQL snippets.

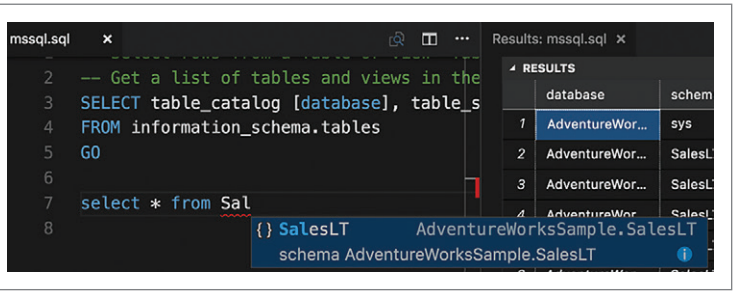


Figure 4 **IntelliSense Reads the Schema and Helps Build TSQL**

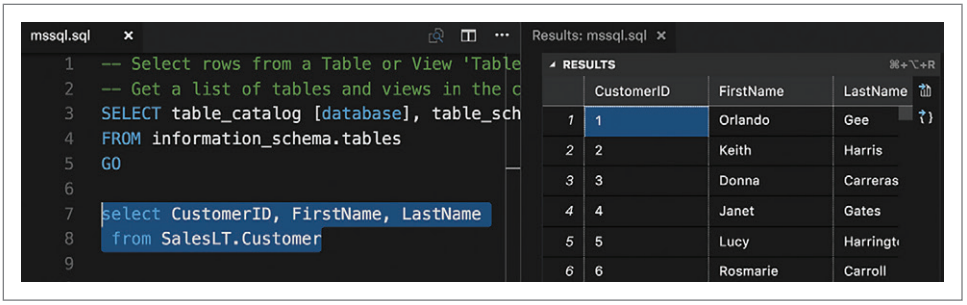


Figure 5 **Results of a Customer Data Query**

Begin by typing sql in the editor window and IntelliSense will display the list of built-in snippets, as shown in **Figure 2**.

You can see that this isn't limited to simply querying existing data—mssql will execute any (valid and permissible) TSQL. I like to list the databases or tables and views from a selected database to verify that I've connected to the correct database, and the ListTablesAndViews snippet is great for that. Normally, I'd have to ask Dr. GoogleBing to help me with TSQL like this, so I'm extra grateful for this snippet.

Pressing Ctrl or Command+Shift+E will execute the command. You can also select the text and right-click for the context menu, which includes an Execute option.

A results pane opens up with the response like the one in **Figure 3**.

In the upper right, notice there are two small icons. The first lets you save the results as a CSV text file. The second is for saving the results as JSON. Let's do a query that gets Customer data, see what those results look like, then save them to JSON. With the list of tables in front of me, I can start typing my query and, as **Figure 4** shows, IntelliSense kicks in with knowledge of the database schema.

I modified my select statement to get three columns from SalesLT.Customer. Then, I was able to highlight the statement and use the Ctrl or Command+Shift+E keyboard shortcut again to execute just that query. As with SQL Server Management Studio, I can execute one or more statements and, in response, see one or more result sets.

The results are presented in a grid as shown in **Figure 5**. You can highlight one or more row and column combinations and then click the CSV or JSON icons, which will then prompt you for a file name to save to. And you can easily select all of the data from a context menu by right-clicking on the grid.

The filename prompt displays the current folder path. If that's where you want to save the file, you don't have to retype the path. Just type the name of the file and it will be saved in that folder. I didn't realize that the first few times, so learn from my mistake.

I selected only the first row of the customer data my query projected, and then used the Save to JSON icon, specifying a file name. Here's the JSON that was output to my file:

```
[
  {
    "CustomerID": "1",
    "FirstName": "Orlando",
    "LastName": "Gee"
  }
]
```

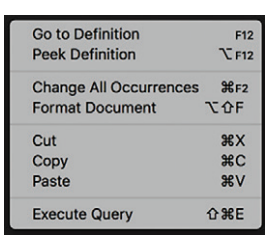


Figure 6 **Editor Window Context Menu**

Keep in mind that you can easily add your own snippets to VS Code. Let's create one to list stored procedures and functions. Start by going to Preferences and User Snippets. You'll see that the mssql extension added a template for SQL snippets. Choose that and the blank template will open. As you add more snippets, you'll continue to work in this file. Also, if you've created some snippets to share (or perhaps find a bug or have another idea for mssql), this is an open source extension, and you can contribute by going to github.com/Microsoft/vscode-mssql to submit pull requests or issues.

After a long session with Dr. GoogleBing and testing various ideas, here's the snippet I created to list all of the stored procedures and functions in the target database:

```
"List Stored Procedures": {
  "prefix": "sqlListStoredProcedures",
  "body": [
    "SELECT [Name],[Type_Desc] " ,
    "FROM [sys].[all_objects] " ,
    "WHERE ([Type] = 'P' OR [Type]='FN' OR [Type]='TF' OR [Type]='IF') ",
    "AND [Is_MS_Shipped] = 0"
  ],
  "description": "List Stored Procedures"
}
```

Now when I type sql in the editor window, sqlListStoreProcedures is among the options. The results of executing that command against my target database are:

```
Name                          Type_Desc
uspPrintError                 SQL_STORED_PROCEDURE
uspLogError                   SQL_STORED_PROCEDURE
ufnGetAllCategories           SQL_TABLE_VALUED_FUNCTION
ufnGetSalesOrderStatusText    SQL_SCALAR_FUNCTION
ufnGetCustomerInformation     SQL_INLINE_TABLE_VALUED_FUNCTION
```

I was able to share this output by right-clicking on the results grid and selecting its "Copy with headers" option.

As the late-night television ads say, "But wait! There's more!" The editor window has a context menu, as well (see **Figure 6**).

The most interesting items on it (in my humble opinion) are Go to Definition and Peek Definition. If you select a table name in the edit window—for example, Customer in the command shown in **Figure 5**—these commands will show you the CREATE script for the customer table.

The mssql extension is constantly evolving and I'm looking forward to future updates. The version I've shared here is still a preview, version 0.3.0. If you have it installed in Visual Studio Code, you'll be notified of updates. You can watch and participate in its evolution on its GitHub site at aka.ms/mssql-marketplace. ■

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

Thanks to the following Microsoft technical experts for reviewing this article: *Kevin Cunnane, Eric Kang and Sanjay Nagamangalam*

DevExpress®

DevExpress Spreadsheet for WPF & WinForms

with Chart, Pivot Table and Cell Editor support

Your users already know Microsoft® Excel® and with the new capabilities we've introduced in our Spreadsheet Control, you can create solutions that utilize a "spreadsheet-first" UX for a broad range of use-case scenarios. Imagine sales entry forms that are an actual invoice or an inventory management document with built-in ordering and data entry options.

Give our Spreadsheet a try today and see how easy it is to build Windows® apps that amaze.

Free 30-day trial
devexpress.com/spreadsheet

#UseTheBest

Use Azure IoT Suite to Boost IoT Development

Dawid Borycki

An Internet of Things (IoT) solution comprises remote telemetry devices, a Web portal, cloud storage and real-time processing. Such a complex structure can make you reluctant to start IoT development. To make things easier, the Microsoft Azure IoT Suite provides two preconfigured solutions: remote monitoring and predictive maintenance. Here, I'll tell you how to create a remote monitoring solution, which will collect and analyze data from a remote IoT device controlled by Windows 10 IoT Core. This device, the Raspberry Pi, will acquire images from a USB camera. Subsequently, the image brightness will be calculated on the IoT device and then streamed to the cloud, where it will be stored, processed and displayed (see **Figure 1**). Furthermore, the end user will not only be able to see the information acquired with the remote device, but also remotely control that device. You can find the complete source code supporting this discussion at msdn.com/magazine/0617magcode.

This article discusses:

- Creating the RemoteCamera UWP app
- Provisioning the remote monitoring solution and registering a device
- Device metadata and communicating with the cloud
- Handling remote commands and sending commands from the cloud

Technologies discussed:

Azure IoT Suite, Universal Windows Platform, Raspberry Pi

Code download available at:

msdn.com/magazine/0617magcode

Remote Device

The basics of programming the Raspberry Pi with Windows 10 IoT Core was already discussed in this magazine by Frank LaVigne (msdn.com/magazine/mt694090) and Bruno Sonnino (msdn.com/magazine/mt808503). LaVigne and Sonnino showed how to set up the development environment and the IoT board, how to configure the IoT unit in your browser using the Device Portal, and how to control GPIO ports with Windows 10 IoT Core. Moreover, LaVigne mentioned in his article that IoT can be used to program and control remote cameras. Here, I develop this thought and explicitly show how to turn the Raspberry Pi into such a device.

To that end, I create the RemoteCamera Universal Windows Platform (UWP) app using the Blank App (Universal Windows) Visual C# project template, and then set the target and minimum API versions to Windows 10 Anniversary Edition (10.0; Build 14393). I use this API version to enable binding to methods, which allows me to directly wire methods of the view model with events fired by visual controls:

```
<Button x:Name="ButtonPreviewStart"
        Content="Start preview"
        Click="{x:Bind remoteCameraViewModel.PreviewStart}" />
```

Next, I declare the UI as shown in **Figure 1**. There are two tabs: Camera capture and Cloud. The first contains controls for starting

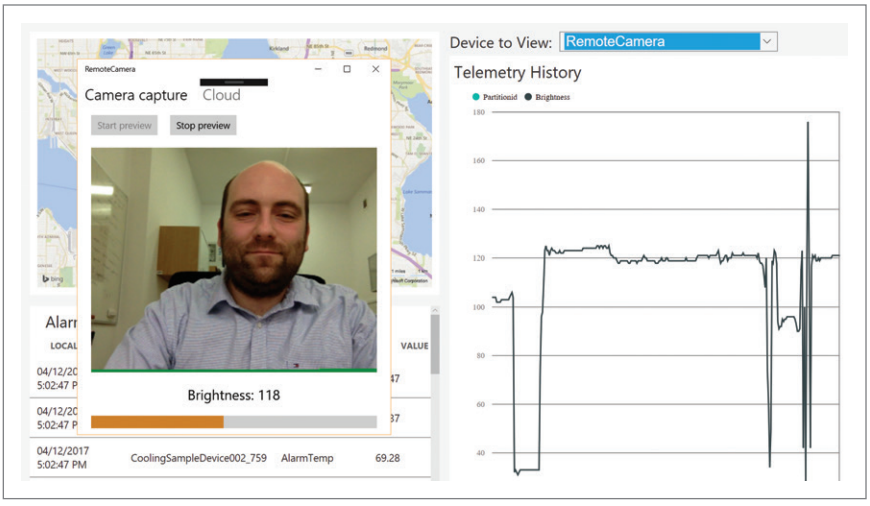


Figure 1 **A Preconfigured Azure IoT Suite Solution Portal and the Remote Universal Windows Platform App, Which Acquires and Processes the Video Stream**

and stopping camera preview, displaying the video stream and presenting image brightness (a label and a progress bar). The second tab contains two buttons, which you can use to connect a device to the cloud and to register a device with the IoT portal. The Cloud tab also has a checkbox that lets you enable streaming telemetry data.

> Note that the RemoteCamera app is universal, so it can be deployed without any changes to your development PC, a smartphone, a tablet or to the Raspberry Pi.

Most of the logic associated with the UI is implemented within the RemoteCameraViewModel class (see the ViewModels subfolder of the RemoteCamera project). This class, apart from implementing a few properties that are bound to the UI, handles the video acquisition, image processing and cloud interaction. Those sub-functionalities are implemented in separate classes: CameraCapture, ImageProcessor and CloudHelper, respectively. I'll discuss CameraCapture and ImageProcessor shortly, while CloudHelper and related helper classes will be described later in the context of the preconfigured Azure IoT solution.

Camera Capture

Camera capture (see CameraCapture.cs under the Helpers folder) is built on top of two elements: Windows.Media.Capture.MediaCapture class and Windows.UI.Xaml.Controls.CaptureElement. The former is used to acquire video, while the latter displays the acquired video stream. I acquire video with a webcam, so I have to declare the corresponding device capability in the Package.appxmanifest.

To initialize the MediaCapture class, you invoke the InitializeAsync method. Eventually, you can pass to that method an instance of the MediaCaptureInitializationSettings class, which lets you specify capture options. You can choose between streaming video or audio and select the capture hardware. Here, I acquire only video from the default webcam (see **Figure 2**).

Next, using the Source property of the CaptureElement class instance, I wire this object with the MediaCapture control to display video stream. I also invoke helper method GetVideoProperties, which reads and then stores the size of a video frame. I use this information later to get the preview frame for processing. Finally, to actually start and stop video acquisition, I invoke StartPreviewAsync and StopPreviewAsync of the MediaCapture class. In the CameraCapture I wrapped those methods with additional logic, verifying initialization and the preview state:

```
public async Task Start()
{
  if (IsInitialized)
  {
    if (!IsPreviewActive)
    {
      await MediaCapture.StartPreviewAsync();

      IsPreviewActive = true;
    }
  }
}
```

When you run the app, you can press the Start preview button, which configures camera acquisition, and you'll see the camera

Figure 2 **Camera Capture Initialization**

```
public MediaCapture { get; private set; } = new MediaCapture();
public bool IsInitialized { get; private set; } = false;

public async Task Initialize(CaptureElement captureElement)
{
  if (!IsInitialized)
  {
    var settings = new MediaCaptureInitializationSettings()
    {
      StreamingCaptureMode = StreamingCaptureMode.Video
    };

    try
    {
      await MediaCapture.InitializeAsync(settings);

      GetVideoProperties();

      captureElement.Source = MediaCapture;
      IsInitialized = true;
    }
    catch (Exception ex)
    {
      Debug.WriteLine(ex.Message);

      IsInitialized = false;
    }
  }
}
```

image after a short while. Note that the RemoteCamera app is universal, so it can be deployed without any changes to your development PC, a smartphone, a tablet or to the Raspberry Pi. If you test the RemoteCamera app with your Windows 10 PC, you need to ensure that apps are allowed to use your camera. You configure this setting with the Settings app (Privacy/Camera). To test the app with Raspberry Pi I use a low-budget Microsoft Life Cam HD-3000. This is a USB webcam, so it's automatically detected by Windows 10 IoT Core after I connect it to one of the four Raspberry Pi USB ports. You'll find a full list of cameras compatible with Windows 10 IoT Core at bit.ly/2p1ZHGD. Once you connect your webcam to the Rasbperry Pi, it will appear under the Devices tab of the Device Portal.

Image Processor

The ImageProcessor class calculates the brightness of the current frame in the background. To perform the background operation, I create a thread with the task-based asynchronous pattern, as shown in **Figure 3**.

Within the while loop, I determine image brightness, and then pass this value to listeners of the ProcessingDone event. This event is fed with an instance of the ImageProcessorEventArgs class, which has only one public property, Brightness. The processing runs until the task receives a cancellation signal. The key element of the image processing is the GetBrightness method, shown in **Figure 4**.

Figure 3 **Calculating Brightness in the Background**

```
public event EventHandler<ImageProcessorEventArgs> ProcessingDone;

private void InitializeProcessingTask()
{
  processingCancellationTokenSource = new CancellationTokenSource();

  processingTask = new Task(async () =>
  {
    while (!processingCancellationTokenSource.IsCancellationRequested)
    {
      if (IsActive)
      {
        var brightness = await GetBrightness();

        ProcessingDone(this, new ImageProcessorEventArgs(brightness));

        Task.Delay(delay).Wait();
      }
    }
  }, processingCancellationTokenSource.Token);
}
```

Figure 4 **The GetBrightness Method**

```
private async Task<byte> GetBrightness()
{
  var brightness = new byte();

  if (cameraCapture.IsPreviewActive)
  {
    // Get current preview bitmap
    var previewBitmap = await cameraCapture.GetPreviewBitmap();

    // Get underlying pixel data
    var pixelBuffer = GetPixelBuffer(previewBitmap);

    // Process buffer to determine mean gray value (brightness)
    brightness = CalculateMeanGrayValue(pixelBuffer);
  }

  return brightness;
}
```

I access the preview frame using GetPreviewBitmap of the CameraCapture class instance. Internally, GetPreviewBitmap uses GetPreviewFrameAsync of the MediaCapture class. There are two versions of GetPreviewFrameAsync. The first, a parameterless method, returns an instance of the VideoFrame class. In this case, you can get the actual pixel data by reading the Direct3DSurface property. The second version accepts an instance of the VideoFrame class and copies pixel data to its SoftwareBitmap property. Here, I use the second option (see the GetPreviewBitmap method of the CameraCapture class) and then access pixel data through the CopyToBuffer method of the SoftwareBitmap class instance shown in **Figure 5**.

> I first verify that the pixel format is BGRA8, which means the image is represented with four 8-bit channels: three for the blue, green, and red colors, and one for alpha or transparency.

I first verify that the pixel format is BGRA8, which means the image is represented with four 8-bit channels: three for the blue, green, and red colors, and one for alpha or transparency. If the input bitmap has a different pixel format, I perform an appropriate conversion. Next, I copy pixel data to the byte array, whose size is determined by image height multiplied by image stride (bit.ly/2om8Ny9). I read both values from an instance of the BitmapPlaneDescription, which I obtain from the BitmapBuffer object, returned by SoftwareBitmap.LockBuffer method.

Given the byte array with pixel data, all I need to do is calculate the mean value of all the pixels. So, I iterate over the pixel buffer (see **Figure 6**).

Figure 5 **Accessing Pixel Data**

```
private byte[] GetPixelBuffer(SoftwareBitmap softwareBitmap)
{
  // Ensure bitmap pixel format is Bgra8
  if (softwareBitmap.BitmapPixelFormat != CameraCapture.BitmapPixelFormat)
  {
    SoftwareBitmap.Convert(softwareBitmap, CameraCapture.BitmapPixelFormat);
  }

  // Lock underlying bitmap buffer
  var bitmapBuffer = softwareBitmap.LockBuffer(BitmapBufferAccessMode.Read);

  // Use plane description to determine bitmap height
  // and stride (the actual buffer width)
  var planeDescription = bitmapBuffer.GetPlaneDescription(0);
  var pixelBuffer = new byte[planeDescription.Height * planeDescription.Stride];

  // Copy pixel data to a buffer
  softwareBitmap.CopyToBuffer(pixelBuffer.AsBuffer());

  return pixelBuffer;
}
```

Then, at each iteration, I convert a given pixel to the gray scale by averaging values from each color channel:

```
private static byte GetGrayscaleValue(byte[] pixelBuffer, uint startIndex)
{
  var grayValue = (pixelBuffer[startIndex]
    + pixelBuffer[startIndex + 1]
    + pixelBuffer[startIndex + 2]) / 3.0;

  return Convert.ToByte(grayValue);
}
```

When you provision the remote monitoring solution, the Azure IoT Suite portal creates several Azure resources: IoT Hub, Stream Analytics jobs, Storage and the App Service.

Brightness is passed to the view through the ProcessingDone event. This event is handled in the MainPage class (MainPage.xaml.cs), where I display brightness in the label and a progress bar. Both controls are bounded to the Brightness property of the RemoteCameraViewModel. Note that ProcessingDone is raised from the background thread. Hence, I modify the RemoteCameraViewModel.Brightness through the UI thread using Dispatcher class, as shown in **Figure 7**.

Figure 6 **Calculating the Mean Value of the Pixels**

```
private byte CalculateMeanGrayValue(byte[] pixelBuffer)
{
  // Loop index increases by four since
  // there are four channels (blue, green, red and alpha).
  // Alpha is ignored for brightness calculation
  const int step = 4;
  double mean = 0.0;

  for (uint i = 0; i < pixelBuffer.Length; i += step)
  {
    mean += GetGrayscaleValue(pixelBuffer, i);
  }

  mean /= (pixelBuffer.Length / step);

  return Convert.ToByte(mean);
}
```

Figure 7 **Modifying the RemoteCameraViewModel.Brightness Through the UI Thread Using the Dispatcher Class**

```
private async void DisplayBrightness(byte brightness)
{
  if (Dispatcher.HasThreadAccess)
  {
    remoteCameraViewModel.Brightness = brightness;
  }
  else
  {
    await Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
    {
      DisplayBrightness(brightness);
    });
  }
}
```

Provisioning the Remote Monitoring Solution

To provision a solution you can use a dedicated portal at azureiotsuite.com. After logging in and choosing your Azure subscription, you'll be redirected to a page where you press the "Create a new solution" rectangle. This will open a Web site where you can choose one of two preconfigured solutions: predictive maintenance or remote monitoring (see **Figure 8**). After choosing the solution another form appears, which lets you set the solution name and region for Azure resources. Here, I set the solution name and region to RemoteCameraMonitoring and West US, respectively.

When you provision the remote monitoring solution, the Azure IoT Suite portal creates several Azure resources: IoT Hub, Stream Analytics jobs, Storage and the App Service. The IoT Hub provides bidirectional communication between the cloud and remote devices. Data streamed by remote devices is transformed by a Stream Analytics job, which typically filters out unnecessary data. Filtered data is stored or directed for further analysis. Finally, the App Service is used to host the Web portal.

Solution provisioning can also be done from the command line. To do so, you can clone or download the solution source code from bit.ly/2osI4RW, and then follow the instruction at bit.ly/2p7MPPc. You also have the option to deploy a solution locally, as shown at

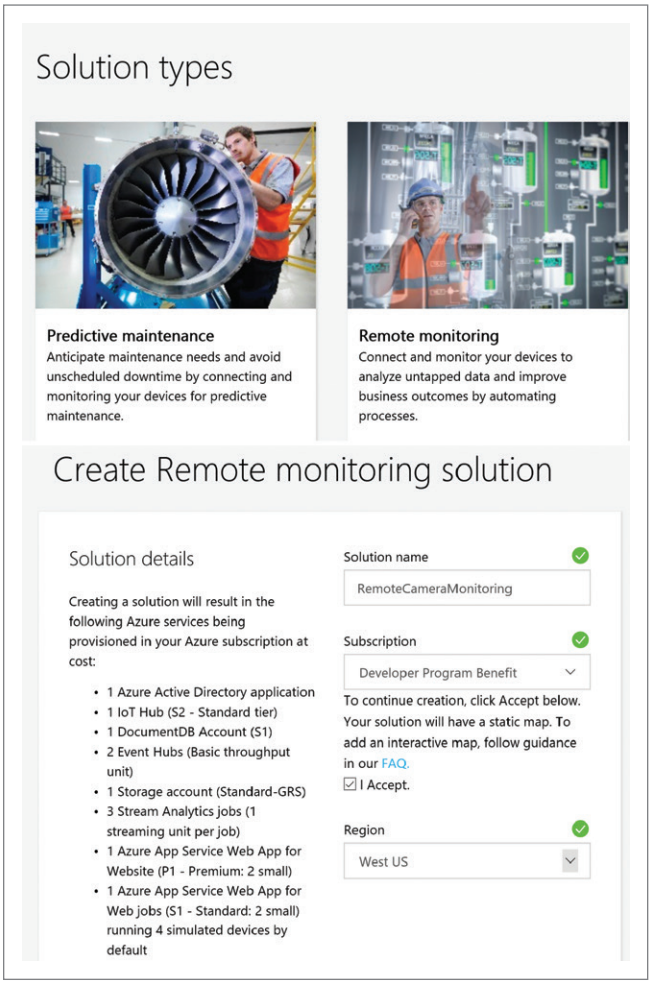


Figure 8 **Azure IoT Suite Preconfigured Solutions (Top) and Remote Monitoring Solution Configuration (Bottom)**

bit.ly/2nEePNi. In this case, no Azure App Service is created because the solution portal runs on a local machine. Such an approach can be particularly useful for development and debugging or when you want to modify a preconfigured solution.

Once the provisioning is finished, you can launch the solution, and its portal will appear in the default browser (refer back to **Figure 1**). This portal has a few tabs. For the purpose of this article, I'll limit my attention to just two of them: dashboard and devices. Dashboard displays the map with remote devices and the telemetry data they stream. The Devices tab shows a list of remote devices, including their status, capabilities and a description. By default, there are several emulated devices. Let me explain how to register the new, non-emulated hardware.

Registering a Device

To register a device you use the solution portal, where you press the Add a device hyperlink located in the bottom-left corner. You can then choose between a simulated or a custom device. Pick the second option and press the Add New button. Now you can define the Device ID. I set this value to RemoteCamera. Afterward, the ADD A CUSTOM DEVICE form displays device credentials (see **Figure 9**), which I'll use later to connect my IoT device to the IoT Hub.

Device Metadata and Communicating with the Cloud

The device you add will appear in the devices list, and you can then send device metadata or device information. Device information comprises a JSON object that describes the remote device. This object tells the cloud endpoint the capabilities of your device, and includes a hardware description and the list of remote commands accepted by your device. These commands can be sent by the end user to your device through the IoT solution portal. In the RemoteCamera app, device info is represented as the DeviceInfo class (in the AzureHelpers subfolder):

```
public class DeviceInfo
{
  public bool IsSimulatedDevice { get; set; }

  public string Version { get; set; }

  public string ObjectType { get; set; }

  public DeviceProperties DeviceProperties { get; set; }

  public Command[] Commands { get; set; }
}
```

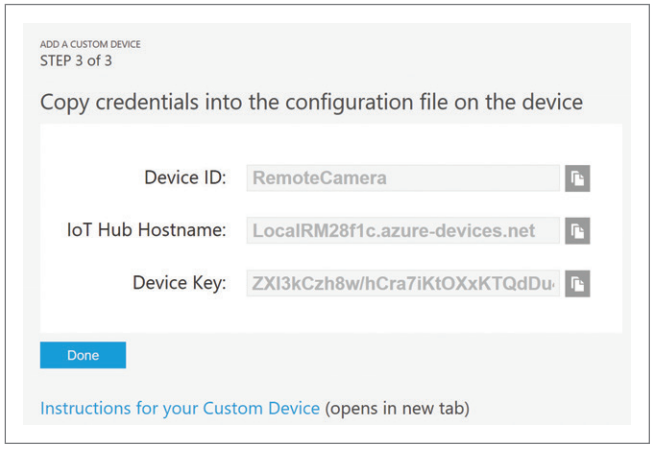


Figure 9 **Summary of Device Registration**

The first two properties of DeviceInfo specify whether a device is simulated and define the version of the DeviceInfo object. As you'll see later, the third property, ObjectType, is set to a string constant, DeviceInfo. This string is used by the cloud, specifically by the Azure Stream Analytics job, to filter device info from telemetry data. Next, DeviceProperties (see the AzureHelpers subfolder) holds a collection of properties (like serial number, memory, platform, RAM) that describe your device. Finally, the Commands property is a collection of remote commands recognized by your device. You define each command by specifying its name and a list of parameters, which are represented by the Command and CommandParameter classes, respectively (see AzureHelpers\Command.cs).

> The Devices tab shows a list of remote devices, including their status, capabilities and a description.

To establish communication between your IoT device and the IoT Hub you use the Microsoft.Azure.Devices.Client NuGet package. This package provides the DeviceClient class, which you use to send and receive messages to and from the cloud. You can create an instance of the DeviceClient using either the Create or CreateFromConnectionString static methods. Here, I use the first option (CloudHelper.cs in the AzureHelpers folder):

```
public async Task Initialize()
{
  if (!IsInitialized)
  {
    deviceClient = DeviceClient.Create(
      Configuration.Hostname, Configuration.AuthenticationKey());

    await deviceClient.OpenAsync();

    IsInitialized = true;

    BeginRemoteCommandHandling();
  }
}
```

As you can see, the DeviceClient.Create method requires you to provide the hostname of the IoT Hub and device credentials (the identifier and the key). You obtain these values from the solution portal during device provisioning (refer back to **Figure 9**). In the RemoteCamera app, I store the hostname, device id and the key within the Configuration static class:

```
public static class Configuration
{
  public static string Hostname { get; } = "<iot-hub-name>.azure-devices.net";

  public static string DeviceId { get; } = "RemoteCamera";

  public static string DeviceKey { get; } = "<your_key>";

  public static DeviceAuthenticationWithRegistrySymmetricKey AuthenticationKey()
  {
    return new DeviceAuthenticationWithRegistrySymmetricKey(DeviceId, DeviceKey);
  }
}
```

Additionally, the Configuration class implements a static method AuthenticationKey, which wraps device credentials into an instance

Figure 10 **Sending Device Information**

```
public async Task SendDeviceInfo()
{
  var deviceInfo = new DeviceInfo()
  {
    IsSimulatedDevice = false,
    ObjectType = "DeviceInfo",
    Version = "1.0",
    DeviceProperties = new DeviceProperties(Configuration.DeviceId),

    // Commands collection
    Commands = new Command[]
    {
      CommandHelper.CreateCameraPreviewStatusCommand()
    }
  };

  await SendMessage(deviceInfo);
}
```

of the DeviceAuthenticationWithRegistrySymmetricKey class. I use this to simplify creation of the DeviceClient class instance.

Once the connection is made, all you need to do is to send the DeviceInfo, as shown in **Figure 10**.

The RemoteCamera app sends the device info, which describes the real hardware, so IsSimulatedDevice property is set to false. As I mentioned, ObjectType is set to DeviceInfo. Moreover, I set the Version property to 1.0. For DeviceProperties I use arbitrary values, which mostly consist of static strings (see the SetDefaultValues method of the DeviceProperties class). I also define one remote command, Update camera preview, which enables remote control of the camera preview. This command has one Boolean parameter, IsPreviewActive, which specifies whether the camera preview should be started or stopped (see the CommandHelper.cs file under AzureHelpers folder).

To establish communication between your IoT device and the IoT Hub you use the Microsoft.Azure.Devices.Client NuGet package.

To actually send data to the cloud I implement the SendMessage method:

```
private async Task SendMessage(Object message)
{
  var serializedMessage = MessageHelper.Serialize(message);

  await deviceClient.SendEventAsync(serializedMessage);
}
```

Basically, you need to serialize your C# object to a byte array that contains JSON-formatted objects (see the MessageHelper static class from AzureHelpers subfolder):

```
public static Message Serialize(object obj)
{
  ArgumentCheck.IsNull(obj, "obj");

  var jsonData = JsonConvert.SerializeObject(obj);

  return new Message(Encoding.UTF8.GetBytes(jsonData));
}
```

Then you wrap the resulting array into the Message class, which you send to the cloud with the SendEventAsync method of the DeviceClient class instance. The Message class is the object, which supplements raw data (a JSON object being transferred) with additional properties. These properties are used to track messages being sent between devices and the IoT Hub.

> Once you connect to the cloud, you can also start sending telemetry data, much as you did when sending device information.

In the RemoteCamera app, establishing a connection with the cloud and sending device info is triggered by two buttons located on the Cloud tab: Connect and initialize and Send device info. The click event handler of the first button is bound to the Connect method of the RemoteCameraViewModel:

```
public async Task Connect()
{
  await CloudHelper.Initialize();
  IsConnected = true;
}
```

The click event handler of the second button is wired with the SendDeviceInfo method of the CloudHelper class instance. This method was discussed earlier.

Once you connect to the cloud, you can also start sending telemetry data, much as you did when sending device information.

Figure 11 **Remote Messages Deserialization and Parsing**

```
private async Task HandleIncomingMessage(Message message)
{
  try
  {
    // Deserialize message to remote command
    var remoteCommand = MessageHelper.Deserialize(message);

    // Parse command
    ParseCommand(remoteCommand);

    // Send confirmation to the cloud
    await deviceClient.CompleteAsync(message);
  }
  catch (Exception ex)
  {
    Debug.WriteLine(ex.Message);

    // Reject message, if it was not parsed correctly
    await deviceClient.RejectAsync(message);
  }
}

private void ParseCommand(RemoteCommand remoteCommand)
{
  // Verify remote command name
  if (string.Compare(remoteCommand.Name,
    CommandHelper.CameraPreviewCommandName) == 0)
  {
    // Raise an event, when the valid command was received
    UpdateCameraPreviewCommandReceived(this,
      new UpdateCameraPreviewCommandEventArgs(
      remoteCommand.Parameters.IsPreviewActive));
  }
}
```

That is, you can use the SendMessage method to which you pass a telemetry object. Here, this object, an instance of the TelemetryData class, has just a single property, Brightness. The following shows a complete example of sending telemetry data to the cloud, implemented within the SendBrightness method of the CloudHelper class:

```
public async void SendBrightness(byte brightness)
{
  if (IsInitialized)
  {
    // Construct TelemetryData
    var telemetryData = new TelemetryData()
    {
      Brightness = brightness
    };

    // Serialize TelemetryData and send it to the cloud
    await SendMessage(telemetryData);
  }
}
```

SendBrightness is invoked right after obtaining the brightness, calculated by the ImageProcessor. This is performed in the ProcessingDone event handler:

```
private void ImageProcessor_ProcessingDone(object sender, ImageProcessorEventArgs e)
{
  // Update display through dispatcher
  DisplayBrightness(e.Brightness);

  // Send telemetry
  if (remoteCameraViewModel.IsTelemetryActive)
  {
    remoteCameraViewModel.CloudHelper.SendBrightness(e.Brightness);
  }
}
```

So, if you now run the RemoteCamera app, start the preview and connect to the cloud, you'll see that brightness values are displayed in the corresponding chart, as shown back in **Figure 1**. Note that, although simulated devices of the preconfigured solution are dedicated to sending temperature and humidity as telemetry data, you can also send other values. Here, I'm sending brightness, which is automatically displayed in the appropriate chart.

Handling Remote Commands

The CloudHelper class also implements methods for handling remote commands received from the cloud. Similarly, as with the case of ImageProcessor, I handle commands in the background

Figure 12 **Updating the Camera Preview**

```
private async void CloudHelper_UpdateCameraPreviewCommandReceived(
  object sender, UpdateCameraPreviewCommandEventArgs e)
{
  if (Dispatcher.HasThreadAccess)
  {
    if (e.IsPreviewActive)
    {
      await remoteCameraViewModel.PreviewStart();
    }
    else
    {
      await remoteCameraViewModel.PreviewStop();
    }
  }
  else
  {
    await Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
    {
      CloudHelper_UpdateCameraPreviewCommandReceived(sender, e);
    });
  }
}
```

Data Quality Made Easy.
Your Data, Your Way.

Melissa provides the full spectrum of data quality to ensure you have data you can trust.

We profile, standardize, verify, match and enrich global **People Data** – name, address, email, phone, and more.

Our data quality solutions are available on-premises and in the Cloud – fast, easy to use, and powerful developer tools and plugins for the **Microsoft® Product Ecosystem**.

Microsoft® SQL Server® Integration Services

Microsoft Dynamics® CRM

Microsoft® .NET

Microsoft Azure

Microsoft® Excel

Start Your Free Trial

www.Melissa.com/msft-pd

Melissa Data is Now Melissa.
Why the change?
See for Yourself at the New www.Melissa.com

melissa™
1-800-MELISSA

(BeginRemoteCommandHandling of the CloudHelper class). Again, I use the task-based asynchronous pattern:

```
private void BeginRemoteCommandHandling()
{
  Task.Run(async () =>
  {
    while (true)
    {
      var message = await deviceClient.ReceiveAsync();

      if (message != null)
      {
        await HandleIncomingMessage(message);
      }
    }
  });
}
```

This method is responsible for creating a task that continuously analyzes messages received from the cloud endpoint. To receive a remote message, you invoke the ReceiveAsync method of the DeviceClient class. ReceiveAsync returns an instance of the Message class, which you use to obtain a raw byte array containing JSON-formatted remote command data. You then deserialize this array to the RemoteCommand object (see RemoteCommand.cs in the AzureHelpers folder), which is implemented in the MessageHelper class (the AzureHelpers subfolder):

```
public static RemoteCommand Deserialize(Message message)
{
  ArgumentCheck.IsNull(message, "message");

  var jsonData = Encoding.UTF8.GetString(message.GetBytes());

  return JsonConvert.DeserializeObject<RemoteCommand>(
    jsonData);
}
```

RemoteCommand has several properties, but you typically use just two: name and parameters, which contain the command name and command parameters. In the RemoteCamera app, I use these values to determine if an expected command was received (see **Figure 11**). If so, I raise the UpdateCameraPreviewCommandReceived event to pass that information to listeners, then I inform the cloud that the command was accepted using the CompleteAsync method of the DeviceClient class. If the command isn't recognized, I reject it using the RejectAsync method.

The UpdateCameraPreviewCommandReceived event is handled in the MainPage class. Depending on the parameter value I obtain from the remote command, I either stop or start the local camera preview. This operation is again dispatched to the UI thread, as shown in **Figure 12**.

Sending Commands from the Cloud

Finally, I show that the solution portal can be used to remotely control your IoT device. For this, you use the Devices tab, where you need to find and then click your device (see **Figure 13**).

This activates a device details pane, where you click the Commands hyperlink. You'll then see another form that allows you to choose and send a remote command. The particular layout of this form depends on the command you choose. Here, I have only one single-parameter command, so there's only one checkbox. If you clear this checkbox, and send the command, RemoteCamera app will stop the preview. All commands you send, along with their status, appear in the command history, as shown in **Figure 14**.

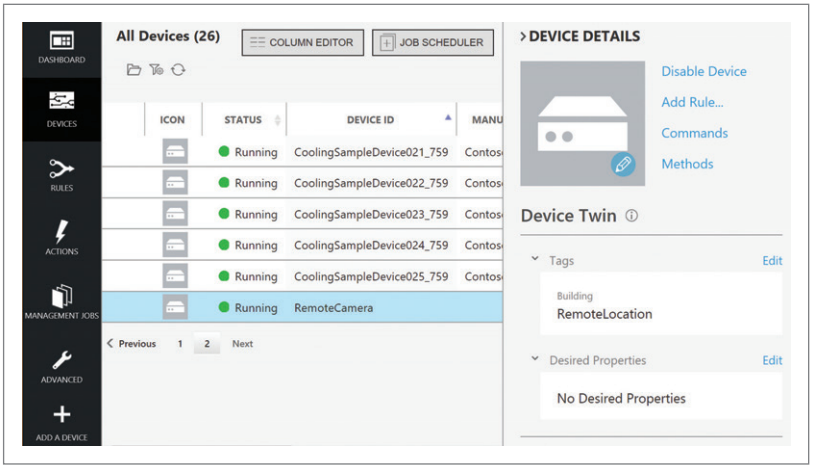


Figure 13 **The IoT Portal Devices Tab Showing RemoteCamera Details**

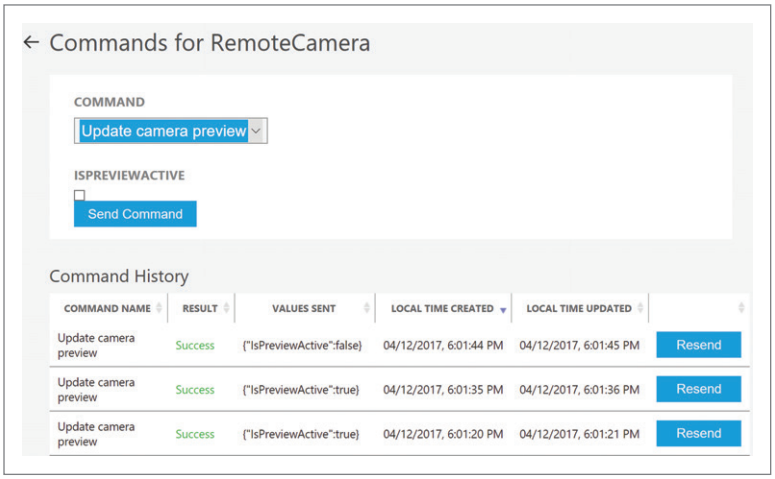


Figure 14 **A Form for Sending Remote Commands to the IoT Device**

Wrapping Up

I showed how to set up the preconfigured remote monitoring Azure IoT Suite solution. This solution collects and displays information about images acquired with the webcam attached to the remote device, and also lets you remotely control the IoT device. The companion source code can run on any UWP device, so you don't actually need to deploy it to the Raspberry Pi. This article, along with the online documentation for the remote monitoring solution and its source code, will help you to jump-start comprehensive IoT development for practical remote monitoring. As you see, with Windows 10 IoT Core, you can go far beyond simple LED blinking. ■

Dawid Borycki *is a software engineer and biomedical researcher, author and conference speaker. He enjoys learning new technologies for software experimenting and prototyping.*

Thanks to the following Microsoft technical expert for reviewing this article: *Rachel Appel*

GrapeCity

Build More

with GrapeCity Developer Solutions

Visual Studio-Integrated UI Controls and Developer Productivity Tools for Delivering Enterprise Apps Across All Platforms and Devices

ActiveReports

Powerful .NET reporting platform for essential business needs

ComponentOne Studio

Elegant, modular .NET UI controls for Visual Studio

Visual Studio

Powerful development tools engineered for Visual Studio developers

Spread Studio

Versatile .NET spreadsheet data and UI components

Xuni Xamarin Edition

Cross-platform grids, charts, and UI controls for native mobile devices

Wijmo JavaScript

Fast, lightweight true JavaScript controls written in TypeScript

FULL SUPPORT FOR Visual Studio 2017

GrapeCity's family of products provides **developers**, **designers**, and **architects** with the ultimate collection of easy-to-use tools for building **sleek**, **high-performing**, **feature-complete** applications. With over 25 years of experience, we understand your needs and offer the industry's best support. **Our team is your team**.
For more information: **1.800.858.2739**

Learn more and get free 30-day trials at
tools.grapecity.com

Visual Studio Microsoft Partner

Git Internals for Visual Studio Developers

Jonathan Waldman

In my commit to Git DevOps article (msdn.com/magazine/mt767697), I explained how the Git version control system (VCS) differs from centralized VCSes with which you might already be familiar. I then demonstrated how to accomplish certain Git tasks using the Git tooling in Visual Studio. In this article, I'll summarize relevant changes to how Git works within the newly released Visual Studio 2017 IDE, discuss how the Git repo is implemented in the file system, and examine the topology of its data store and the structure and content of its various storage objects. I'll conclude with a low-level explanation of Git branches, providing a perspective that I hope will prepare you for more advanced Git operations I'll present in upcoming articles.

Note: I use no servers or remotes in this article—I'm exploring a purely local scenario that you can follow using any Windows machine with Visual Studio 2017 and Git for Windows (G4W) installed (with or without an Internet or network connection). This article is an introduction to Git internals and assumes you're familiar with Visual Studio Git tooling and basic Git operations and concepts.

Visual Studio, Git and You

Git refers not only to the repository ("repo") that contains the version-control data store, but also to the engine that processes commands to manage it: Plumbing commands carry out low-level operations; porcelain commands bundle up plumbing commands in macro-like fashion, making them easier if less granular to invoke. As you master Git, you'll discover that some tasks require the use of these commands (some of which I'll use in this article), and that invoking them requires a command-line interface (CLI). Unfortunately, Visual Studio 2017 no longer installs a Git CLI because it uses a new Git engine called MinGit that doesn't provide one. MinGit ("minimal Git"), introduced with G4W 2.10, is a portable, reduced-feature-set API designed for Windows applications that need to interact with Git repositories. G4W and, by extension, MinGit, are forks of the official Git open source project. This means they both inherit the official Git fixes and updates as soon as they're available—and it ensures that Visual Studio can do the same.

To access a Git CLI (and to follow along with me), I recommend installing the full G4W package. While other Git CLI/GUI tooling options are available, G4W (as MinGit's official parent) is a wise choice—especially because it shares its configuration files with MinGit. To obtain the latest G4W setup, visit the Downloads section at the site's official source: git-scm.com. Run the setup program and select the Git Bash Here checkbox (creates a Git command-prompt window) and the Git GUI Here checkbox (creates a Git GUI window)—which makes it easy to right-click a folder in Windows Explorer—and select one of those two options for the current folder (the "Bash" in Git Bash refers to Bourne Again Shell, which presents a Git CLI in a Unix shell for G4W). Next, select Use Git from the Windows Command Prompt, which configures your environment so that you can conveniently run Git commands from either a Visual Studio package manager console (Windows PowerShell) or a command-prompt window.

If G4W is installed using the options I've recommended here, **Figure 1** depicts the communication pathways that will be in effect when communicating with a Git repo: Visual Studio 2017 uses the MinGit API while PowerShell and command-prompt sessions use the G4W CLI—a very different communication pathway en route to the Git repo. Although they serve as different communication endpoints, MinGit and G4W are derived from the official Git source code—and they share their configuration files. Notice that

This article discusses:

- Changes to how Git works within Visual Studio 2017
- How the Git repo is implemented in the file system
- The topology of its data store and the structure and content of its storage objects
- How Git internally defines a branch

Technologies discussed:

Visual Studio 2017, Git for Windows 2.10, Windows PowerShell

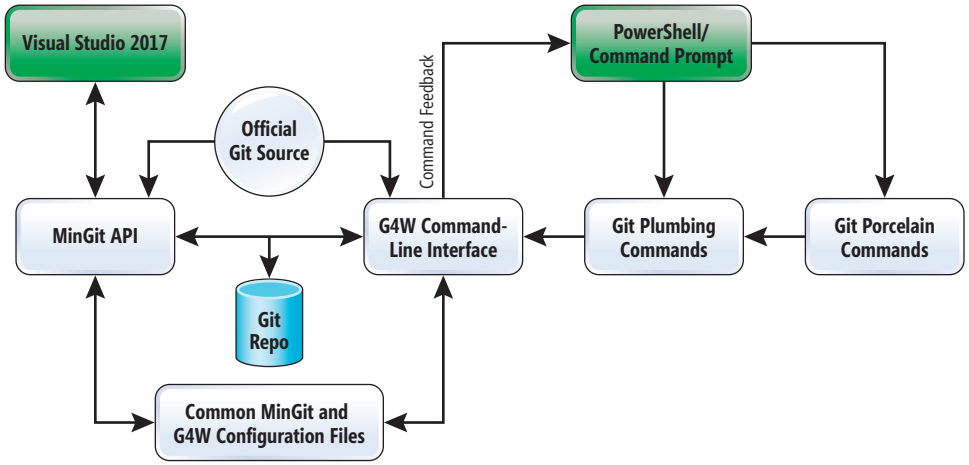


Figure 1 **Communication Paths to and from the MinGit API and the Git for Windows Command-Line Interface**

when you issue porcelain commands, they're translated into plumbing commands before being processed by the CLI. The point here is to understand that Git experts often resort to—and some thrive on—issuing bare-metal Git plumbing commands to a CLI because doing so is the most direct, lowest-level way with which to manage, query and update a Git repo. In contrast to low-level plumbing commands, higher-level porcelain commands and Git operations exposed by the Visual Studio IDE also can update the Git repo—yet it's not always clear how, especially because porcelain commands often accept options that change what they do when they're invoked. I've concluded that familiarity with Git plumbing commands is essential to wielding the power of Git and that's why I strongly recommend installing G4W alongside Visual Studio 2017. (Read details about Git plumbing and porcelain commands at git-scm.com/docs.)

Low-Level Git

It's natural for a Visual Studio developer to try to leverage existing knowledge of a VCS, such as Team Foundation Server (TFS), when transitioning to Git. Indeed, there's an overlap in the terms and concepts used to describe operations in both systems—such as checking out/checking in code, merging, branching and so on. However, the assumption that a similar vocabulary implies similar underlying operations is downright wrong and dangerous. That's because the decentralized Git VCS is fundamentally different in how it stores and tracks files and in the way it implements familiar version-control features. In short, when transitioning to Git, it's probably best to just forget everything you know about centralized VCSes and start afresh.

When you're working on a Visual Studio project that's under Git source control, the typical edit/stage/commit workflow works something like this: You add, edit and delete (collectively hereafter, "change") files in your project as needed. When ready, you stage some or all of those changes before committing them to the repo. Once committed, those changes become part of the repo's complete and transparent history. Now, let's see how Git manages all that internally.

The Directed Acyclic Graph Behind the scenes, each commit ends up as a vertex (node) on a Git-managed directed acyclic graph ("DAG," in graph-theory parlance). The DAG represents the Git repo and each vertex represents a data element known as a commit object (see **Figure 2**). Vertices in a DAG are connected with a line called an edge; it's customary for DAG edges to be drawn as arrows so they can express a parent/child relationship (the head points to the parent vertex; the tail points to the child vertex). The origin vertex represents the repo's first commit; a terminal vertex has no child. DAG edges express the exact parent-child relationship between each of its vertices. Because Git commit objects ("commits") are vertices, Git can leverage the DAG structure to model the parent-child relationship between every commit, giving Git its ability to generate a history of changes from any commit back to the repo's initial commit. Furthermore, unlike linear graphs, a DAG supports branches (a parent with more than one child), as well as merges (a child with more than one parent). A Git branch is spawned whenever a commit object produces a new child and a merge occurs when commit objects are combined to form a child.

I've explored the DAG and its associated terminology in great detail because such knowledge is a prerequisite to understanding advanced Git operations, which tend to work by manipulating vertices on the Git DAG. Furthermore, DAGs help to visualize the Git repo, and they're widely leveraged in teaching materials, during presentations and by Git GUI tools.

Git Objects Overview So far I've mentioned only the Git commit object. However, Git actually stores four different object types in its

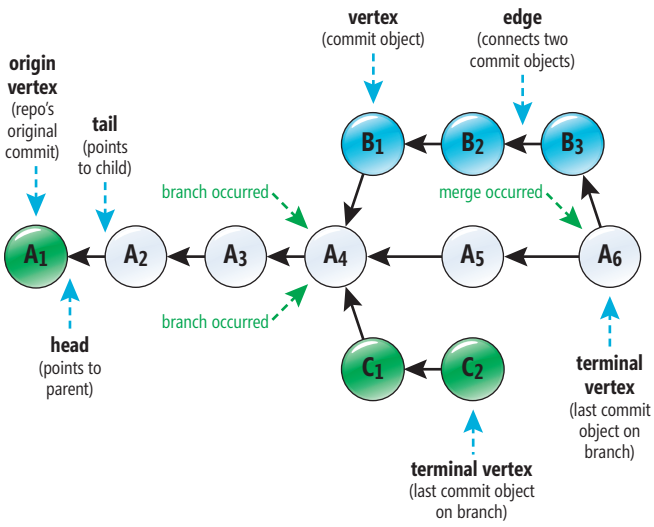


Figure 2 **A Directed Acyclic Graph Showing Vertex, Edge, Head, Tail, Origin Vertex and Terminal Vertices; Three Branches (A, B and C); Two Branch Events (at A4); and One Merge Event (B3 and A5 Are Merged at A6)**

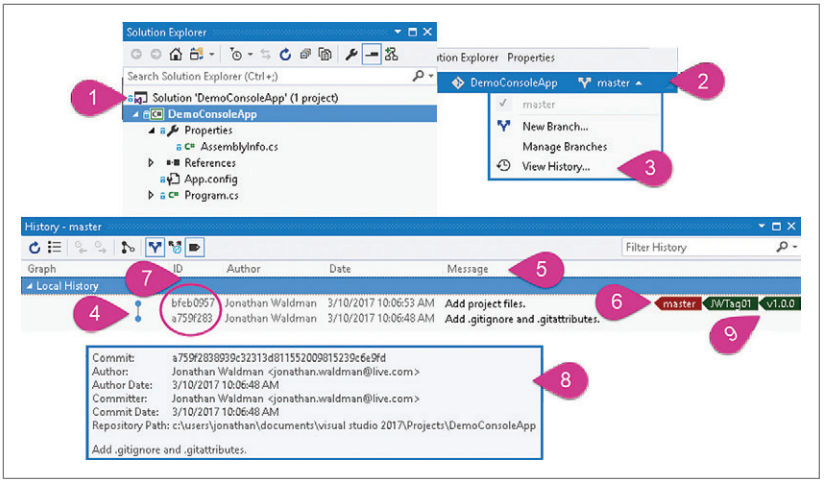


Figure 3 **A New Visual Studio Project with Its Git-Repo History Report**

repo: commit, tree, blob and tag. To investigate each of these, launch Visual Studio (I'm using Visual Studio 2017, but earlier versions that include Git support will work similarly) and create a new console application using File | New Project. Give your project a name, check the Create new Git repository checkbox and click OK. (If you haven't configured Git within Visual Studio before, you'll see a Git User Information dialog box appear. If you do, specify your name and e-mail address—this information is written to the Git repo for each of your commits—and check the "Set in global .gitconfig" checkbox if you want to use this information for every Git repo on your machine.)

When done, open a Solution Explorer window (see **Figure 3, Marker 1**). You'll see light-blue lock icons next to files that are checked-in—even though I haven't yet issued a commit! (This is an example showing that Visual Studio sometimes carries out actions against your repo that you might not anticipate.) To see exactly what Visual Studio did, look at the history of changes for the current branch.

Git names the default branch master and makes it the current branch. Visual Studio displays the current branch name at the right edge of its status bar (**Marker 2**). The current branch identifies the commit object on the DAG that will be the parent of the next commit (more on branches later). To view the commit history for the current branch, click the master label (**Marker 2**) then select View History (**Marker 3**) from the menu.

The History - master window displays several columns of information. At the left (**Marker 4**) are two vertices on the DAG; each graphically represents a commit on the Git DAG. The ID, Author, Date and Message columns (**Marker 5**) show details about the commit. The HEAD for the master branch is indicated by a dark red pointer (**Marker 6**)—I'll fully explain the meaning of this toward the end of this article. This HEAD marks the location of the next edge arrow's head after a commit adds a new vertex to the DAG.

The report shows that Visual Studio issued two commits, each with its own Commit ID (**Marker 7**). The first (earliest) is uniquely identified with ID a759f283; the second with bfeb0957. These values are truncated from the full 40-character hexadecimal Secure Hash Algorithm 1 (SHA-1). The SHA-1 is a cryptographic hash function engineered to detect corruption by taking a message, such as the commit data, and creating a message digest—the full SHA-1 hash value—such as the commit ID. In short, the SHA-1 hash acts not only like a checksum, but also like a GUID because it provides roughly 1.46×10^{48} unique combinations. Like many other Git tools, Visual Studio expresses only the first eight characters of the full value because these provide 4.3 billion unique values, enough to avoid collisions in your daily work. If you want to see the full SHA-1 value, hover the mouse over a line in the History report (**Marker 8**).

While the View History report's message column indicates the stated purpose of each commit (provided by the committer during a commit), it is, after all, just a comment. To examine a commit's actual changes, right-click a row in the list and select View Commit Details (see **Figure 4**).

The first commit (**Marker 1**) shows two changes: .gitignore and .gitattributes (I discussed these files in my previous article). The [add] next to each indicates files added to the repo. The second commit (**Marker 2**) shows five files added and also displays its parent-commit object's ID as a clickable link. To copy the entire SHA-1 value to the clipboard, simply click the Actions menu and select Copy Commit ID.

File-System Implementation of a Git Repo To see how Git stores these files in the repo, right-click the solution (not the project) in Solution Explorer and select Open Folder in File Explorer. In the solution's root you'll see a hidden folder called .git (if you don't see .git, click Hidden items in the File Explorer View menu). The .git folder is the Git repo for your project. Its objects folder defines the DAG: All DAG vertices and all parent-child relationships between each vertex are encoded by files that represent every commit in the repo starting with the origin vertex (refer back to **Figure 2**). The .git folder's HEAD file and refs folder define branches. Let's look at these .git items in detail.

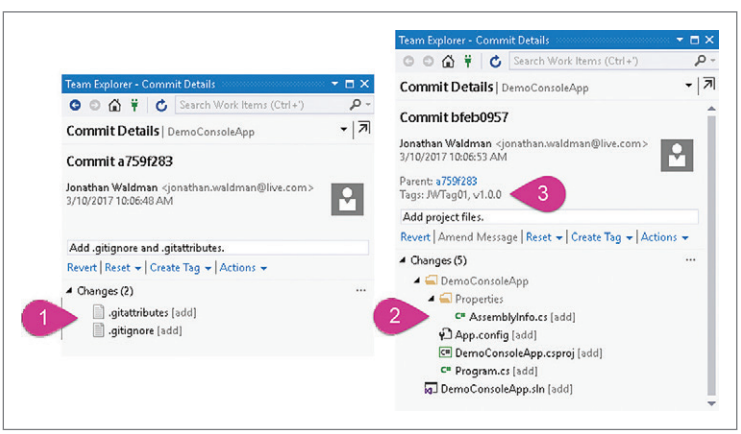


Figure 4 **Commit Details for the Repo's First Two Commits**

Exploring Git Objects

The .git\objects folder stores *all* Git object types: commit (for commits), tree (for folders), blob (for binary files) and tag (a friendly commit-object alias).

File Format APIs

Working with Files?

CREATE CONVERT PRINT

MODIFY COMBINE

FREE TRIAL

Aspose.Total

Manipulate Word, Excel, PDF, PowerPoint, Outlook and more than 100 other file formats in your applications without installing Microsoft Office.

DOC, XLS, PDF, PPT, MSG, BMP, PNG, XML and many more!

Platforms supported: .NET, Java, Cloud, Android, SharePoint, Reporting Services, and JasperReports

ASPOSE
File Format APIs

CONTACT US

US: +1 903 306 1676
EU: +44 141 628 8900
AU: +61 2 8006 6987

sales@asposeptyltd.com

Try for FREE at
www.aspose.com

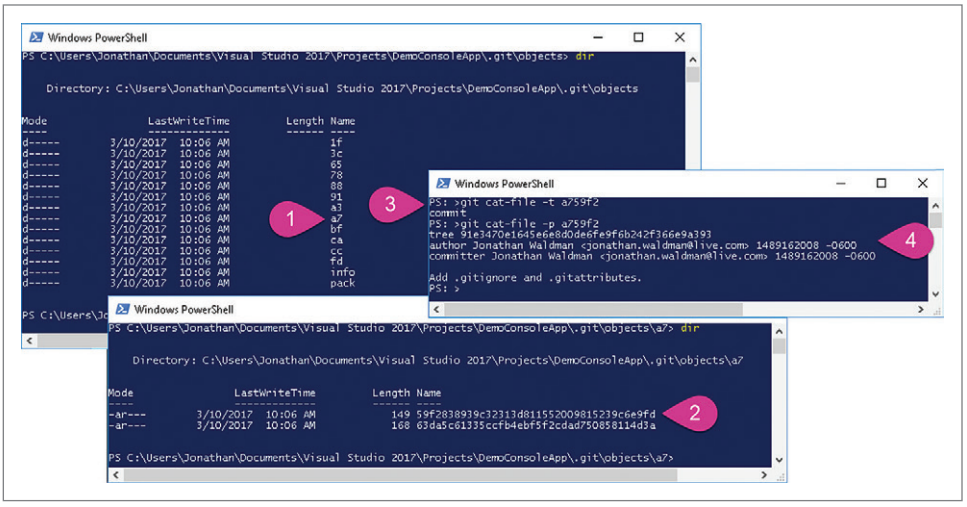


Figure 5 **Exploring Git Objects Using a Git Command-Line Interface**

Commit Object Now's the time to launch a Git CLI. You can use whichever tool you prefer (Git Bash, PowerShell or command window)—I'll use PowerShell. To begin, navigate to the solution root's .git\objects folder, then list its contents (**Figure 5, Marker 1**). You'll see that it contains a number of folders named using two-character hex values. To avoid exceeding the number of files permitted in a folder by the OS, Git removes the first two characters from each 40-byte SHA-1 value in order to create a folder name, then it uses the remaining 38 characters as the file name for the object to store. To illustrate, my project's first commit has ID a759f283, so that object will appear in a folder called a7 (the first two characters of the ID). As expected, when I open that folder, I see a file named 59f283. Remember that all of the files stored in these hex-named folders are Git objects. To save space, Git zlib-compresses the files in the object store. Because this kind of compression produces binary files, you won't be able to view these files using a text editor. Instead, you'll need to invoke Git commands that can properly extract Git-object data and present it using a format you can digest.

I already know that file 59f283 contains a commit object because it's a commit ID. But sometimes you'll see a file in the objects folder without knowing what it is. Git provides the cat-file plumbing command to report the type of an object, as well as its contents (**Marker 3**). To obtain the type, specify the -t (type) option when invoking the command, along with the few unique characters of the Git object's file name:

```
git cat-file -t a759f2
```

On my system, this reports the value "commit"—indicating that the file starting with a759f2 contains a commit object. Specifying only the first five characters of the SHA-1 hash value is usually enough, but you can provide as many as you want (don't forget to add the two characters from the folder name). When you issue the same command with the -p (pretty print) option, Git extracts information from the commit object and presents it in a format that's human-readable (**Marker 4**).

A commit object is composed of the following properties: Parent Commit ID, Tree ID, Author Name, Author Email Address, Author Commit Timestamp, Committer Name, Committer Email Address, Committer Commit Timestamp and Commit Message (the parent commit ID isn't displayed for the first commit in the repo). The SHA-1 for each commit object is computed from all of the values contained in these commit-object properties, virtually guaranteeing that each commit object will have a unique commit ID.

Tree and Blob Objects Notice that although the commit object contains information about the commit, it doesn't contain any files or folders. Instead, it has a Tree ID (also an SHA-1 value) that points to a Git tree object. Tree objects are stored in the .git\objects folder along with all other Git objects.

Figure 6 depicts the root tree object that's part of each commit object. The root tree object maps, in turn, to blob objects (covered next) and other tree objects, as needed.

Because my project's second commit (Commit ID bfeb09) includes files, as well as folders (see the earlier **Figure 4**), I'll use it to illustrate how the tree object works. **Figure 7, Marker 1** shows the cat-file -p bfeb09 output. This time, notice that it includes a parent property, which correctly references the SHA-1 value for the first commit object. (Remember that it's a commit object's parent reference that enables Git to construct and maintain its DAG of commits.)

The root tree object maps, in turn, to blob objects (zlib-compressed files) and other tree objects, as needed.

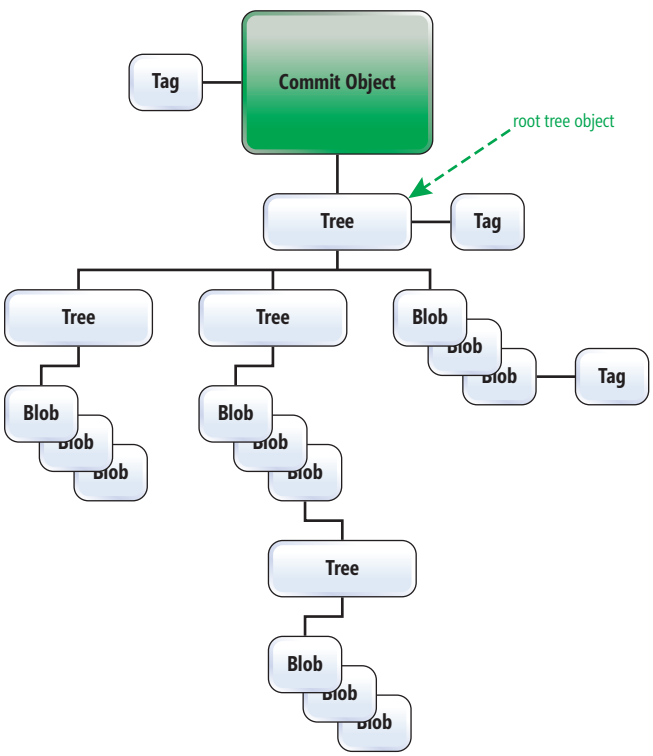


Figure 6 **Visualization of Git Objects That Express a Commit**

Commit bfeb09 contains a tree property with ID ca853d. **Figure 7, Marker 2** shows the cat-file -p ca853d output. Each tree object contains a permissions property corresponding to the POSIX permissions mask of the object (040000 = Directory, 100644 = Regular non-executable file, 100664 = Regular non-executable group-writeable file, 100755 = Regular executable file, 120000 = Symbolic link, and 160000 = Gitlink); type (tree or blob); SHA-1 (for the tree or blob); and name. The name is the folder name (for tree objects) or the file name (for blob objects). Observe that this tree object is composed of three blob objects and another tree object. You can see that the three blobs refer to files .gitattributes, .gitignore and DemoConsole.sln, and that the tree refers to folder DemoConsoleApp (**Figure 7, Marker 3**). Although tree object ca853d is associated with the project's second commit, its first two blobs represent files .gitattributes and .gitignore—files added during the first commit (see **Figure 4, Marker 1**)! The reason these files appear in the tree for the second commit is that each commit represents the previous commit object along with changes captured by the current commit *object*. To "walk the tree" one level deeper, **Figure 7, Marker 3** shows the cat-file -p a763da output, showing three more blobs (App.config, DemoConsoleApp.csproj and Program.cs) and another tree (folder Properties).

Blob objects are again just zlib-compressed files. If the uncompressed file contains text, you can extract a blob's entire content using the same cat-file command along with the blob ID (**Figure 7, Marker 5**). Because blob objects represent files, Git uses the SHA-1 blob ID to determine if a file changed from the previous commit; it also uses SHA-1 values when diffing any two commits in the repo.

Tag Object The cryptic alphanumeric nature of SHA-1 values can be a bit unwieldy to communicate. The tag object lets you assign a friendly name to any commit, tree or blob object—although it's most common to tag only commit objects. There are two types of tag object: lightweight and annotated. Both types appear as files in the .git\refs\tags folder, where the file name is the tag name. The content of a lightweight tag file is the SHA-1 to an existing commit, tree or blob object. The content of an annotation tag file is the SHA-1 to a tag object, which is stored in the .git\objects folder along with all other Git objects. To view the content of a tag object, leverage the same cat-file -p command. You'll see the SHA-1 value of the object that was tagged, along with the object type, tag author, date-time and tag message. There are a number of ways to tag commits in Visual Studio. One way is to click the Create Tag link in the Commit Details window (**Figure 4**). Tag names appear in the Commit Details window (**Figure 4, Marker 3**) and in View History reports (see the earlier **Figure 3, Marker 9**).

Git populates the info and pack folders in the .git\objects folder when it applies storage optimizations to objects in the repo. I'll discuss these folders and the Git file-storage optimizations more fully in an upcoming article.

Armed with knowledge about the four Git object types, realize that Git is referred to as a content-addressable file system because any kind of content across any number of files and folders can be reduced to a single SHA-1 value. That SHA-1 value can later be used to accurately and reliably recreate the same content. Put another way, the SHA-1 is the key and the content is the value in an exalted implementation of the usually prosaic key-index-driven lookup table. Additionally, Git can economize when file content hasn't changed between commits because an unchanged file produces the same SHA-1 value. This means that the commit object can reference the same SHA-1 blob or tree ID value used by a previous commit without having to create any new objects—this means no new copies of files!

Branching

Before truly understanding what a Git branch is, you must master how Git internally defines a branch. Ultimately, this boils down to grasping the purpose of two key terms: head and HEAD.

The first, head (all lowercase), is a reference Git maintains for every new commit object. To illustrate how this works, **Figure 8** shows several commits and branch operations. For Commit 01, Git creates the first head reference for the repo and names it master by default (master is an arbitrary name with no special meaning other than it's a default name—Git teams often rename this reference). When Git creates a new head reference, it creates a text file in its ref\heads folder and places the full SHA-1 for the new commit object into that file. For Commit 01, this means that Git creates a file called master and places the SHA-1 for commit object A_1 into that file. For Commit 02, Git updates the master head file in the heads folder by removing the old SHA-1 value and replacing it with the full SHA-1 commit ID for A_2. Git does the same thing for Commit 03: It updates the head file called master in the heads folder so that it holds the full commit ID for A_3.

You might have guessed correctly that the file called master in the heads folder is the branch name for the commit object to which it points. Oddly, perhaps, at first, a branch name points to a single commit object rather than to a sequence of commits (more on this specific concept in a moment).

Observe the Create Branch & Checkout Files section in **Figure 8**. Here, the user created a new branch for a print-preview feature in Visual Studio.

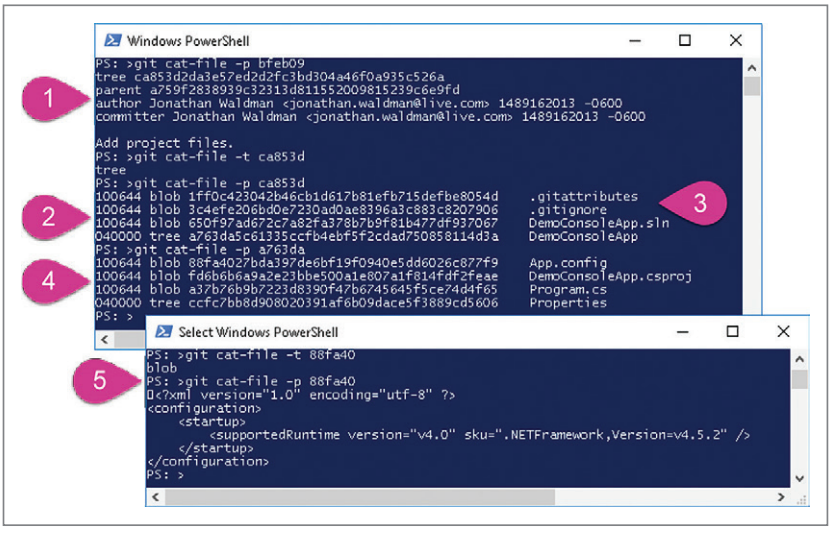


Figure 7 **Using the Git CLI to Explore Tree Object Details**

The user named the branch feat_print_preview, based it on master and checked the Checkout branch checkbox in Team Explorer's New Local Branch From pane. Checking the checkbox tells Git that you want the new branch to become the current branch (I'll explain this in a moment). Behind the scenes, Git creates a new head file in the heads folder called feat_print_preview and places the SHA-1 value for commit object A_3 into it. This now means that two files exist in the heads folder: master and feat_print_preview—both of which point to A_3.

At Commit 04, Git is faced with a decision: Normally, it would update the SHA-1 value for the file reference in the heads folder—but now it's got two file references in that folder, so which one should it update? That's what HEAD does. HEAD (all uppercase) is a single file in the root of the .git folder that points to a "head" (all lowercase) file in the heads folder. (Note that "HEAD" is actually a file that's always named HEAD, whereas "head" files have no particular name.) The head file HEAD contains the commit ID that will be assigned as the parent ID for the next commit object. In a practical sense, HEAD marks Git's current location on the DAG, and there can be many heads but there's always only one HEAD.

Going back to **Figure 8**, Commit 01 shows that HEAD points to the head file called master, which, in turn, points to A_1 (that is, the master head file contains the SHA-1 for commit object A_1). At Commit 02, Git doesn't need to do anything with the HEAD file because HEAD already points to the file master. Ditto for Commit 03. However, in the Create and Check-Out New Branch step, the user created a branch and checked out the files for the branch by marking the Checkout branch checkbox. In response, Git updates HEAD so that it points to the head file called feat_print_preview rather than to master. (If the user hadn't checked the Checkout branch checkbox, HEAD would continue to point to master.)

Armed with knowledge about HEAD, you now can see that Commit 04 no longer requires Git to make any decision: Git simply inspects the value of HEAD and sees that it points to the head file called feat_print_preview. It then knows that it must update the SHA-1 in the feat_print_preview head file so that it contains the commit ID for B1.

In the Checkout Branch step, the user accessed the Team Explorer branches pane, right-clicked the master branch and chose Checkout. In response, Git checks out the files for Commit A_3 and updates the HEAD file so that it points to the head file called master.

At this point, it should be clear why branch operations in Git are so efficient and fast: Creating a new branch boils down to creating one text file (head) and updating another (HEAD). Switching branches involves updating only a single text file (HEAD) and then usually a small performance hit as files in the working directory are updated from the repo.

Notice that commit objects contain no branch information! In fact, branches are maintained by only the HEAD file and the various files in the heads folder that serve as references. Yet when developers using Git talk about being on a branch or they refer to a branch, they often are colloquially referring to the sequence of commit objects that originates with master or from a newly formed branch. The earlier **Figure 2** shows what many developers would identify as three branches: A, B and C. Branch A follows the sequence A_1 through A_6. Branch activity at A_4 produces two new branches: B_1 and C_1. So the sequence of commits that begins with B_1 and continues to B_3 can be referred to as Branch B while the sequence from C_1 to C_2 can be referred to as Branch C.

The takeaway here is not to forget the formal definition of a Git branch: It's simply a pointer to a commit object. Moreover, Git maintains branch pointers for all branches (called heads) and a single branch pointer for the current branch (called HEAD).

In my next article I'll explore details about checking out and checking in files from and to the repo, and the role of index and how it constructs tree objects during each commit. I'll also explore how Git optimizes storage and how merges and diffs work. ■

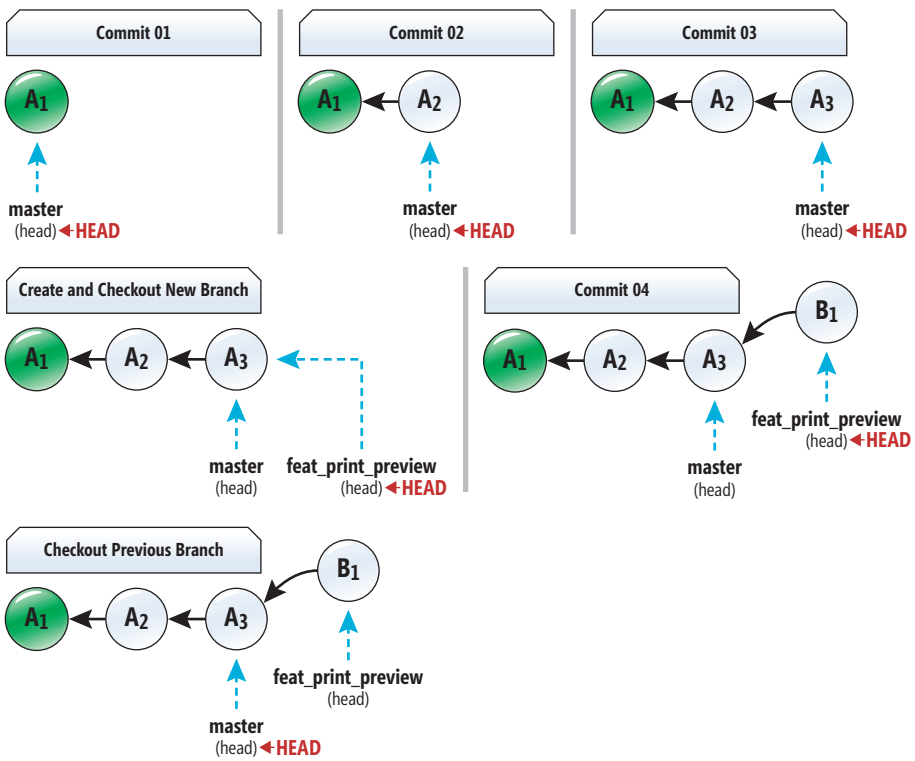
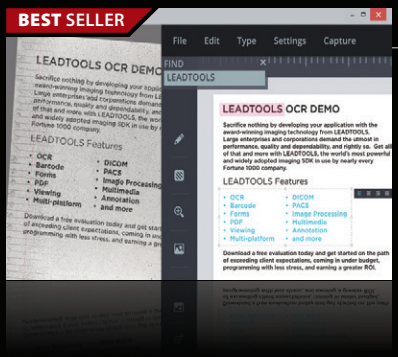


Figure 8 **Two Heads Are Better Than One: Git Maintains Various Files in Its Heads Folder Along with a Single HEAD File**

Jonathan Waldman *is a Microsoft Certified Professional who has worked with Microsoft technologies since their inception and who specializes in software ergonomics. Waldman is a member of the Pluralsight technical team and he currently leads institutional and private-sector software-development projects. He can be reached at jonathan.waldman@live.com.*

Thanks to the following Microsoft technical experts for reviewing this article: *Kraig Brockschmidt and Ralph Squillace*

BEST SELLER



LEADTOOLS Document Imaging SDKs V19 | from $2,995.00 SRP

LEAD TECHNOLOGIES

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

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

BEST SELLER

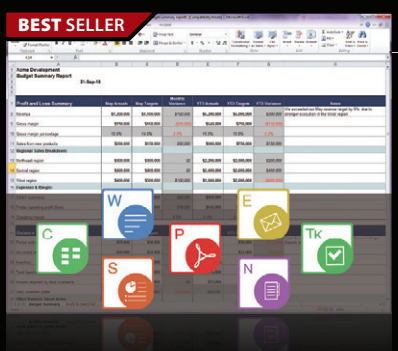
Help & Manual Professional | from $586.04

ec software

Help and documentation for .NET and mobile applications.

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

BEST SELLER



Aspose.Total for .NET | from $2,939.02

ASPOSE
File Format APIs

Every Aspose .NET API in one package.

- Programmatically manage popular file formats including Word, Excel, PowerPoint and PDF
- Work with charts, diagrams, images, project plans, emails, barcodes, OCR, Visio, OneNote, 3D and CAD files alongside many more document management features in your .NET applications
- Common uses also include mail merging, adding barcodes to documents, building dynamic reports on the fly and extracting text from most document types

BEST SELLER

DevExpress DXperience 16.2 | from $1,439.99

DevExpress

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

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

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

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

European Headquarters
ComponentSource
2 New Century Place
East Street
Reading, Berkshire
RG1 4ET
United Kingdom

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

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

GSA Schedule Contract GS-35F-0188R

Authentication and Data Access with Visual Studio Mobile Center

Alessandro Del Sole

At Connect(); 2016, Microsoft unveiled a preview of Visual Studio Mobile Center (mobile.azure.com), a new portal that offers—in one place—everything developers need to build mobile apps in a DevOps approach, from back-end services and continuous integration, to analytics and test automation. Visual Studio Mobile Center puts together the tools and services Microsoft has been offering through Microsoft Azure Mobile Apps, Visual Studio Team Services, Xamarin Test Cloud and HockeyApp. I recommend you read the article, "Mobile DevOps Exploring Visual Studio Mobile Center," by Thomas Dohmke (bit.ly/2f7a8Wk) for a general overview of what's available in Visual Studio Mobile Center.

In this article, I'll focus on authentication and tables with a practical approach, describing how to consume these services from a cross-platform app that runs on Android and iOS, built with Xamarin.Forms. The sample app will show and store a list of books a user purchased and is based on the Model-View-ViewModel (MVVM) pattern for better code reuse. Some of the features described here require an Azure subscription. If you don't have one, you can request a free trial at azure.microsoft.com. You need Visual Studio 2017 or 2015 with the latest Xamarin extensions installed to build the sample application. You can download the free Community edition from visualstudio.com.

This article discusses:

- Adding an app to the Visual Studio Mobile Center
- Creating tables
- Setting up authentication
- Consuming back-end services from Xamarin clients

Technologies discussed:

Xamarin, Microsoft Azure Platform, Visual Studio 2017, Android, iOS

Code download available at:

msdn.com/magazine/0617magcode

> In this article, I'll focus on authentication and tables with a practical approach.

Adding an App to the Visual Studio Mobile Center

In order to leverage any service from Visual Studio Mobile Center, the first thing you need to do is associate a new app to the portal. To accomplish this, you need to enter the portal (mobile.azure.com) and sign in with either a Microsoft or GitHub account. The welcome page of the Mobile Center shows a button called Add new app. As shown in **Figure 1**, when you click this button, you need to specify the app name, an optional description, the target OS and the development platform.

At the time of this writing, Mobile Center only supports iOS and Android applications (support for Windows is planned), and the list of available development platforms varies depending on the OS of your choice. Enter the information as shown in **Figure 1**, making sure you select Xamarin, and click Add new app. Feel free to select iOS instead of Android if you have a Mac computer that you can use to build Xamarin projects, as the sample code will run on both. Also, you can associate an app twice in order to enable the support for analytics and crash reports on both OSes.

After this, the Visual Studio Mobile Center shows how to include analytics and crash reports in a Xamarin app (including Xamarin.Forms) using the Mobile Center SDK NuGet packages. This will be done later in the sample code, so let's move forward.

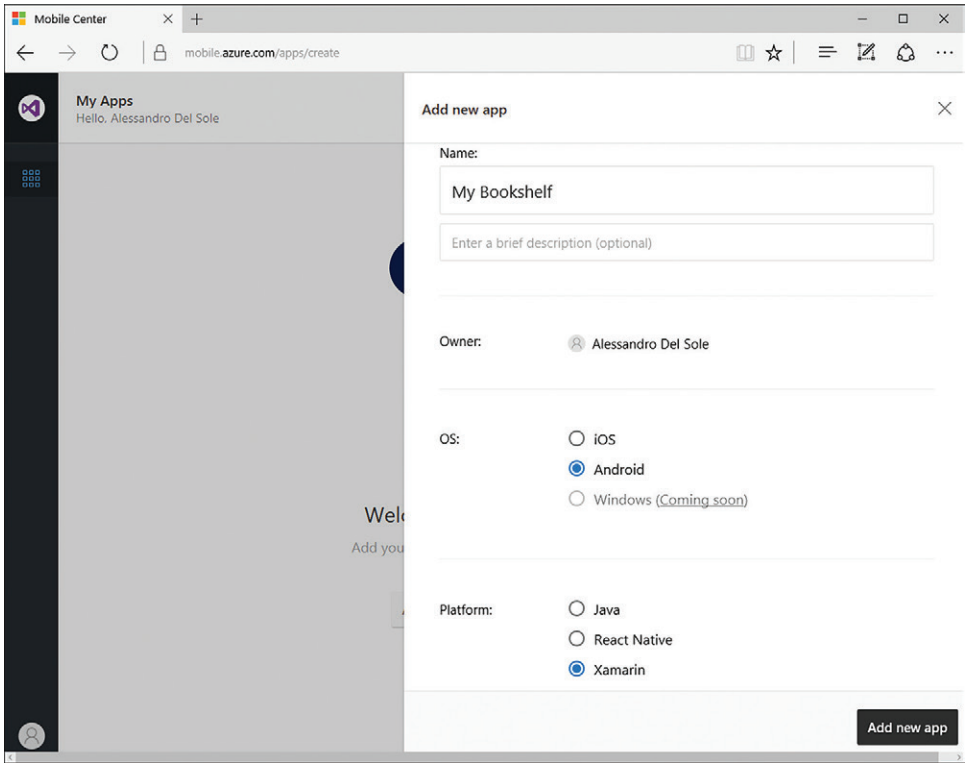


Figure 1 **Adding an App to Visual Studio Mobile Center**

Creating Tables

You'll now set up a new table to store a list of books, so click Tables in the toolbar. Because Mobile Center relies on the Azure platform for identity and tables, you'll be asked to connect your Azure subscription with the Connect Subscription button. Once the subscription has been connected, you'll be able to click Create Table to get started. At this point, Visual Studio Mobile Center creates a new mobile app back end and an app endpoint in the Azure Mobile Apps service. See the endpoint form at https://mobile-{your-app-id}.azurewebsites.net/. In the popup that appears, enter Book as the new table name. Then, select Authenticated, Soft delete and Per-user data checkboxes and unselect Dynamic schema. The following is a brief description of each option:

- **Authenticated** lets you restrict access to the table to only authenticated users. Anonymous access is granted when this option is not selected.
- **Soft delete** lets you mark records as deleted instead of removing them from the database. This gives you an option to undelete records. Queries can be filtered to return only active records.
- **Dynamic schema** lets you dynamically create new columns on the client instead of the server when new objects are inserted. This is beneficial during development, but a security problem during production usage, so it should be avoided when the app graduates to production.
- **Per-user data** adds a user column to the table and lets you filter data based on the current user information. Selecting this option makes sense when authentication is enabled.

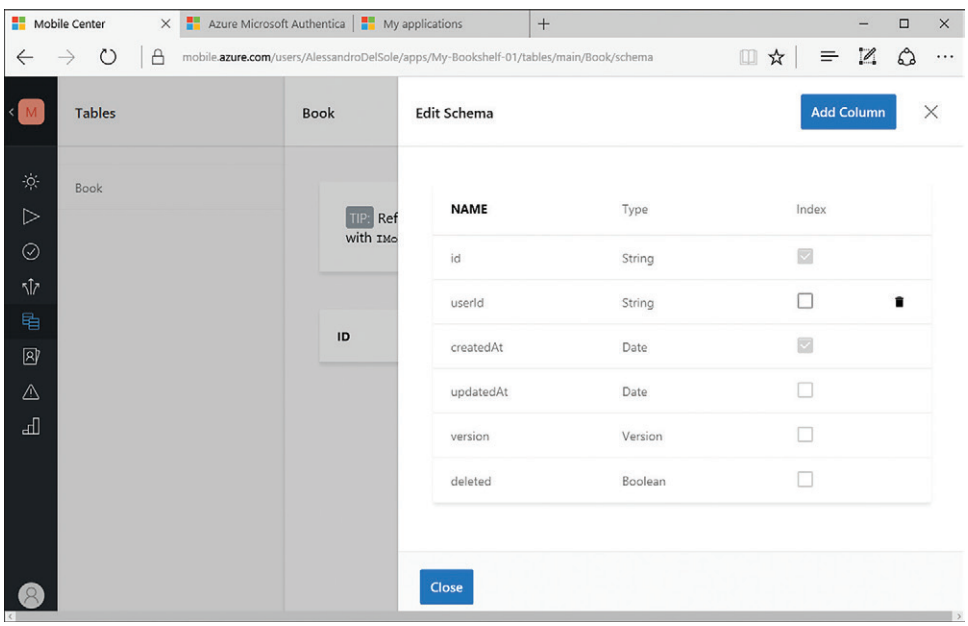


Figure 2 **The Schema of a New Table, Including System Columns**

When you click Create, Mobile Center will generate a table with a number of system columns, which you can see by selecting the Show system columns switch control or when you click Edit Schema to add your columns, as shown in **Figure 2**.

Columns can be of five different types: String, Date, Number, Boolean and Version. Actually, Version is only intended for internal usage with the version column. These types are very flexible because they can receive a variety of platform-specific data types. For instance, in the Microsoft .NET Framework, you can store objects of type Int32 and Double into a column of type Number, and this applies to the other supported development platforms, as well. It's worth noting that using DateTimeOffset for dates is preferred to using DateTime, as the former includes and takes into account time zone information, whereas the latter does not. Now, click Add Column and add two simple columns: Title and Author (of type String). A popup appears and lets you enter the column name and the data type. Once you've added the desired columns, an additional step is required to set column permissions. To accomplish this, you use the options button (an icon with three dots aligned vertically), which provides a menu with several commands. Among the others, the Change Permissions command lets you specify individual permissions for each operation against data, as you can see in **Figure 3**.

Click on Save when ready. It's worth mentioning that the options menu lets you upload data from .csv files, as well as clear data in the table and delete the table. Another very useful feature is that the page for a table will also show the stored data inside a grid.

Setting Up Authentication

The next step is to set up authentication. In Mobile Center, click the Identity button in the toolbar on the left side of the page. You'll see the authentication endpoint for Azure Active Directory (Azure AD) and the list of available authentication providers: Azure Active Directory, Microsoft Account, Facebook, Google and Twitter. You can definitely select multiple authentication services, but for this example, use the Microsoft Account service. Whatever authentication service you click, a popup appears showing a link to the documentation for each provider and a list of fields that must be configured based on the selected provider. In the case of the Microsoft Account service, the documentation (available at bit.ly/2peCBgf) explains how to register an app and how to retrieve the app ID and app client secret through the following steps:

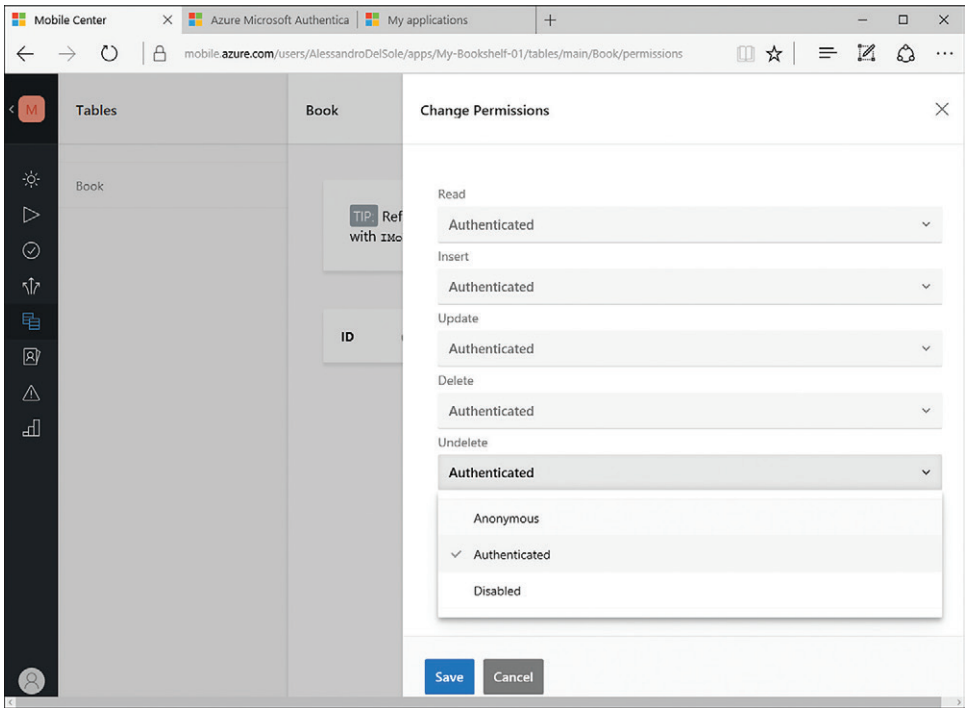


Figure 3 **Specifying Permissions for a Table**

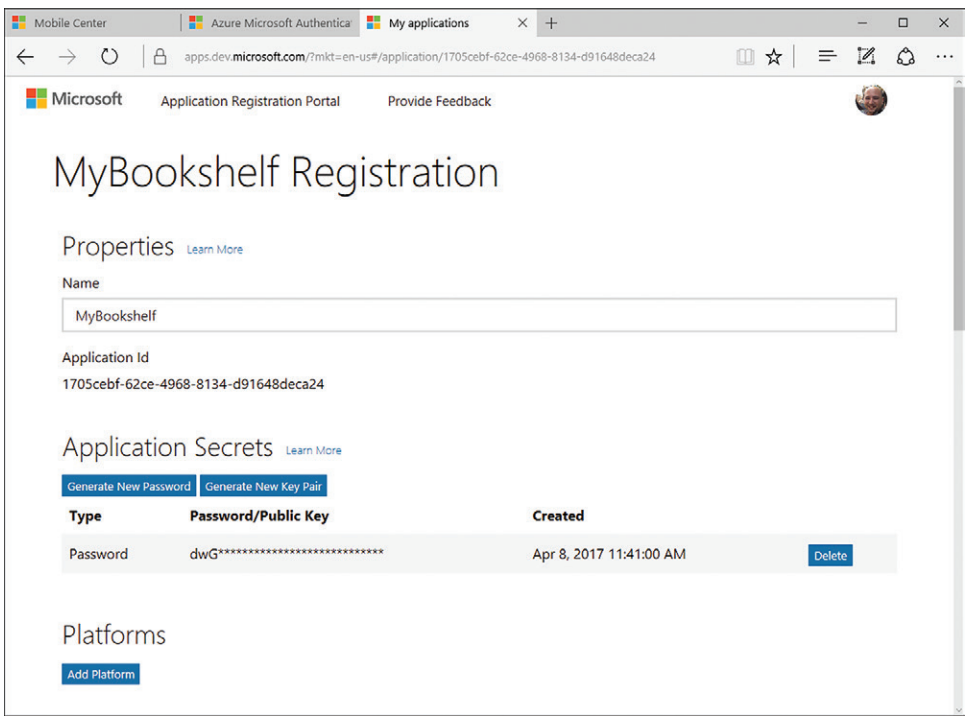


Figure 4 **Registering an Application in the Microsoft Developer Portal**

DocuVieware 3

HTML5 Viewer & Document Management Kit

NEW RELEASE

Easy integration

Full support for custom snap-in

Zero-footprint solution

Fully customizable UI

Mobile devices optimization

Fast & crystal-clear rendering

Check the **New Features** and the **Online Demos**

DOWNLOAD YOUR FREE TRIAL www.docuvieware.com

Powered by GdPicture.NET 14

DocuVieware is an ORPALIS imaging technologies product

1. **Open the Microsoft Account Developer Portal.** This is available at apps.dev.microsoft.com. Here, you'll see a list of apps associated with your Microsoft Account (if any).
2. **Click Add an App.** You'll be asked to specify an app name, which must not include special characters. For the current example, you can enter MyBookshelf. Then, click Create Application.
3. **Generate ID and client secret.** When a new app is added, the developer portal generates the Application Id, which is visible in the application page, as shown in **Figure 4**. You also need a Client Secret, which you obtain by clicking Generate New Password. A popup will show the Client Secret, and this is actually the only time you'll be able to see it clearly, so copy it into a safe location for later use.
4. **Specify a redirect URI.** This is required to tell the authentication service what page it will need to show after the user has supplied the credentials. To accomplish this, click Add Platform, then Web. In the Redirect URIs box you must enter the proper redirect URI, which has the following form: https://mobile-{your-app-id}.azurewebsites.net/.auth/login/microsoftaccount/callback. **Figure 5** demonstrates this step.

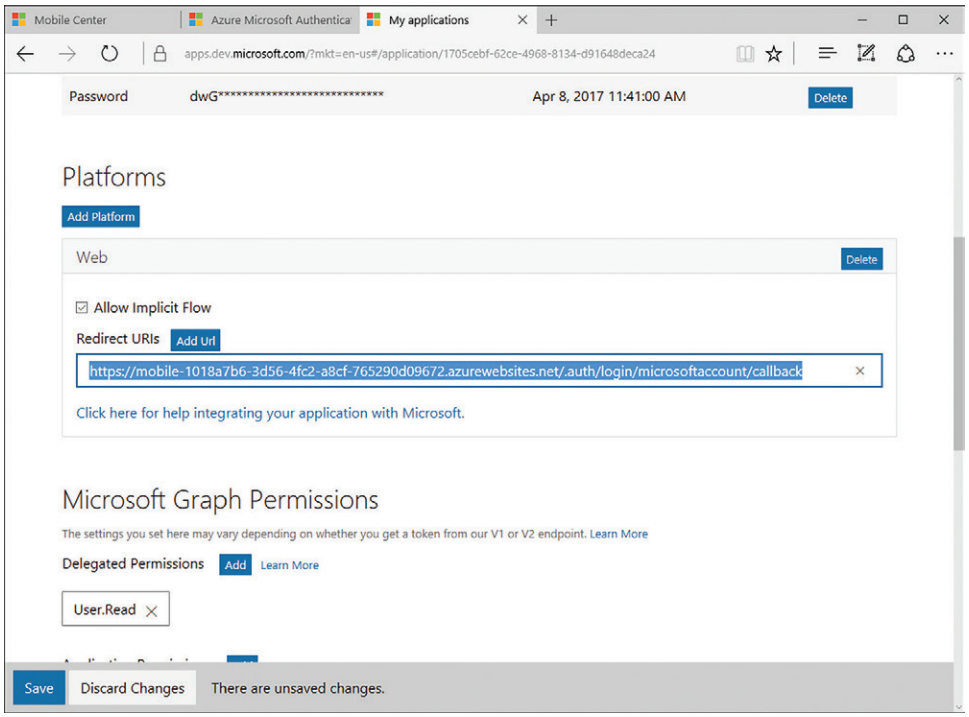


Figure 5 **Entering a Redirect URI**

Save your changes, go back to Mobile Center and add the Client ID and Client Secret to the configuration page for your app, as shown in **Figure 6**.

You can specify additional permissions under Scope, but this is not the case here, so click Save. With a few steps, you've successfully configured authentication for your app; similar steps apply to the other authentication providers. Now it's time to create a cross-platform app and write some code.

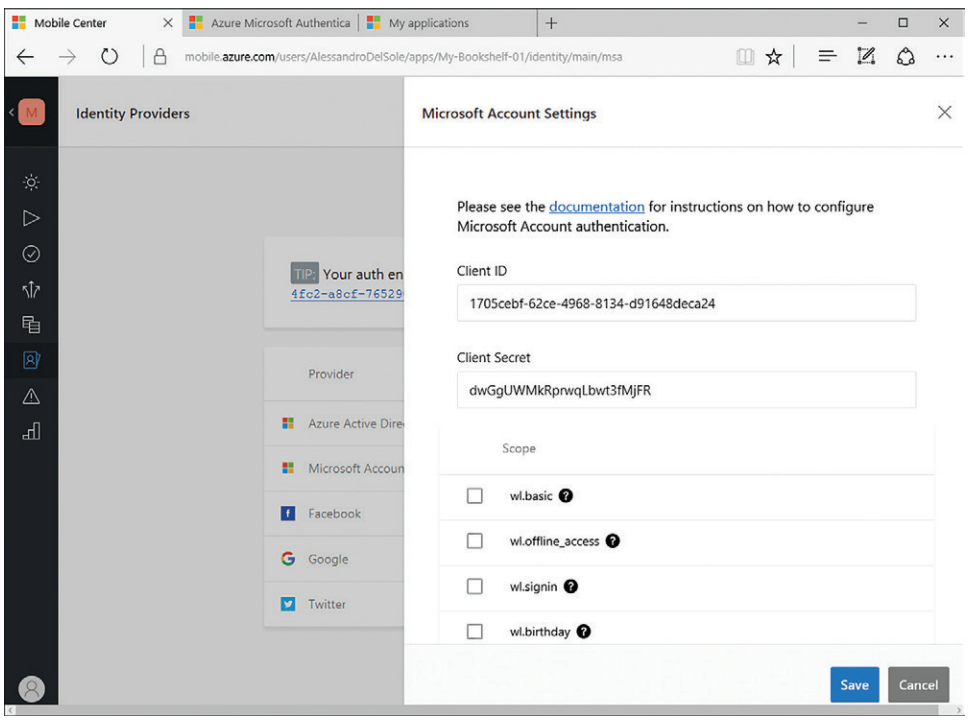


Figure 6 **Configuring the Microsoft Account Authentication Service**

Creating and Setting Up a Xamarin Sample Project

In Visual Studio 2017 select File | New | Project. Locate the Cross-Platform node under Visual C# and select the project template called Cross Platform App (Xamarin.Forms or Native). Call the project MyBookshelf and click OK. In the New Cross Platform App dialog, make sure you select the Blank App template, then Xamarin.Forms under UI Technology and Portable Class Library (PCL) under Code Sharing Strategy, and click OK. In order to work against Visual Studio Mobile Center services, including the Mobile Center SDK, you need to download and install the following NuGet packages:

- **Microsoft.Azure.Mobile.Crashes:** Enables an app to share crash information with the Mobile Center. It has a dependency on the Microsoft.Azure.Mobile package, which is also installed.
- **Microsoft.Azure.Mobile.Analytics:** Enables an app to share usage information with the Mobile Center and has the same dependency on Microsoft.Azure.Mobile.
- **Microsoft.Azure.Mobile.Client:** Installs the Azure Mobile Client SDK and lets a Windows or Xamarin app work with the Azure Mobile Apps service.

The next step is modeling objects that map tables in the back-end service.

Defining a Data Model

For better code reuse, you can create a base class that maps the basic structure of a table, and then write derived classes that extend the base object with specific columns. The base class must also implement the INotifyPropertyChanged interface to raise change notification with data binding to the UI. Having said that,

Figure 7 **Defining the Data Model**

```
// Requires the following directives:
// using System.ComponentModel and System.Runtime.CompilerServices
public abstract class TableBase : INotifyPropertyChanged
{
  private string id;
  public string Id
    {
      get
      {
        return id;
      }

      set
      {
        id = value; OnPropertyChanged();
      }
    }

    private string userId;
    public string UserId
    {
      get
      {
        return userId;
      }

      set
      {
        userId = value; OnPropertyChanged();
      }
    }

    private DateTimeOffset createdAt;
    public DateTimeOffset CreatedAt
    {
      get
      {
        return createdAt;
      }

      set
      {
        createdAt = value; OnPropertyChanged();
      }
    }

    private DateTimeOffset updatedAt;
    public DateTimeOffset UpdatedAt
    {
      get
      {
        return updatedAt;
      }

      set
      {
        updatedAt = value; OnPropertyChanged();
      }
    }

    private string version;
    public string Version
    {
      get
      {
        return version;
      }

      set
      {
        version = value; OnPropertyChanged();
      }
    }

    private bool deleted;
    public bool Deleted
    {
      get
      {
        return deleted;
      }

      set
      {
        deleted = value; OnPropertyChanged();
      }
    }

    public TableBase()
    {
      this.CreatedAt = DateTimeOffset.Now;
      this.UpdatedAt = DateTimeOffset.Now;
    }

    public event PropertyChangedEventHandler PropertyChanged;

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

public class Book: TableBase
{
    private string author;
    public string Author
    {
      get
      {
        return author;
      }
      set
      {
        author = value; OnPropertyChanged();
      }
    }

    private string title;
    public string Title
    {
      get
      {
        return title;
      }
      set
      {
        title = value; OnPropertyChanged();
      }
    }
}
```

TEXT CONTROL

THE TOP 5 EMR/EHR HEALTHCARE SOFTWARE VENDORS INTEGRATED TEXT CONTROL REPORTING

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

Our award-winning developer libraries are completely independent from MS Word or any other third-party application and can be seamlessly integrated into your business application.

www.textcontrol.com


Visual Studio
Partner
WE'RE
CHANGING
THE WAY
YOU LOOK AT
REPORTING
Microsoft
.NET
WINDOWS FORMS
WPF
ASP.NET MVC
ASP.NET AJAX
CLOUD WEB API

Figure 8 **Authentication.cs in Android**

```
using System;
using MyBookshelf.Authentication;
using Microsoft.WindowsAzure.MobileServices;
using Xamarin.Forms;
using System.Threading.Tasks;
using MyBookshelf.Droid;

[assembly: Dependency(typeof(Authentication))]
namespace MyBookshelf.Droid
{
  public class Authentication : IAuthentication
  {
    public async Task<MobileServiceUser> LoginAsync(MobileServiceClient client,
      MobileServiceAuthenticationProvider provider)
    {
      try
      {
        var user = await client.LoginAsync(Forms.Context, provider);
        return user;
      }
      catch (Exception)
      {
        return null;
      }
    }
  }
}
```

you now add to the PCL project a folder called Model and two code files called TableBase.cs and Book.cs. **Figure 7** shows the full code for both class definitions.

With this approach, you don't need to repeat the same common properties for each class; you can simply take advantage of inheritance. In the PCL project, add a code file called Constants.cs, whose content is the following:

```
public static class Constants
{
  public const string BaseUrl = "https://mobile-{your-app-id}.azurewebsites.net/";
}
```

With this approach, you don't need to repeat the same common properties for each class; you can simply take advantage of inheritance.

This constant will contain the URL of your back-end service and will be used instead of writing the URL manually every time.

Implementing Authentication

The Azure Mobile Client SDK makes authenticating a user very simple. In fact, you just need to invoke the MobileService-

Figure 9 **Authentication.cs in iOS**

```
using Microsoft.WindowsAzure.MobileServices;
using MyBookshelf.Authentication;
using MyBookshelf.iOS;
using System;
using System.Threading.Tasks;

[assembly: Xamarin.Forms.Dependency(typeof(Authentication))]
namespace MyBookshelf.iOS
{
  public class Authentication : IAuthentication
  {
    public async Task<MobileServiceUser> LoginAsync(MobileServiceClient client,
      MobileServiceAuthenticationProvider provider)
    {
      try
      {
        // Attempt to find root view controller to present authentication page
        var window = UIKit.UIApplication.SharedApplication.KeyWindow;
        var root = window.RootViewController;
        if (root != null)
        {
          var current = root;
          while (current.PresentedViewController != null)
          {
            current = current.PresentedViewController;
          }

          // Login and save user status
          var user = await client.LoginAsync(current, provider);
          return user;
        }
        return null;
      }
      catch (Exception)
      {
        return null;
      }
    }
  }
}
```

Figure 10 **Implementing a Data Access Layer**

```
using Microsoft.WindowsAzure.MobileServices;
using MyBookshelf.Model;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace MyBookshelf.DataAccess
{
  public class DataManager
  {
    public MobileServiceClient Client { get; set; }

    public DataManager(string serviceUrl)
    {
      Client = new MobileServiceClient(serviceUrl);
    }

    public async Task SaveAsync<T>(IEnumerable<T> items) where T : TableBase
    {
      var table = this.Client.GetTable<T>();

      foreach (T item in items)
      {
        if (item.Id == null)
        {
          item.Id = Guid.NewGuid().ToString("N");
          await table.InsertAsync(item);
        }
        else
        {
          await table.UpdateAsync(item);
        }
    }
  }

    public async Task<IEnumerable<T>> LoadAsync<T>(string userId)
      where T : TableBase
    {
      var table = this.Client.GetTable<T>();

      var query = await table.Where(t => t.UserId == userId).ToEnumerableAsync();

      return query;
    }
  }
}
```

Figure 11 **Exposing a ViewModel**

```
using Microsoft.WindowsAzure.MobileServices;
using MyBookshelf.Authentication;
using MyBookshelf.Model;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Runtime.CompilerServices;
using System.Threading.Tasks;
using Xamarin.Forms;

namespace MyBookshelf.ViewModel
{
  public class BookViewModel : INotifyPropertyChanged
  {
    public event PropertyChangedEventHandler PropertyChanged;

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

    private bool isBusy;
    public bool IsBusy
    {
      get
      {
        return isBusy;
      }
      set
      {
        isBusy = value; OnPropertyChanged();
      }
    }

    private bool isUserAuthenticated;

    public bool IsUserAuthenticated
    {
      get
      {
        return isUserAuthenticated;
      }

      set
      {
        isUserAuthenticated = value; OnPropertyChanged();
      }
    }

    private string userId;

    public string UserId
    {
      get
      {
        return userId;
      }

      set
      {
        userId = value; OnPropertyChanged();
      }
    }

    private ObservableCollection<Book> books;

    public ObservableCollection<Book> Books
    {
      get
      {
        return books;
      }

      set
      {
        books = value; OnPropertyChanged();
      }
    }

    private Book selectedBook;

    public Book SelectedBook
    {
      get
      {
        return selectedBook;
      }

      set
      {
        selectedBook = value; OnPropertyChanged();
      }
    }

    public BookViewModel()
    {
      this.Books = new ObservableCollection<Book>();
      this.IsBusy = false;
    }


    public Command SaveBooks
    {
      get
      {
        return new Command(async () =>
        {
          if (IsUserAuthenticated)
          {
            this.IsBusy = true;
            await App.DataManager.SaveAsync(this.Books);
            this.IsBusy = false;
          }
      });
      }

    }

    public Command LoadBooks
    {
      get
      {
        return new Command(async () =>
        {
          if (IsUserAuthenticated)
          {
            this.IsBusy = true;
            var listOfBooks = await App.DataManager.LoadAsync<Book>(UserId);
            this.Books = new ObservableCollection<Book>(listOfBooks);
            this.IsBusy = false;
          }
        });
      }
    }

    public Command AddNewBook
    {
      get
      {
        return new Command(() =>
          this.Books.Add(new Book { UserId = this.UserId }));
      }
    }

    public async Task LoginAsync()
    {
      var authenticator = DependencyService.Get<IAuthentication>();
      var user = await authenticator.LoginAsync(App.DataManager.Client,
        MobileServiceAuthenticationProvider.MicrosoftAccount);
      if (user == null)
      {
        IsUserAuthenticated = false;
        return;
      }
      else
      {
        UserId = user.UserId;
        IsUserAuthenticated = true;
        return;
      }
    }
  }
}
```

Client.LoginAsync method. LoginAsync takes two arguments: the context in which the authentication UI must be presented and the authentication provider via the MobileServiceAuthenticationProvider enumeration. It returns an object of type MobileServiceUser, which contains information about the authenticated user. In Xamarin.Forms, the code that authenticates the user can't be shared in the PCL because iOS and Android manage UI contexts differently. The problem can be solved using the dependency service pattern, therefore adding to the PCL an interface that provides an abstract login method definition, and platform-specific implementations of the interface. In the PCL project, add a new folder called Authentication and an interface called IAuthentication.cs that looks like the following:

```
using Microsoft.WindowsAzure.MobileServices;
using System.Threading.Tasks;

namespace MyBookshelf.Authentication
{
  public interface IAuthentication
  {
    Task<MobileServiceUser> LoginAsync(MobileServiceClient client,
      MobileServiceAuthenticationProvider provider);
  }
}
```

This custom definition of LoginAsync takes an instance of the MobileServiceClient class that's passed by a View Model and that will be used to invoke its LoginAsync method from the platform-specific projects. Add a file called Authentication.cs to both the Android and iOS projects, then look at **Figure 8** and **Figure 9**, which show the Android and iOS implementations, respectively.

With this approach, each platform-specific project can invoke LoginAsync passing the proper execution context.

> In Xamarin.Forms, the code that authenticates the user can't be shared in the PCL because iOS and Android manage UI contexts differently.

Implementing Data Access with the Azure Mobile Client SDK

The MobileServiceClient class also provides everything you need to work with records in a table. You first obtain a reference to a table with the GetTable method, then with methods such as ToEnumerableAync, InsertAsync, UpdateAsync and DeleteAsync, you can query, insert, update and delete records, respectively. These are all generic methods, so you can write a data access layer that can be reused to work with any object that derives from TableBase. Having said that, in the PCL project, add a DataAccess folder and a file called DataManager.cs inside. Before writing the DataManager class, add the following code to the App.xaml.cs file, so that a variable of type DataManager is available at the app level:

```
internal static DataManager DataManager;
public App()
{
  InitializeComponent();
  DataManager = new DataManager(Constants.BaseUrl);
  MainPage = new MyBookshelf.MainPage();
}
```

Figure 10 shows the code for the DataManager class.

In the SaveAsync method, the Id property is used to detect if an item is new or existing. If it's new, a new GUID is generated and the item is inserted; otherwise, it's updated. In LoadAsync, the UserId property is used to filter the list of items based on the current user. This is possible because of generics and generic constraints, which let both methods work with any object that derives from TableBase.

The ViewModel

A ViewModel class will be responsible for exposing a collection of books, commands that let you work with Book items and the login logic. Add a new file called BookViewModel.cs into a subfolder called ViewModel in the PCL project, then write the code shown in **Figure 11**.

Figure 12 **The UI for the Main Page**

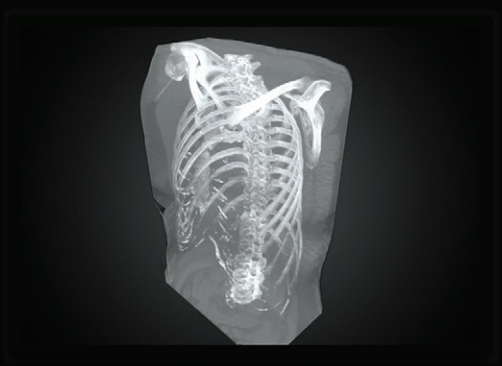
```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  xmlns:local="clr-namespace:MyBookshelf"
  x:Class="MyBookshelf.MainPage">
  <Grid BindingContext="{Binding}">
    <Grid.RowDefinitions>
      <RowDefinition Height="40"/>
      <RowDefinition Height="30"/>
      <RowDefinition />
      <RowDefinition Height="40" />
    </Grid.RowDefinitions>

    <Label Text="My Bookshelf" FontSize="Large" Margin="5"/>

    <ActivityIndicator x:Name="Indicator1" IsRunning="{Binding IsBusy}"
      IsVisible="{Binding IsBusy}" Grid.Row="1"/>

    <ListView HasUnevenRows="True" ItemsSource="{Binding Books}"
      x:Name="BooksListView"
      Grid.Row="2" SelectedItem="{Binding SelectedBook}" >
        <ListView.ItemTemplate>
          <DataTemplate>
            <ViewCell>
              <ViewCell.View>
               <Frame OutlineColor="Blue">
                 <Grid>
                    <Grid.RowDefinitions>
                      <RowDefinition/>
                      <RowDefinition/>
                    </Grid.RowDefinitions>
                    <Entry Placeholder="Title" Text="{Binding Title}"/>
                    <Entry Placeholder="Author" Text="{Binding Author}"
                  Grid.Row="1"/>
                 </Grid>
               </Frame>
              </ViewCell.View>
            </ViewCell>
          </DataTemplate>
        </ListView.ItemTemplate>
    </ListView>

   <StackLayout Grid.Row="3" Orientation="Horizontal"
     BindingContext="{Binding}">
     <Button Text="Load" x:Name="LoadButton" Command="{Binding LoadBooks}"/>
     <Button Text="Save" x:Name="SaveButton" Command="{Binding SaveBooks}"/>
     <Button Text="Add" x:Name="AddButton" Command="{Binding AddNewBook}"/>
   </StackLayout>
  </Grid>
</ContentPage>
```

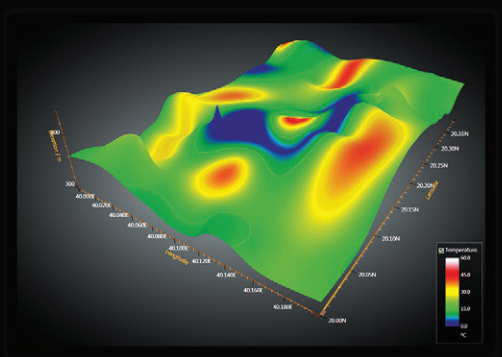
Arction

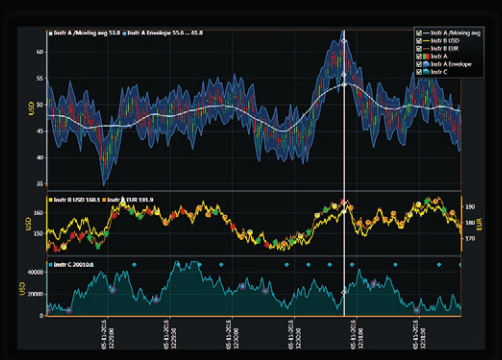
[WPF]
[Windows Forms]
[Free Gauges]
[Data Visualization]
[Volume Rendering]
[3D / 2D Charts] [Maps]

LightningChart®

The fastest and most advanced charting components







Create eye-catching and powerful charting applications for engineering, science and trading

- DirectX GPU-accelerated
- Optimized for real-time monitoring
- Supports gigantic datasets
- Full mouse-interaction
- Outstanding technical support
- Hundreds of code examples

NEW
- Now with Volume Rendering extension
- Flexible licensing options

Get free trial at
LightningChart.com/ms

Notice how the LoginAsync method invokes the platform-specific implementation of the IAuthentication interface via the DependencyService.Get method, passing the active instance of the MobileServiceClient class and the current authentication provider.

The UI

The UI for the sample application is very simple. If a ListView is data-bound to an instance of the ViewModel, an ActivityIndicator will show the progress indicator when the app is busy, and a few buttons are data-bound to commands in the ViewModel. The XAML for the UI is shown in **Figure 12** and goes in the MainPage.xaml file.

In the codebehind, you must assign an instance of the ViewModel to the binding context of the page and you'll need to check that login is necessary every time the user opens the page. Here's the code:

```
private BookViewModel ViewModel { get; set; }
public MainPage()
{
  InitializeComponent();
  this.ViewModel = new BookViewModel();
  this.BindingContext = this.ViewModel;
}

protected async override void OnAppearing()
{
  base.OnAppearing();
  if (this.ViewModel.IsUserAuthenticated == false) await this.ViewModel.LoginAsync();
}
```

LoginAsync is called within the OnAppearing method because this lets you call asynchronous methods at page initialization and every time a page is displayed.

Configuring the App for Analytics and Crash Reporting

With a single line of code, you can configure your Xamarin.Forms app to send usage and crash reports to Visual Studio Mobile Center.

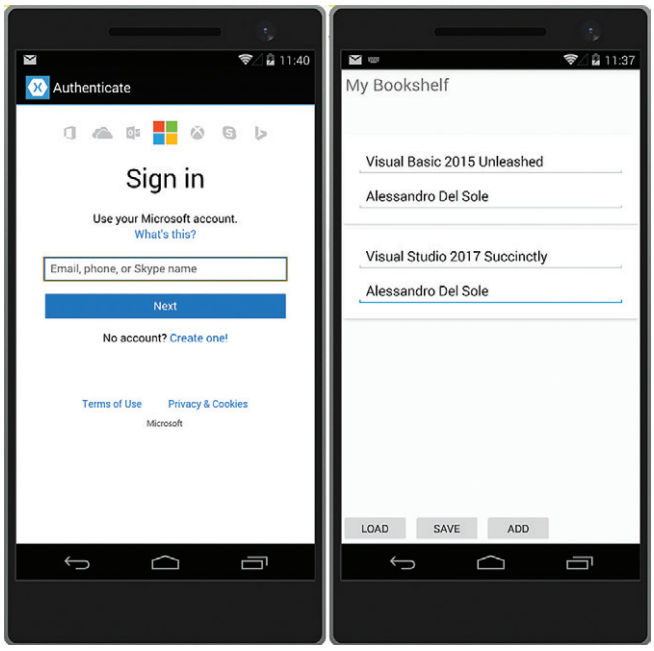


Figure 13 **The Sample App Running on Android**

In the App.xaml.cs, override the OnStart method as follows:

```
// Requires using directives for Microsoft.Azure.Mobile,
// Microsoft.Azure.Mobile.Analytics and Microsoft.Azure.Mobile.Crashes
protected override void OnStart()
{
  MobileCenter.Start("android={Your Android App secret here};" +
    "ios={Your iOS App secret here}",
    typeof(Analytics), typeof(Crashes));
}
```

As you can see, you can enable both the Android and iOS projects by specifying the app secret, which can be found in the Getting Started page of Mobile Center. All the reports can be found in the Analytics and Crashes pages of Mobile Center.

Testing the Application

If you now run the sample app, the first thing you'll see is the login screen for the Microsoft account. Once you've been authenticated, you'll be able to view, edit and save the list of books, as demonstrated in **Figure 13**.

Of course, a similar result will appear on iOS devices. If you try to add and save some items, and then go to the table page in Visual Studio Mobile Center, you'll see a grid showing your records. Further improvements to the code might involve implementing offline sync with the Azure Mobile Client SDK and more specific error handling in case the user isn't authenticated.

Further improvements to the code might involve implementing offline sync with the Azure Mobile Client SDK and more specific error handling in case the user isn't authenticated.

Wrapping Up

Visual Studio Mobile Center dramatically simplifies the way you implement back-end services for your mobile apps—such as authentication and tables—not only with a convenient UI, but also by setting up most of the backing infrastructure on the Azure Mobile Apps service on your behalf. Visit the Mobile Center frequently in order to verify the availability of new features. ■

Alessandro Del Sole *has been a Microsoft MVP since 2008. Awarded MVP of the Year five times, he has authored many books, eBooks, instructional videos and articles about .NET development with Visual Studio. Del Sole works as a senior .NET developer, focusing on .NET and mobile app development, training, and consulting. Follow him on Twitter: @progalex.*

Thanks to the following technical expert for reviewing this article: *Adrian Hall*

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

WASH, DC
JUNE 12-15, 2017
MARRIOTT MARQUIS

INTENSE TRAINING FOR DEVELOPERS, ENGINEERS, PROGRAMMERS, ARCHITECTS AND MORE!

Development Topics include:

- Visual Studio / .NET
- JavaScript / HTML5
- Angular
- Native Mobile & Xamarin
- Software Practices
- Database and Analytics
- ASP.NET Core
- Web API
- ALM / DevOps
- Cloud Computing
- UWP
- Unit Testing

Register by June 12 for Best Savings

Save $200!*

MUST USE DISCOUNT CODE DCEB01

*Some restrictions apply. Discount off of 4 day packages only.

REGISTER NOW

EVENT PARTNER
Microsoft

SUPPORTED BY
Visual Studio msdn magazine Visual Studio MAGAZINE

PRODUCED BY
1105 MEDIA
YOUR GROWTH. OUR BUSINESS.

DC AGENDA AT-A-GLANCE

ALM / DevOps	Cloud Computing	Database and Analytics	Native Client	Software Practices	Visual Studio / .NET Framework	Web Client	Web Server

START TIME	END TIME	Visual Studio Live! Pre-Conference Workshops: Monday, June 12, 2017 *(Separate entry fee required)*			
7:30 AM	9:00 AM	Pre-Conference Workshop Registration - Coffee and Morning Pastries			
9:00 AM	6:00 PM	**M01** Workshop: Distributed Cross-Platform Application Architecture - *Jason Bock & Rockford Lhotka*	**M02** Workshop: Practical ASP.NET DevOps with VSTS or TFS - *Brian Randell*	**M03** Workshop: SQL Server 2016 for Developers - *Andrew Brust & Leonard Lobel*	
6:45 PM	9:00 PM	Dine-A-Round			
START TIME	**END TIME**	**Visual Studio Live! Day 1: Tuesday, June 13, 2017**			
7:00 AM	8:00 AM	Registration - Coffee and Morning Pastries			
8:00 AM	9:00 AM	**KEYNOTE:** Microsoft PowerApps and Flow—Building Apps that Mean Business, with Low to No Code - **Saurabh Pant,** *Principal Product Manager, Customer and Partner Success Team, Microsoft*			
9:15 AM	10:30 AM	**T01** Go Mobile With C#, Visual Studio, and Xamarin - *Kevin Ford*	**T02** Build Better JavaScript Apps with TypeScript - *Brian Noyes*	**T03** What's New in Azure IaaS v2 - *Eric D. Boyd*	**T04** Tactical DevOps with VSTS - *Brian Randell*
10:45 AM	12:00 PM	**T05** Conquer the Network - Making Resilient and Responsive Connected Mobile C# Apps - *Roy Cornelissen*	**T06** Getting Started with Aurelia - *Brian Noyes*	**T07** Cloud Oriented Programming - *Vishwas Lele*	**T08** PowerShell for Developers - *Brian Randell*
12:00 PM	1:30 PM	Lunch - Visit Exhibitors			
1:30 PM	2:45 PM	**T09** Lessons Learned from Real World Xamarin.Forms Projects - *Nick Landry*	**T10** Getting to the Core of .NET Core - *Adam Tuliper*	**T11** Cloud Enable an Existing WPF LOB App - *Robert Green*	**T12** New SQL Server 2016 Security Features for Developers - *Leonard Lobel*
3:00 PM	4:15 PM	**T13** Xamarin vs. Cordova - *Sahil Malik*	**T14** Assembling the Web - A Tour of WebAssembly - *Jason Bock*	**T15** Go Serverless with Azure Functions - *Eric D. Boyd*	**T16** No Schema, No Problem! Introduction to Azure DocumentDB - *Leonard Lobel*
4:15 PM	5:30 PM	Welcome Reception			
START TIME	**END TIME**	**Visual Studio Live! Day 2: Wednesday, June 14, 2017**			
7:30 AM	8:00 AM	Registration - Coffee and Morning Pastries			
8:00 AM	9:15 AM	**W01** To Be Announced	**W02** Creating Reactive Applications With SignalR - *Jason Bock*	**W03** Bots are the New Apps: Building Bots with ASP.NET WebAPI & Language Understanding - *Nick Landry*	**W04** What's New in Visual Studio 2017 - *Robert Green*
9:30 AM	10:45 AM	**W05** Write Once Run Everywhere, Cordova, Electron, and Angular2 - *Sahil Malik*	**W06** Explore Web Development with Microsoft ASP.NET Core 1.0 - *Mark Rosenberg*	**W07** Exploring C# 7 New Features - *Adam Tuliper*	**W08** Get Started with Git and GitHub - *Robert Green*
11:00 AM	12:00 PM	**GENERAL SESSION: TO BE ANNOUNCED - Adam Tuliper,** *Technical Evangelist, Microsoft*			
12:00 PM	1:30 PM	Birds-of-a-Feather Lunch - Visit Exhibitors			
1:30 PM	2:45 PM	**W09** Distributed Architecture: Microservices and Messaging - *Rockford Lhotka*	**W10** Angular 101: Part 1 - *Deborah Kurata*	**W11** A Busy Developer's Intro to Windows Containers - *Vishwas Lele*	**W12** Continuous Delivery 3.0, The Next Step - *Marcel de Vries*
3:00 PM	4:15 PM	**W13** Agile Failures: Stories From The Trenches - *Philip Japikse*	**W14** Angular 101: Part 2 - *Deborah Kurata*	**W15** Use Docker to Develop, Build and Deploy Applications, a Primer - *Mark Rosenberg*	**W16** Mobile DevOps Demystified with Xamarin, VSTS and HockeyApp - *Roy Cornelissen*
4:30 PM	5:45 PM	**W17** Top 10 Ways to Go from Good to Great Scrum Master - *Benjamin Day*	**W18** SASS and CSS for Developers - *Robert Boedigheimer*	**W19** Microservices with Azure Container Service & Service Fabric - *Vishwas Lele*	**W20** A/B Testing, Canary Releases and Dark Launching, Implementing Continuous Delivery on Azure - *Marcel de Vries*
6:45 PM	10:30 PM	Visual Studio Live! Monuments by Moonlight Tour			
START TIME	**END TIME**	**Visual Studio Live! Day 3: Thursday, June 15, 2017**			
7:30 AM	8:00 AM	Registration - Coffee and Morning Pastries			
8:00 AM	9:15 AM	**TH01** Let's Write a Windows 10 App: A Basic Introduction to Universal Apps - *Billy Hollis*	**TH02** JavaScript for the C# (and Java) Developer - *Philip Japikse*	**TH03** Entity Framework Core for Enterprise Applications - *Benjamin Day*	**TH04** Overcoming the Challenges of Mobile Development in the Enterprise - *Roy Cornelissen*
9:30 AM	10:45 AM	**TH05** Implementing the MvvM Pattern in Your Xamarin Apps - *Kevin Ford*	**TH06** Integrating AngularJS & ASP.NET MVC - *Miguel Castro*	**TH07** Power BI: Analytics for Desktop, Mobile and Cloud - *Andrew Brust*	**TH08** Exploring Microservices in a Microsoft Landscape - *Marcel de Vries*
11:00 AM	12:15 PM	**TH09** Building Cross-Platform Business Apps with CSLA. NET - *Rockford Lhotka*	**TH10** Debugging Your Website with Fiddler and Chrome Developer Tools - *Robert Boedigheimer*	**TH11** Big Data with Hadoop, Spark and Azure HDInsight - *Andrew Brust*	**TH12** End-to-End Dependency Injection & Testable Code - *Miguel Castro*
12:15 PM	1:15 PM	Lunch			
1:15 PM	2:30 PM	**TH13** Creating Great Looking Android Applications Using Material Design - *Kevin Ford*	**TH14** Building Single Page Web Applications Using Aurelia.js and the MVVM Pattern - *Ben Hoelting*	**TH15** Using Cognitive Services in Business Applications - *Michael Washington*	**TH16** Extend and Customize the Visual Studio Environment - *Walt Ritscher*
2:45 PM	4:00 PM	**TH17** Take the Tests: Can You Evaluate Good and Bad Designs? - *Billy Hollis*	**TH18** Tools for Modern Web Development - *Ben Hoelting*	**TH19** Introduction to Azure Machine Learning Studio (for the Non-Data Scientist) - *Michael Washington*	**TH20** Windows Package Management with NuGet and Chocolatey - *Walt Ritscher*

Speakers and sessions subject to change

vslive.com/dcmsdn

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

CHICAGO

SEPTEMBER 18 - 21, 2017

DOWNTOWN MARRIOTT

GOLD SPONSOR

Suncfusion

SUPPORTED BY

Visual Studio msdn magazine Visual Studio MAGAZINE

PRODUCED BY

1105 MEDIA

INTENSE TRAINING FOR DEVELOPERS, ENGINEERS, PROGRAMMERS, ARCHITECTS AND MORE!

Development Topics Include:

- Visual Studio / .NET Framework
- JavaScript / HTML5 Client
- Native Client
- Software Practices
- Database and Analytics
- Angular JS
- ASP.NET / Web Server
- Agile
- ALM / DevOps
- Cloud Computing
- Windows Client
- Xamarin

REGISTER NOW

Register Today and Save $300!

Use promo code VSLCHI2

ROCK YOUR CODE TOUR 2017

"I loved that this conference isn't a sales pitch. It's actual developers giving the sessions - I felt the classes were really geared towards me!"
- Jon Plater, Kantar Retail

"The speakers were clearly experts. Nice breadth of topics. Great hotel; great location."
- Fred Kauber, Tranzact

vslive.com/chicagomsdn

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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

Make C# More Dynamic with Hyperlambda

Thomas Hansen

Traditionally, there have been just two types of programming languages: statically compiled languages and dynamically interpreted languages. What if there's actually a third type of programming language, one that fits in neither of these categories and has the potential to make all other languages seem hopelessly rigid in comparison?

Hyperlambda is that language—a Turing-complete execution environment built entirely upon Active Events (see my previous article at msdn.com/magazine/mt795187). Hyperlambda is neither an interpreted language nor a compiled language. It doesn't even have syntax. Hyperlambda is a non-programming language, perhaps even an anti-programming language. Technically, it's not even a programming language at all. At its core, Hyperlambda is simply a relational file format that allows you to declare tree structures. Arguably, it has more in common with HTML and XML than with C# and JavaScript. It turns out, though, that tree structures just happen to be able to describe what we often refer to as execution trees.

This article discusses:

- Execution trees and lambda objects
- Using and extending Hyperlambda
- Modifying the execution tree
- Several use cases
- Creating "Hello World" in Hyperlambda

Technologies discussed:

C#, Hyperlambda, Active Events

Execution trees are what your CPU actually executes. All programming instructions can basically be understood as branches of a tree, which your CPU moves through as it evaluates your code. For instance, an if/else block creates two possible branches that are mutually exclusive, and your CPU chooses only one, depending on the results of the condition in the if statement. Essentially, what your computer does as it executes code is to move through an execution tree, manipulating its memory in the process. This implies that if you can describe an execution tree and manipulate memory, you can also describe anything you've previously described using C# and JavaScript.

Everything Is a Tree

Just like everything is a "list" in Lisp, everything is a "tree" in Hyperlambda. Here's an example of some Hyperlambda pseudocode that creates an execution tree after being parsed:

```
if:condition
  do-x:argument1
else
  do-y:argument1
```

Parsing this code results in a lambda object. This lambda object, or execution tree, can be evaluated by invoking the eval Active Event in Phosphorus Five (P5). You can think of an Active Event basically as a "function object" that allows you to invoke a method created in, for instance, C#, with a graph object or a tree object as its only argument. P5 implements the Active Event design pattern.

Once such a lambda object has been created, the eval Active Event in P5 sequentially executes the root node's children nodes, invoking these as Active Events, from top to bottom. The lambda object created in the previous example contains four nodes, two of which are root nodes. Hence, eval will execute the "if" before it executes the "else."

Therefore, a lambda object becomes the "DOM structure" Hyperlambda creates. If I wanted to, I could've used XML instead of Hyperlambda to describe this tree structure. It just so happens that Hyperlambda is the superior method for describing such tree structures. It often helps to think of Hyperlambda as HTML or XML, and lambda objects as the resulting DOM structure.

Hyperlambda has an extremely simple syntax. All nodes can have a name, a value and children nodes. Here are all its major control structures:

- A colon separates names and values of nodes.
- Two spaces opens up the children collection of nodes.

A node's value can optionally have a type declaration, squeezed in between the name and its value. The type declaration is optional, though, and rarely used. String is the default type declaration in Hyperlambda.

What Makes Hyperlambda Unique

A static language, such as C++ or C#, compiles its code down to some machine-executable thing up front, before it starts executing the result. A dynamic language, such as JavaScript, normally interprets the code on the fly, as it's executing the code. Hyperlambda, in contrast, does neither. Instead, you declare an execution tree, which is parsed by P5, producing lambda objects that are then evaluated using the eval Active Event in P5. The lambda objects you create don't even consider syntax, and you can use any file format capable of declaring relational tree structures to declare your lambda objects, which means that Hyperlambda ends up being no more of a programming language than HTML or XML are. However, because it's still a Turing-complete execution environment, it allows you to do everything you can do with any other programming language. Arguably, I could've used HTML to declare my lambda objects, at which point HTML would've become a Turing-complete programming language.

Branching Your Tree

Every keyword in Hyperlambda is simply an Active Event. Invoking these Active Events often results in recursively invoking eval, according to some sort of condition. This makes Hyperlambda extremely extendible. Creating a new keyword literally means creating five lines of code. Here's an example in C#:

```
[ActiveEvent (Name = "what-is-the-meaning-of-life?")]
private static void foo (ApplicationContext context, ActiveEventArgs e)
{
  e.Args.Value = 42;
}
```

In fact, both the if and else keywords are created as Active Events like this.

You don't have to use C# to create Active Events or keywords. You can also create your own keywords in Hyperlambda:

```
create-event:what-is-the-meaning-of-life?
  return:int:42
```

The observant reader might realize at this point that although Hyperlambda is definitely a very high-level abstraction, at some philosophical level it's also a very low-level abstraction—arguably far lower than any other existing programming language, because what you're modifying isn't code but the actual execution tree. This characteristic of Hyperlambda has huge benefits, the same way the DOM model has huge benefits for HTML and XML. First of all, it enables extremely flexible and self-modifiable code. This means that the separation between what is normally thought of as creating code and executing code becomes much thinner—allowing for code that does a little bit of both. Evaluating a lambda object with some input data might easily create new logic and additional branches in your execution tree. For example, if you don't like a particular if/else block for some reason, you can dynamically remove it, even after you've started evaluating the lambda object containing it. In fact, your execution tree might look completely different, depending on the input data you supply to your lambda objects.

> Among Hyperlambda's other odd traits is the fact that it has no variables.

Just as you can use the DOM model to modify parts of your HTML page in JavaScript, you can use the "lambda object model" to modify parts of your lambda objects during execution. Literally, your execution tree becomes a dynamic entity that can change during execution.

Transforming back and forth between the Hyperlambda file format, which is serializable and persistent in nature, and lambda objects, which are execution trees, is as easy as invoking an Active Event.

Finding Your Branch

Among Hyperlambda's other odd traits is the fact that it has no variables. I don't mean no variables in the F# sense, where everything is immutable. I literally mean *Hyperlambda has no variables!* At this point, of course, you're skeptical. After all, being able to change the state of your execution tree and memory is key to creating a Turing-complete execution environment. However, if you can change the state of your execution tree, you actually don't need what we traditionally refer to as variables.

In Hyperlambda, everything can change. Changing anything in Hyperlambda simply implies referencing the nodes you wish to change and supplying a source for your change operation. So, the first thing that's needed is a way to reference nodes in your tree, and this is done with lambda expressions, like so:

```
_data:Thomas
if:x:/@_data?value
  =:Thomas
  say-hello:Yo boss
else
  say-hello:Yo stranger
```

This code shows one lambda expression, which refers to the _data node's value and is supplied as an argument to the if Active Event invocation. This if invocation compares the result of the lambda expression to the static value "Thomas," and if the value of "_data"

equals "Thomas," it will evaluate the lambda object inside the if node. Otherwise, it will evaluate the lambda object in the else node.

A lambda expression can reference any nodes in your tree, and results in a subtree, based upon the original tree. The relationship between a lambda expression and a tree structure is somewhat analogous to the relationship between SQL and a table. Lambda expressions are easily visualized by those who have created XPath expressions. You can compare them to LINQ or "pre-compiled enumerators," if you wish. The latter is actually how they're implemented. If you believe that building LINQ queries dynamically adds value, purely logically you must believe that lambda expressions might possess value.

Pruning Your Tree

At this point, all that's needed to have a Turing-complete execution environment is the ability to change parts of the tree. Here, I'm doing so with the "set" Active Event:

```
_input:Thomas
_output
if:x:/@_input?value
  =:Thomas
  set:x:/@_output?value
    src:Yo boss
else
  set:x:/@_output?value
    src:Yo stranger
```

The set Active Event basically works like the assignment operator in C#. At this point, we have a Turing-complete "non-programming language" that allows you to do everything you're used to doing in a traditional programming language—without ever actually having created a programming language or even having considered syntax. Instead, you modify the execution tree directly as the lambda object is evaluated.

> One lambda object can transform any other lambda object the same way an XSLT file can transform an XML file.

There are more Active Events you can use to modify your tree. For instance, "add" appends a bunch of nodes to the result of a lambda expression, leading to one or more additional nodes, while "insert-before" and "insert-after" works similarly to add, except that they inject one or more nodes before or after a bunch of nodes. To remove nodes, values, or names is as easy as not supplying a source to set, at which point it will simply remove the node, its value, or its name, depending on what type declaration you use for your expression.

Using constructs like these makes C# a super-dynamic programming language by giving the programmer the tools needed to be able to change the execution tree directly. Arguably, these traits of Hyperlambda make it the most dynamic programming environment on the planet by far. Even super-dynamic languages, such as Lisp and JavaScript, seem hopelessly static and rigid in comparison to Hyperlambda.

At this point, you might realize that in Hyperlambda, there's absolutely no difference between data and logic—logic *is* data and there are no semantic differences at all between the two. This implies that you can just as easily change your logic as you can your data, which, in turn, results in a super-dynamic execution model where your execution tree might completely change during its execution. Just how cool this is can be illustrated with a few highly instructive use cases.

Nurturing Your Tree

Use case 1: Imagine you have 100 records in your database. Each record contains some Hyperlambda. If you wish, you could select all these records, transform them into lambda objects and append them to a destination lambda object, evaluating their combined result afterward. You could even create a new single function object out of these records, persist them to a file on disc and use this file in the same way you'd invoke a function.

Use case 2: If you want, you can evaluate only half of a function, removing the parts of the function you don't want to execute. As far as I know, no other programming languages on the planet have this capability. It happens to be an extremely useful feature.

Use case 3: You can, if you need to, extract half of the lambda objects from five different functions, creating a single new function derived from the "better parts" of your original functions.

How is this possible? One lambda object can transform any other lambda object the same way an XSLT file can transform an XML file. If you don't want to execute any while loops in your system, for whatever reason, you can easily create a lambda object that transforms all of your while loops and turns them into for-each loops and have it run in the background as an automated process.

In case you hadn't realized, this is how HTML works in relationship to its DOM model. In Hyperlambda, modifying lambda objects is as easy as inserting, modifying or deleting an HTML element in your DOM with JavaScript. If you think this capability adds value to HTML, purely logically you must agree that this could potentially add value to "logic." Hyperlambda opens up a completely new dimension with regard to software development.

Use case 4: Consider the CMS in System42 (bit.ly/2pwMOY9). This CMS allows you to create lambda pages, which instead of simply displaying static HTML, are 100 percent dynamic and interactive in nature. "Big deal," some might argue, claiming that PHP can at least in theory do the same thing. However, these lambda pages are also reusable components. The dynamic nature of Hyperlambda means you can, for instance, check for the existence of a "parent-widget" argument during the execution of your lambda page and, if it exists, instead of creating a page, you can inject the same page as a user control into another widget on your page.

In fact, you don't even have to modify your original page; you can instead transform its lambda object before you evaluate its "create-widget" invocation. To categorize things as absolutes the way you're used to in traditional programming languages is completely meaningless in Hyperlambda.

The result is that every single lambda object you create in Hyperlambda is also a potentially reusable component you can inject into other parts of your system. This implies that you can incrementally

GROUPDOCS
Document Manipulation APIs

Manipulating Files?

APIs to view, export, annotate, compare, sign, automate and search documents in your applications.

GroupDocs.Total

GroupDocs.**Viewer**
GroupDocs.**Annotation**
GroupDocs.**Conversion**
GroupDocs.**Comparison**
GroupDocs.**Signature**
GroupDocs.**Assembly**
GroupDocs.**Metadata**
GroupDocs.**Search**
GroupDocs.**Text**

Try for Free

.NET Libraries

Java Libraries

Cloud APIs

Contact Us:
US: +1 903 306 1676
EU: +44 141 628 8900
AU: +61 2 8006 6987
sales@asposeptyltd.com

Visit us at www.groupdocs.com

GROUPDOCS
Document Manipulation APIs

create increasingly complex systems, building on systems you've previously created. In a traditional programming language, you'd have to choose between building a system or building a reusable component. In Hyperlambda, such mutually exclusive choices are meaningless, and everything tends to become a little bit of everything else. Hyperlambda's reusability features completely dwarf everything you've ever seen before in your programming life.

Use case 5: In Hyperlambda, you can actually recursively execute a folder on disc, passing in arguments to the execution, eliminating everything that used to constrain you in traditional programming. Or you can invoke a file exactly as you'd invoke a function, having the invocation of your file return multiple values to its caller. You can also easily invoke the body of an HTTP POST request transmitted to your server, effectively replacing every single Web service endpoint you've ever created with five to 15 lines of code. All of these are things I do almost every single day as I use Hyperlambda.

> The result is that every single lambda object you create in Hyperlambda is also a potentially reusable component you can inject into other parts of your system.

Use case 6: In Hyperlambda, even thinking about a Web page as purely a Web page is meaningless because just by replacing a single assembly in your application pool you can easily create a Windows Forms application using the exact same code. In Hyperlambda, reusing the same code base for your Web apps and Windows Forms apps is actually ridiculously easy—even the GUI parts of your code.

When I show Hyperlambda to other software developers, some tend to dismiss it because Hyperlambda seems to be much more verbose than traditional programming languages. For instance, the previous code example could easily have been described with a single line of code in C#, while in Hyperlambda it took nine lines. For simple tasks, Hyperlambda is admittedly sometimes more verbose. However, Hyperlambda features one trait that mitigates this issue—the capability to transform lambda objects, which gives you code that's much less verbose in the long run. So, basically, you pay a small price up front—more verbose code in the short run—to end up with far less verbose code in the long run. This allows you to deliver far more functionality with far less effort. Hyperlambda enables exponential software productivity growth. Let that last sentence sink in a little bit, please.

Hyperlambda's Hello World

The following is the "Hello World" application created in Hyperlambda. If you paste this code into, for instance, a lambda page in the System42 CMS, you'll create a button that, when clicked, changes its inner value to "I was clicked." Download P5, and try it out by starting the Apps/CMS app. Then create a lambda page and paste in this code:

```
create-widget:foo-widget
  element:button
  class:btn btn-default
  innerValue:Click me!
  onclick
    set-widget-property:foo-widget
      innerValue:I was clicked
```

Seven lines of Hyperlambda arguably replace potentially hundreds of lines of code in JavaScript, C#, or HTML, depending what framework, library, or language you use to create something similar. And if you read through the code carefully, you can probably guess what HTML markup it will produce. Hyperlambda tends to be closer to natural language than other programming languages. Linguistically, it resembles the way humans naturally convey information to other humans more closely than, for example, C# or JavaScript.

As an additional bonus, if you create your own Windows Forms assembly, using an alternative implementation of the previously mentioned create-widget Active Event, you can use the exact same code to create a Windows Forms application.

Is Hyperlambda a Silver Bullet?

I've been accused of advertising silver bullets for as long as I've advocated Active Events. I admit that some of my claims probably sound like magic. And, yes, Hyperlambda creates more verbose code initially, as well as adding more overhead to the execution of your programming instructions, due to its internal implementation semantics. I also occasionally find bugs, and I do refactor my things, resulting hopefully in better code. Obviously, Hyperlambda is not perfect.

However, Hyperlambda's biggest problem is actually in your head. Hyperlambda completely invalidates everything you've been taught for the last 50 years as best practices for software development. And it does it with style, putting its money where its mouth is!

At the end of the day, though, the answer to whether Hyperlambda is a silver bullet is not one I can give you. This is something you'll have to figure out for yourself. It may help to check out an example of a GMail clone, Sephia Five, built entirely in Hyperlambda (github.com/polterguy/sephia-five). Sephia Five features PGP cryptography and 100 perfect perfect virus and malware protection, and it works on all devices capable of displaying HTML. Sephia Five is also several orders of magnitudes more efficient in bandwidth usage than GMail. And its initial release was literally built in five days, from scratch. All of these claims can be checked, verified, and mathematically reproduced and proven!

To learn more about Hyperlambda, feel free to read the guide at github.com/polterguy/phosphorusfive-dox. And you can download Hyperlambda along with Phosphorus Five at github.com/polterguy/phosphorusfive. ■

Thomas Hansen *has been creating software since he was 8 years old, when he started writing code using the Oric-1 computer in 1982. Occasionally he creates code that does more good than harm. His passions include the Web, AJAX, Agile methodologies and software architecture.*

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

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

ANAHEIM, CA
OCT 16-19, 2017
HYATT REGENCY
A Disneyland® Good Neighbor Hotel

California CODIN'

INTENSE TRAINING FOR DEVELOPERS, ENGINEERS, PROGRAMMERS, ARCHITECTS AND MORE!

Development Topics include:

- Visual Studio / .NET
- JavaScript / HTML5
- Angular
- Native Mobile & Xamarin
- Software Practices
- Database and Analytics
- ASP.NET Core
- Web API
- ALM / DevOps
- Cloud Computing
- UWP
- Unit Testing

Register Now and Save $300!*

Use promo code VSLAN1 *SAVINGS BASED ON 4-DAY PACKAGES ONLY.

REGISTER NOW

vslive.com/anaheimmsdn

EVENT PARTNER Microsoft

SUPPORTED BY Visual Studio msdn magazine Visual Studio MAGAZINE

PRODUCED BY 1105 MEDIA YOUR GROWTH. OUR BUSINESS.

JAMES MCCAFFREY

Restricted Boltzmann Machines Using C#

A restricted Boltzmann machine (RBM) is a fascinating software component that has some similarities to a basic neural network. An RBM has two sets of nodes—visible and hidden. Each set of nodes can act as either inputs or outputs relative to the other set. Each node has a value of zero or one and these values are calculated using probability rather than deterministically.

Each visible-layer node is conceptually connected to each hidden-layer node with a numeric constant called a weight, typically a value between -10.0 and +10.0. Each node has an associated numeric constant called a bias. The best way to start to understand what an RBM is is by way of a diagram. Take a look at **Figure 1**.

In **Figure 1**, the visible nodes are acting as the inputs. There are six visible (input) nodes and three hidden (output) nodes. The values of the visible nodes are (1, 1, 0, 0, 0, 0) and the computed values of the hidden nodes are (1, 1, 0). There are 6 * 3 = 18 weights connecting the nodes. Notice that there are no visible-to-visible or hidden-to-hidden weights. This restriction is why the word "restricted" is part of the RBM name.

Each of the red weight arrows in **Figure 1** points in both directions, indicating that information can flow in either direction. If nodes are zero-base indexed, then the weight from visible[0] to hidden[0] is 2.78, the weight from visible[5] to hidden[2] is 0.10 and so on. The bias values are the small green arrows pointing into each node so the bias for visible[0] is -0.62 and the bias for hidden[0] is +1.25 and so on. The p value inside each node is the probability that the node takes a value of one. So, hidden[0] has p = 0.995, which means that its calculated value will almost certainly be one, and in fact it is, but because RBMs are probabilistic, the value of hidden[0] could have been zero.

You probably have many questions right about now, such as where the weights and bias values come from, but bear with me—you'll see how all the parts of the puzzle fit together shortly. In the sections that follow, I'll describe the RBM input-output mechanism, explain where the weights and bias values come from, present a demo program that corresponds to **Figure 1**, and give an example of how RBMs can be used.

This article assumes you have intermediate or better programming skills, but doesn't assume you know anything about RBMs. The demo program is coded using C#, but you should have no trouble refactoring the code to another language such as Python or JavaScript if you wish. The demo program is too long to present in its entirety, but the complete source code is available in the file download that accompanies this article. All error checking was removed to keep the main ideas as clear as possible

Code download available at msdn.com/magazine/0617magcode.

The RBM Input-Output Mechanism

The RBM input-output mechanism is (somewhat surprisingly) relatively simple and is best explained by an example. Suppose, as in **Figure 1**, the visible nodes act as inputs and have values (1, 1, 0, 0, 0, 0). The value of hidden node[0] is computed as follows: The six weights from the visible nodes into hidden[0] are (2.78, 1.32, 0.80, 2.23, -4.27, -2.22) and the bias value for hidden[0] is 1.25.

The p value for hidden[0] is the logistic sigmoid value of the sum of the products of input values multiplied by their associated weights, plus the target node bias. Put another way, multiply each input node value by the weight pointing from the node into the target node, add those products up, add in the target node bias value and then take the logistic sigmoid of that sum:

```
p[0] = logsig( (1 * 2.78) + (1 * 1.32) + (0 * 0.80) +
               (0 * 2.23) + (0 * -4.27) + (0 * -2.22) + 1.25 )
     = logsig( 2.78 + 1.32 + 1.25 )
     = logsig( 5.36 )
     = 0.995
```

The logistic sigmoid function, which appears in many machine learning algorithms, is defined as:

$$logsig(x) = 1.0 / (1.0 + exp(-x))$$

where the exp function is defined as:

$$exp(x) = e^x$$

where e (Euler's number) is approximately 2.71828.

So, at this point, the p value for hidden[0] has been calculated as 0.995. To calculate the final zero or one value for the hidden[0] node you'd use a random number generator to produce a pseudo-random

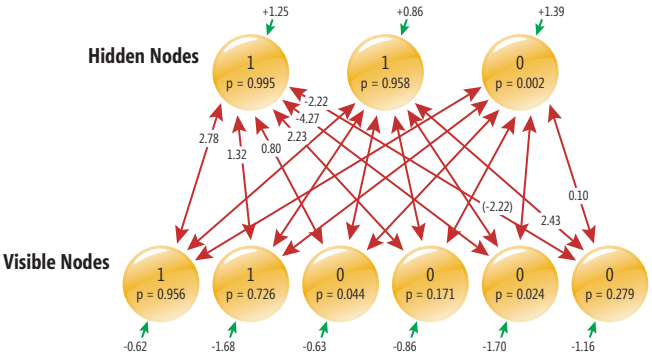


Figure 1 **An Example of a Restricted Boltzmann Machine**

value between 0.0 and 1.0. If the random number is less than 0.995 (which it will be 99.5 percent of the time), the node value is set to one; otherwise (0.05 percent of the time), it's set to zero.

The other hidden nodes would be computed in the same way. And if the hidden nodes were acting as inputs, the values of the visible nodes would be calculated as output values in the same way.

Determining the Weights and Bias Values

Determining a set of RBM output values for a given set of input values is easy, but from where do the weights and bias values come? Unlike neural networks, which require a set of training data with known input values and known, correct, output values, RBMs can essentially train themselves, so to speak, using only a set of values for the visible nodes. Interesting! Suppose you have a set of 12 data items, like so:

```
(1, 1, 0, 0, 0, 0) // A
(0, 0, 1, 1, 0, 0) // B
(0, 0, 0, 0, 1, 1) // C

(1, 1, 0, 0, 0, 1) // noisy A
(0, 0, 1, 1, 0, 0) // B
(0, 0, 0, 0, 1, 1) // C

(1, 0, 0, 0, 0, 0) // weak A
(0, 0, 1, 0, 0, 0) // weak B
(0, 0, 0, 0, 1, 0) // weak C

(1, 1, 0, 1, 0, 0) // noisy A
(1, 0, 1, 1, 0, 0) // noisy B
(0, 0, 1, 0, 1, 1) // noisy C
```

Because RBM visible node values are zero and one, you can think of them as individual binary features (such as "like" and "don't like") or as binary-encoded integers. Suppose each of the 12 data items represents a person's like or don't like opinion for six films: "Alien," "Inception," "Spy," "EuroTrip," "Gladiator," "Spartacus." The first two films are science fiction. The next two films are comedy (well, depending on your sense of humor) and the last two films are history (sort of).

The first person likes "Alien" and "Inception," but doesn't like the other four films. If you look at the data, you can imagine that there are three types of people. Type "A" people like only science fiction films. Type "B" like only comedy films and type "C" like only history films. Notice that there's some noise in the data, and there are weak and noisy versions of each person type.

The number of visible nodes in an RBM is determined by the number of dimensions of the input data—six in this example. The number of hidden nodes is a free parameter that you must choose. Suppose you set the number of hidden nodes to three. Because each RBM node value can be zero or one, with three hidden nodes there are a total of eight people types that can be detected: (0, 0, 0), (0, 0, 1), . . . (1, 1, 1).

There are several ways to train an RBM. The most common algorithm is called CD-1, which stands for contrastive divergence, single-step. The algorithm is very clever and isn't at all obvious. The CD-1 training algorithm is presented in high-level pseudo-code in **Figure 2**.

The goal of training is to find a set of weights and bias values so that when the RBM is fed a set of input values in the visible nodes and generates a set of output nodes in the hidden nodes, then when the hidden nodes are used as inputs, the original visible nodes values will (usually) be regenerated. The only way I was able to understand CD-1 was by walking through a few concrete examples.

> Determining a set of RBM output values for a given set of input values is easy, but where do the weights and bias values come from?

Figure 2 **The CD-1 Training Algorithm**

```
(v represents the visible nodes)
(h represents the hidden nodes)
(lr is a small learning rate value)

loop n times
  for each data item
    compute h from v
    let posGrad = outer product(v, h)
    compute v' from h
    compute h' from v'
    let negGrad = outer product(v', h')
    let delta W = lr * (posGrad - negGrad)
    let delta vb = lr * (v - v')
    let delta hb = lr * (h - h')
  end for
end loop
```

Suppose the learning rate is set to 0.01 and that at some point the current training input item is (0, 0, 1, 1, 0, 0) and the 18 weights are:

```
0.01  0.02  0.03
0.04  0.05  0.06
0.07  0.08  0.09
0.10  0.11  0.12
0.13  0.14  0.15
0.16  0.17  0.18
```

The row index, 0 to 5, represents the index of a visible node and the column index, 0 to 2, represents the index of a hidden node. So the weight from visible[0] to hidden[2] is 0.03.

The first step in CD-1 is to compute the h values from the v values using the probabilistic mechanism described in the previous section. Suppose h turns out to be (0, 1, 0). Next, the positive gradient is computed as the outer product of v and h:

```
0  0  0
0  0  0
0  1  0
0  1  0
0  0  0
0  0  0
```

The outer product isn't very common in machine learning algorithms (or any other algorithms for that matter), so it's quite possible you haven't seen it before. The Wikipedia article on the topic gives a pretty good explanation. Notice that the shape of the positive gradient matrix will be the same as the shape of the weight matrix.

Next, the h values are used as inputs, along with the current weights, to produce new output values v'. Suppose v' turns out to be (0, 1, 1, 1, 0, 0). Next, v' is used as the input to compute a new h'. Suppose h' is (0, 0, 1).

The negative gradient is the outer product of v' and h' and so is:

```
0  0  0
0  0  1
0  0  1
0  0  1
0  0  0
0  0  0
```

The result of posGrad - negGrad is:

```
0   0   0
0   0  -1
0  +1  -1
0  +1  -1
0   0   0
0   0   0
```

If you review the algorithm carefully, you'll see that cell values in the delta gradient matrix can only be one of three values: 0, +1 or -1. Delta gradient values of +1 correspond to weights that should be increased slightly. Values of -1 correspond to weights that should be decreased slightly. Clever! The amount of increase or decrease is set by a learning rate value. So the weight from visible[1] to hidden[2] would be decreased by 0.01 and the weight from visible[2] to hidden[1] would be increased by 0.01. A small learning rate value makes training take longer, but a large learning rate can skip over good weight values.

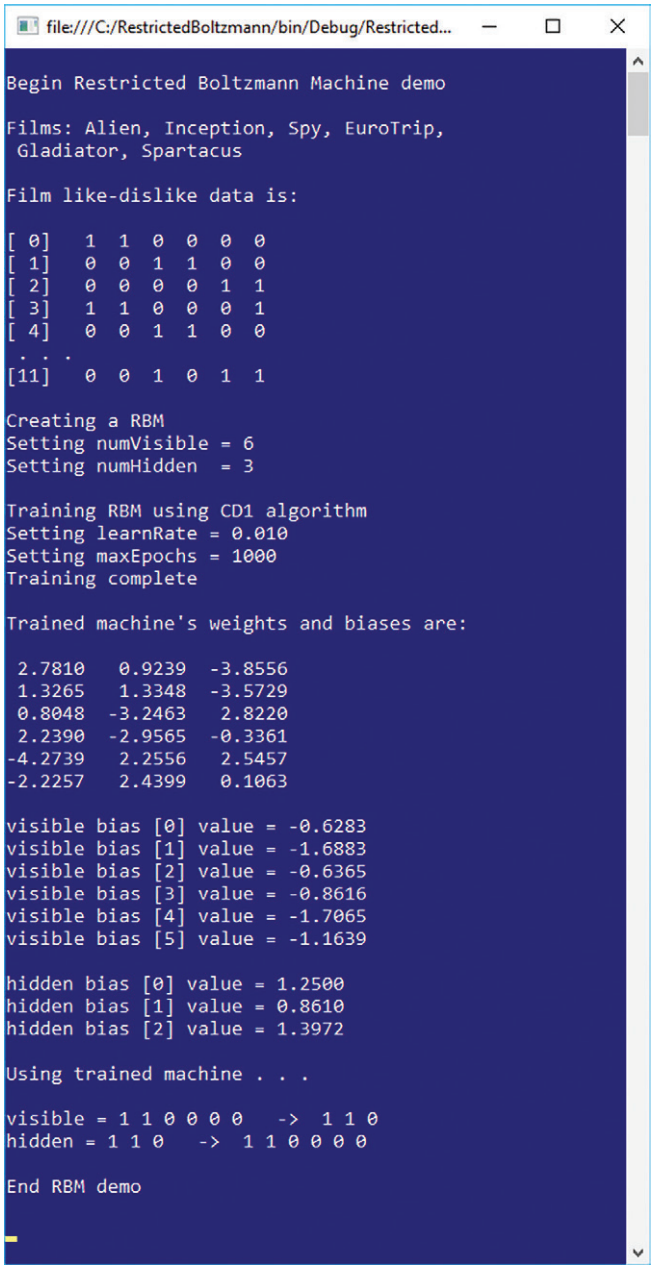


Figure 3 **Demo of a Restricted Boltzmann Machine**

So, how many iterations of training should be performed? In general, setting the number of training iterations and choosing a value of the learning rate are matters of trial and error. In the demo program that accompanies this article, I used a learning rate of 0.01 and a maximum number of iterations set to 1,000. After training, I got the weights and bias values shown in **Figure 1**.

Interpreting a Restricted Boltzmann Machine

OK, so it's possible to take a set of data where each value is zero or one, then set a number of hidden nodes, and get some weights and bias values. What's the point?

One way to think of an RBM is as a kind of compression machine. For the example film preference data, if you feed a type A person as input (1, 1, 0, 0, 0, 0), you'll usually get (1, 1, 0) as output. If you feed (1, 1, 0) as input to the hidden nodes, you almost always get (1, 1, 0, 0, 0, 0) as output in the visible nodes. In other words, (1, 1, 0, 0, 0, 0) and slight variations are mapped to (1, 1, 0). This behavior is closely related to, but not quite the same as, factor analysis in classical statistics.

Take a look at the demo program in **Figure 3**. The demo corresponds to the film like-dislike example. The demo creates a 6-3 RBM and trains it using the 12 data items presented in the previous section. The hardcoded data is set up like so:

```
int[][] trainData = new int[12][];
trainData[0] = new int[] { 1, 1, 0, 0, 0, 0 };
trainData[1] = new int[] { 0, 0, 1, 1, 0, 0 };
...
trainData[11] = new int[] { 0, 0, 1, 0, 1, 1 };
```

In most situations you'd read data from a text file using a helper method. The demo RBM is created and trained like this:

```
int numVisible = 6;
int numHidden = 3;
Machine rbm = new Machine(numVisible, numHidden);
double learnRate = 0.01;
int maxEpochs = 1000;
rbm.Train(trainData, learnRate, maxEpochs);
```

A small learning rate value makes training take longer, but a large learning rate can skip over good weight values.

The choice of setting the number of hidden nodes to three was arbitrary and the values for learnRate and maxEpochs were determined by trial and error. After training, the RBM is exercised like this:

```
int[] visibles = new int[] { 1, 1, 0, 0, 0, 0 };
int[] computedHidden = rbm.HiddenFromVis(visibles);
Console.Write("visible = ");
ShowVector(visibles, false);
Console.Write("  ->  ");
ShowVector(computedHidden, true);
```

If you experiment with the code, you'll notice that the computed hidden values are almost always one of three patterns. Person type A (or weak or noisy version) almost always generates (1, 1, 0). Type B generates (1, 0, 1). And type C generates (0, 1, 1). And if you feed the three patterns as inputs, you'll almost always get the three

Extreme Performance Linear Scalability

NCache™

Distributed Cache

- In-Memory App Data Caching
- ASP.NET Sessions & View State
- Runtime Data Sharing

NosDB

NoSQL Database

- Schema-Free JSON Documents
- Multiple Shards & Data Replication
- Powerful SQL, LINQ, ADO.NET

100% Native .NET

Open Source

sales@alachisoft.com

alachisoft.com

US: +1 (925) 236 3830

primary input patterns back. The point is the RBM has deduced that the data can be placed into one of three buckets. The specific bit patterns aren't important.

Yet another interpretation of this behavior is that an RBM acts as an auto-encoder. And it's also possible to chain several RBMs together to create a prediction system called a deep belief network (DBN). In fact, this is arguably the most common use of RBMs.

Implementing a Restricted Boltzmann Machine

Once you understand how RBMs work, they're actually quite simple. But coding a demo program is a bit more complex than you might expect. There are many design possibilities for an RBM. Take a look at the demo run in **Figure 3**.

> Once you understand how RBMs work, they're actually quite simple. But coding a demo program is a bit more complex than you might expect.

The demo illustrates the film preference example from the previous sections of this article so there are six visible nodes. The demo program defines a Machine object with 10 member fields. The first six fields in the class definition are:

```
public class Machine
{
  public Random rnd;
  public int numVisible;
  public int numHidden;
  public int[] visValues;
  public double[] visProbs;
  public double[] visBiases;
...
```

All fields are declared with public scope for simplicity. The Random object is used when converting a node probability to a concrete zero or one value. Variables numVisible and numHidden (OK, OK, I know they're objects) hold the number of hidden and visible nodes. Integer array visValues holds the zero or one values of the visible nodes. Note that you can use a Boolean type if you wish. Double array visBiases holds the bias values associated with each visible node. Double array visProbs holds visible node probabilities. Note that the visProbs array isn't necessary because node values can be computed on the fly; however, storing the probability values is useful if you want to examine the behavior of the RBM during runtime.

The other four Machine class fields are:

```
public int[] hidValues;
public double[] hidProbs;
public double[] hidBiases;
public double[][] vhWeights;
```

Arrays hidValues, hidBiases, and hidProbs are the node values, associated bias values, and node probabilities, respectively. The vhWeights object is an array-of-arrays style matrix where the row index corresponds to a visible node and the column index corresponds to a hidden node.

The key class method computes the values of the hidden nodes using values in a parameter that corresponds to the visible nodes. That method's definition begins with:

```
public int[] HiddenFromVis(int[] visibles)
{
  int[] result = new int[numHidden];
  ...
```

Next, the calculations of the hidden-layer nodes are done node by node:

```
for (int h = 0; h < numHidden; ++h) {
  double sum = 0.0;
  for (int v = 0; v < numVisible; ++v)
    sum += visibles[v] * vhWeights[v][h];
  sum += hidBiases[h]; // Add the hidden bias
  double probActiv = LogSig(sum); // Compute prob
  double pr = rnd.NextDouble();  // Determine 0/1
  if (probActiv > pr) result[h] = 1;
  else result[h] = 0;
}
```

The code mirrors the explanation of the input-output mechanism explained earlier. Function LogSig is a private helper function, because the Microsoft .NET Framework doesn't have a built-in logistic sigmoid function (at least that I'm aware of).

The key method concludes by returning the computed hidden node values to the caller:

```
  ...
  return result;
}
```

The rest of the demo code implements the CD-1 training algorithm as described earlier. The code isn't trivial, but if you examine it carefully, you should be able to make the connections between RBM concepts and implementation.

Wrapping Up

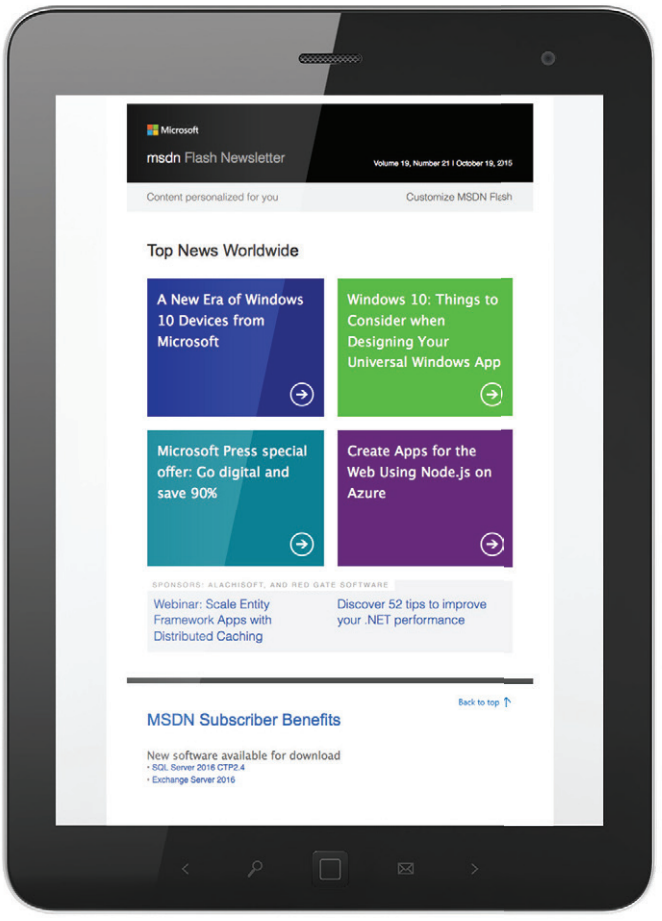
Restricted Boltzmann machines are simple and complicated at the same time. The RBM input-output mechanism is both deterministic and probabilistic (sometimes called stochastic), but is relatively easy to understand and implement. The more difficult aspect of RBMs, in my opinion, is understanding how they can be useful.

As a standalone software component, an RBM can act as a lossy compression machine to reduce the number of bits needed to represent some data, or can act as a probabilistic factor analysis component that identifies core concepts in a data set. When concatenated, RBMs can create a deep neural network structure called a "deep belief network" that can make predictions.

RBMs were invented in 1986 by my Microsoft colleague Paul Smolensky, but gained increased attention relatively recently when the CD-1 training algorithm was devised by researcher and Microsoft collaborator Geoffrey Hinton. Much of the information presented in this article is based on personal conversations with Smolensky, and the 2010 research paper, "A Practical Guide to Training Restricted Boltzmann Machines," by Hinton. ■

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 who reviewed this article: *Ani Anirudh, Qiuyuan Huang and Paul Smolensky*



Get news from MSDN in your inbox!

Sign up to receive **MSDN FLASH**, which delivers the latest resources, SDKs, downloads, partner offers, security news, and updates on national and local developer events.

msdn magazine

msdn.microsoft.com/flashnewsletter

The Ultimate Education Destination

2017 Orlando

ROYAL PACIFIC RESORT AT UNIVERSAL ORLANDO
NOVEMBER 12-17

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

- 6 FULL Days of Training
- 5 Co-Located Conferences
- 1 Low Price
- Create Your Own Agenda from Hundreds of Sessions
- Expert Education and Training
- Knowledge Share and Networking

CONNECT WITH LIVE! 360

twitter.com/live360
@live360

facebook.com
Search "Live 360"

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

EVENT PARTNERS
Microsoft Magenic

PLATINUM SPONSOR
SMARTBEAR

SUPPORTED BY
Visual Studio msdn magazine Redmond Channel Partner

LIVE! 360

TECH EVENTS WITH PERSPECTIVE

NEW: HANDS-ON LABS

Join us for full-day, pre-conference hands-on labs Sunday, November 12.

Only $595 through August 11

5 GREAT CONFERENCES, 1 GREAT PRICE

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

Visual Studio Live!: Code in Paradise at VSLive!™, featuring unbiased and practical development training on the Microsoft Platform.

SQL Server LIVE!

TRAINING FOR DBAs AND IT PROS

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

TECHMENTOR

IN-DEPTH TRAINING FOR IT PROS

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

Office & SharePoint LIVE!

ON-PREMISE, CLOUD & CROSS-PLATFORM TRAINING

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

Modern Apps LIVE!

MOBILE, CROSS-DEVICE & CLOUD DEVELOPMENT

Modern Apps Live!: Presented in partnership with Magenic, this unique conference delivers how to architect, design and build a complete Modern App.

REGISTER NOW

REGISTER BY AUGUST 11 AND SAVE $500!*

Use promo code L360MAY2

*Savings based on 5-day packages only. See website for details.

PRODUCED BY

Redmond Magazine | Virtualization & Cloud Review | Visual Studio Magazine | 1105 Media

LIVE360EVENTS.COM

Custom Iterators with Yield

In my last column (msdn.com/magazine/mt797654), I delved into the details of how the C# foreach statement works under the covers, explaining how the C# compiler implements the foreach capabilities in Common Intermediate Language (CIL). I also briefly touched on the yield keyword with an example (see **Figure 1**), but little to no explanation.

This is a continuation of that article, in which I provide more detail about the yield keyword and how to use it.

Iterators and State

By placing a break point at the start of the GetEnumerator method in **Figure 1**, you'll observe that GetEnumerator is called at the start of the foreach statement. At that point, an iterator object is created and its state is initialized to a special "start" state that represents the fact that no code has executed in the iterator and, therefore, no values have been yielded yet. From then on, the iterator maintains its state (location), as long as the foreach statement at the call site continues to execute. Every time the loop requests the next value, control enters the iterator and continues where it left off the previous time around the loop; the state information stored in the iterator object is used to determine where control must resume. When the foreach statement at the call site terminates, the iterator's state is no longer saved. **Figure 2** shows a high-level sequence diagram of what takes place. Remember that the MoveNext method appears on the IEnumerator<T> interface.

In **Figure 2**, the foreach statement at the call site initiates a call to GetEnumerator on the CSharpBuiltInTypes instance called keywords. As you can see, it's always safe to call GetEnumerator again; "fresh" enumerator objects will be created when necessary. Given the iterator instance (referenced by iterator), foreach begins each iteration with a call to MoveNext. Within the iterator, you yield a value back to the foreach statement at the call site. After the yield return statement, the GetEnumerator method seemingly pauses until the next MoveNext request. Back at the loop body, the foreach statement displays the yielded value on the screen. It then loops back around and calls MoveNext on the iterator again. Notice that the second time, control picks up at the second yield return statement. Once again, the foreach displays on the screen what CSharpBuiltInTypes yielded and starts the loop again. This process continues until there are no more yield return statements within the iterator. At that point, the foreach loop at the call site terminates because MoveNext returns false.

Code download available at itl.tc/MSDN.2017.06.

Another Iterator Example

Consider a similar example with the BinaryTree<T>, which I introduced in the previous article. To implement the BinaryTree<T>, I first need Pair<T> to support the IEnumerable<T> interface using an iterator. **Figure 3** is an example that yields each element in Pair<T>.

In **Figure 3**, the iteration over the Pair<T> data type loops twice: first through yield return First, and then through yield return Second. Each time the yield return statement within GetEnumerator is encountered, the state is saved and execution appears to "jump" out of the GetEnumerator method context and into the loop body. When the second iteration starts, GetEnumerator begins to execute again with the yield return Second statement.

Figure 1 **Yielding Some C# Keywords Sequentially**

```
using System.Collections.Generic;
public class CSharpBuiltInTypes: IEnumerable<string>
{
  public IEnumerator<string> GetEnumerator()
  {
    yield return "object";
    yield return "byte";
    yield return "uint";
    yield return "ulong";
    yield return "float";
    yield return "char";
    yield return "bool";
    yield return "ushort";
    yield return "decimal";
    yield return "int";
    yield return "sbyte";
    yield return "short";
    yield return "long";
    yield return "void";
    yield return "double";
    yield return "string";
  }
    // The IEnumerable.GetEnumerator method is also required
    // because IEnumerable<T> derives from IEnumerable.
  System.Collections.IEnumerator
    System.Collections.IEnumerable.GetEnumerator()
  {
    // Invoke IEnumerator<string> GetEnumerator() above.
    return GetEnumerator();
  }
}
public class Program
{
  static void Main()
  {
    var keywords = new CSharpBuiltInTypes();
    foreach (string keyword in keywords)
    {
      Console.WriteLine(keyword);
    }
  }
}
```

Implementing IEnumerable with IEnumerable<T>

System.Collections.Generic.IEnumerable<T> inherits from System.Collections.IEnumerable. Therefore, when implementing IEnumerable<T>, it's also necessary to implement IEnumerable. In **Figure 3**, it's done explicitly, and the implementation simply involves a call to the IEnumerable<T> GetEnumerator implementation. This call from IEnumerable.GetEnumerator to IEnumerable<T>.GetEnumerator will always work because of the type compatibility (via inheritance) between IEnumerable<T> and IEnumerable. Because the signatures for both GetEnumerators are identical (the return type doesn't distinguish a signature), one or both implementations must be explicit. Given the additional type safety offered by the IEnumerable<T> version, the IEnumerable implementation should be explicit.

The following code uses the Pair<T>.GetEnumerator method and displays "Inigo" and "Montoya" on two consecutive lines:

```
var fullname = new Pair<string>("Inigo", "Montoya");
foreach (string name in fullname)
{
  Console.WriteLine(name);
}
```

Placing a Yield Return Within a Loop

It's not necessary to hardcode each yield return statement, as I did in both CSharpPrimitiveTypes and Pair<T>. Using the yield return statement, you can return values from inside a loop construct. **Figure 4** uses a foreach loop. Each time the foreach within GetEnumerator executes, it returns the next value.

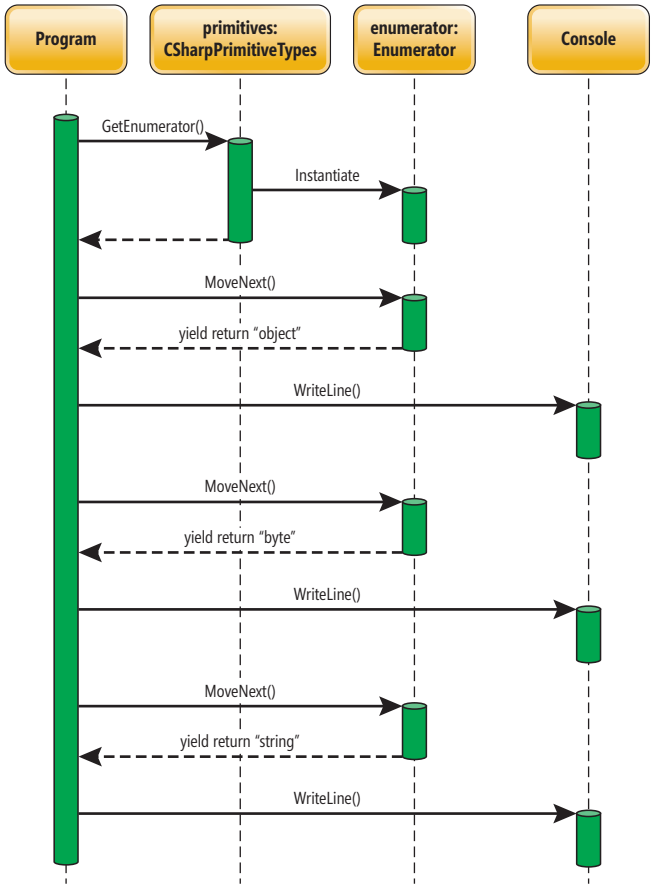


Figure 2 **Sequence Diagram with Yield Return**

Figure 3 **Using Yield to Implement BinaryTree<T>**

```
public struct Pair<T>: IPair<T>,
  IEnumerable<T>
{
  public Pair(T first, T second) : this()
  {
    First = first;
    Second = second;
  }
  public T First { get; }  // C# 6.0 Getter-only Autoproperty
  public T Second { get; } // C# 6.0 Getter-only Autoproperty

  #region IEnumerable<T>
  public IEnumerator<T> GetEnumerator()
  {
    yield return First;
    yield return Second;
  }
#endregion IEnumerable<T>

  #region IEnumerable Members
  System.Collections.IEnumerator
    System.Collections.IEnumerable.GetEnumerator()
  {
    return GetEnumerator();
  }
  #endregion
}
```

In **Figure 4**, the first iteration returns the root element within the binary tree. During the second iteration, you traverse the pair of subelements. If the subelement pair contains a non-null value, you traverse into that child node and yield its elements. Note that foreach (T item in tree) is a recursive call to a child node.

As observed with CSharpBuiltInTypes and Pair<T>, you can now iterate over BinaryTree<T> using a foreach loop. **Figure 5** demonstrates this process.

And here are the results:

```
John Fitzgerald Kennedy
Joseph Patrick Kennedy
Patrick Joseph Kennedy
Mary Augusta Hickey
Rose Elizabeth Fitzgerald
John Francis Fitzgerald
Mary Josephine Hannon
```

Canceling Further Iteration: Yield Break

Sometimes you might want to cancel further iteration. You can do so by including an if statement so that no further statements within

The Origin of Iterators

In 1972, Barbara Liskov and a team of scientists at MIT began researching programming methodologies, focusing on user-defined data abstractions. To prove much of their work, they created a language called CLU that had a concept called "clusters" (CLU being the first three letters of this term). Clusters were predecessors to the primary data abstraction that programmers use today: objects. During their research, the team realized that although they were able to use the CLU language to abstract some data representation away from end users of their types, they consistently found themselves having to reveal the inner structure of their data to allow others to intelligently consume it. The result of their consternation was the creation of a language construct called an iterator. (The CLU language offered many insights into what would eventually be popularized as "object-oriented programming.")

the code are executed. However, you can also use yield break to cause MoveNext to return false and control to return immediately to the caller and end the loop. Here's an example of such a method:

```
public System.Collections.Generic.IEnumerable<T>
  GetNotNullEnumerator()
{
  if((First == null) || (Second == null))
  {
    yield break;
  }
  yield return Second;
  yield return First;
}
```

This method cancels the iteration if either of the elements in the Pair<T> class is null.

Figure 4 **Placing Yield Return Statements Within a Loop**

```
public class BinaryTree<T>: IEnumerable<T>
{
  // ...

  #region IEnumerable<T>
  public IEnumerator<T> GetEnumerator()
  {
    // Return the item at this node.
    yield return Value;

    // Iterate through each of the elements in the pair.
    foreach (BinaryTree<T> tree in SubItems)
    {
      if (tree != null)
      {
        // Because each element in the pair is a tree,
        // traverse the tree and yield each element.
        foreach (T item in tree)
        {
          yield return item;
        }
      }
    }
  }
  #endregion IEnumerable<T>

  #region IEnumerable Members
  System.Collections.IEnumerator
    System.Collections.IEnumerable.GetEnumerator()
  {
    return GetEnumerator();
  }
  #endregion
}
```

Figure 5 **Using foreach with BinaryTree<string>**

```
// JFK
var jfkFamilyTree = new BinaryTree<string>(
  "John Fitzgerald Kennedy");

jfkFamilyTree.SubItems = new Pair<BinaryTree<string>>(
  new BinaryTree<string>("Joseph Patrick Kennedy"),
  new BinaryTree<string>("Rose Elizabeth Fitzgerald"));

// Grandparents (Father's side)
jfkFamilyTree.SubItems.First.SubItems =
  new Pair<BinaryTree<string>>(
  new BinaryTree<string>("Patrick Joseph Kennedy"),
  new BinaryTree<string>("Mary Augusta Hickey"));

// Grandparents (Mother's side)
jfkFamilyTree.SubItems.Second.SubItems =
  new Pair<BinaryTree<string>>(
  new BinaryTree<string>("John Francis Fitzgerald"),
  new BinaryTree<string>("Mary Josephine Hannon"));

foreach (string name in jfkFamilyTree)
{
  Console.WriteLine(name);
}
```

A yield break statement is similar to placing a return statement at the top of a function when it's determined there's no work to do. It's a way to exit from further iterations without surrounding all remaining code with an if block. As such, it allows multiple exits. Use it with caution, because a casual reading of the code might overlook the early exit.

How Iterators Work

When the C# compiler encounters an iterator, it expands the code into the appropriate CIL for the corresponding enumerator design pattern. In the generated code, the C# compiler first creates a nested private class to implement the IEnumerator<T> interface, along with its Current property and a MoveNext method. The Current

Figure 6 **C# Equivalent of Compiler-Generated C# Code for Iterators**

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

public class Pair<T> : IPair<T>, IEnumerable<T>
{
  // ...

  // The iterator is expanded into the following
  // code by the compiler.
  public virtual IEnumerator<T> GetEnumerator()
  {
    __ListEnumerator result = new __ListEnumerator(0);
    result._Pair = this;
    return result;
  }
  public virtual System.Collections.IEnumerator
    System.Collections.IEnumerable.GetEnumerator()
  {
    return new GetEnumerator();
  }

  private sealed class __ListEnumerator<T> : IEnumerator<T>
  {
    public __ListEnumerator(int itemCount)
    {
      _ItemCount = itemCount;
    }

    Pair<T> _Pair;
    T _Current;
    int _ItemCount;

    public object Current
    {
      get
      {
        return _Current;
      }
    }

    public bool MoveNext()
    {
      switch (_ItemCount)
      {
        case 0:
          _Current = _Pair.First;
          _ItemCount++;
          return true;
        case 1:
          _Current = _Pair.Second;
          _ItemCount++;
          return true;
        default:
          return false;
      }
    }
  }
}
```

TECHMENTOR

IN-DEPTH TRAINING FOR IT PROS

AUGUST 7 - 11, 2017

MICROSOFT HEADQUARTERS

REDMOND, WA

PLUG IN TO NEW KNOWLEDGE @ THE SOURCE

WHAT SETS TECHMENTOR APART?

+ Immediately usable IT education
+ Training you need today, while preparing you for tomorrow
+ Zero marketing-speak, a strong emphasis on doing more with the technology you already own, and solid coverage of what's just around the corner
+ Intimate setting, where your voice is heard, making it a viable alternative to huge, first-party conferences
+ Experience life @ Microsoft Headquarters for a full week

YOU OWE IT TO YOURSELF, YOUR COMPANY AND YOUR CAREER TO BE AT TECHMENTOR REDMOND 2017!

HOT TRAINING TOPICS INCLUDE:

+ Windows Server + Hyper-V + Windows PowerShell + DSC + DevOps + Azure + Security + And More! +

++

REGISTER NOW

SAVE $300 THROUGH JUNE 30
USE PROMO CODE TMJUN1

[TECHMENTOREVENTS.COM/REDMOND]

EVENT SPONSOR: Microsoft

GOLD SPONSOR: GOVERLAN REACH

SUPPORTED BY: Redmond | Redmond Channel Partner | Virtualization Review

PRODUCED BY: 1105 MEDIA

property returns a type corresponding to the return type of the iterator. As you saw in **Figure 3**, Pair<T> contains an iterator that returns a T type. The C# compiler examines the code contained within the iterator and creates the necessary code within the MoveNext method and the Current property to mimic its behavior. For the Pair<T> iterator, the C# compiler generates roughly equivalent code (see **Figure 6**).

Because the compiler takes the yield return statement and generates classes that correspond to what you probably would've written manually, iterators in C# exhibit the same performance characteristics as classes that implement the enumerator design pattern manually. Although there's no performance improvement, the gains in programmer productivity are significant.

Creating Multiple Iterators in a Single Class

Previous iterator examples implemented IEnumerable<T>.GetEnumerator, which is the method that foreach seeks implicitly. Sometimes you might want different iteration sequences, such as iterating in reverse, filtering the results or iterating over an object projection other than the default. You can declare additional iterators in the class by encapsulating them within properties or methods that return IEnumerable<T> or IEnumerable. If you want to iterate over the elements of Pair<T> in reverse, for example, you could provide a GetReverseEnumerator method, as shown in **Figure 7**.

Note that you return IEnumerable<T>, not IEnumerator<T>. This is different from IEnumerable<T>.GetEnumerator, which returns IEnumerator<T>. The code in Main demonstrates how to call GetReverseEnumerator using a foreach loop.

Yield Statement Requirements

You can use the yield return statement only in members that return an IEnumerator<T> or IEnumerable<T> type, or their nongeneric equivalents. Members whose bodies include a yield return statement may not have a simple return. If the member uses the yield return statement, the C# compiler generates the necessary code to maintain the state of the iterator. In contrast, if the member uses the return statement instead of yield return, the programmer is responsible for maintaining his own state machine and returning an instance of one of the iterator interfaces. Further, just as all code paths in a method with a return type must contain a return statement accompanied by a value (assuming they don't throw an exception), all code paths in an iterator must contain a yield return statement if they are to return any data.

Figure 7 **Using Yield Return in a Method That Returns IEnumerable<T>**

```
public struct Pair<T>: IEnumerable<T>
{
  ...

  public IEnumerable<T> GetReverseEnumerator()
  {
    yield return Second;
    yield return First;
  }
  ...
}

public void Main()
{
  var game = new Pair<string>("Redskins", "Eagles");
  foreach (string name in game.GetReverseEnumerator())
  {
    Console.WriteLine(name);
  }
}
```

If the member uses the yield return statement, the C# compiler generates the necessary code to maintain the state of the iterator.

The following additional restrictions on the yield statement result in compiler errors if they're violated:

- The yield statement may appear only inside a method, a user-defined operator, or the get accessor of an indexer or property. The member must not take any ref or out parameter.
- The yield statement may not appear anywhere inside an anonymous method or lambda expression.
- The yield statement may not appear inside the catch and finally clauses of the try statement. Furthermore, a yield statement may appear in a try block only if there is no catch block.

Wrapping Up

Overwhelmingly, generics was the cool feature launched in C# 2.0, but it wasn't the only collection-related feature introduced at the time. Another significant addition was the iterator. As I outlined in this article, iterators involve a contextual keyword, yield, that C# uses to generate underlying CIL code that implements the iterator pattern used by the foreach loop. Furthermore, I detailed the yield syntax, explaining how it fulfills the GetEnumerator implementation of IEnumerable<T>, allows for breaking out of a loop with yield break and even supports a C# method that returns an IEnumeable<T>.

Much of this column derives from my "Essential C#" book (IntelliTect.com/EssentialCSharp), which I am currently in the midst of updating to "Essential C# 7.0." For more information on this topic, check out Chapter 16. ■

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 IntelliTect technical experts for reviewing this article: *Kevin Bost, Grant Erickson, Chris Finlayson, Phil Spokas and Michael Stokesbary*

Spreadsheets Made Easy.

Fastest Calculations

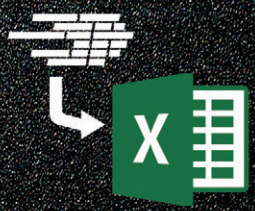
Evaluate complex Excel-based models and business rules with the fastest and most complete Excel-compatible calculation engine available.

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

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

REDMOND
AUGUST 14-18, 2017
MICROSOFT HEADQUARTERS

JOIN US AT MICROSOFT HEADQUARTERS THIS SUMMER

- Rub elbows with blue badges
- Experience life on campus
- Enjoy lunch in the Commons and visit the Company Store
- Networking event on Lake Washington, Wednesday, August 16
- And so much more!

SUNDAY, AUG 13: PRE-CON HANDS-ON LABS

Choose From:

- Angular
- Dev Ops with ASP.NET Core/EF Core
- SQL Server 2016

NEW! Only $595!

SPACE IS LIMITED

EVENT PARTNER: Microsoft

GOLD SPONSORS: smartsheet, Syncfusion

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

PRODUCED BY: 1105 MEDIA — YOUR GROWTH. OUR BUSINESS.

INTENSE TRAINING FOR DEVELOPERS, ENGINEERS, PROGRAMMERS, ARCHITECTS AND MORE!

Development Topics include:

- AngularJS
- ASP.NET Core
- Azure
- Analytics
- DevOps
- .NET Framework
- Software Practices
- SQL Server
- Visual Studio 2017
- Web API
- UWP
- Xamarin

TURN THE PAGE FOR FULL AGENDA DETAILS

Register by June 9 and Save $400*

Use promo code VSLRED2

*Savings based on 5 day packages only.

REGISTER NOW

ROCK YOUR CODE TOUR 2017

REDMOND AUG 14-18

"I liked that there was representation of Android and iOS, as well as, Microsoft. I prefer working with Microsoft tech/code but can't ignore Android and iOS." *– Chris Nacey, Site Crafting*

"This is the first conference that I have attended @Microsoft Headquarters and it truly has been a fantastic experience. Definitely exceeded my expectations and I look forward to coming back." *– David Campbell, G3 Software*

vslive.com/redmondmsdn

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

REDMOND
AUGUST 14-18, 2017
MICROSOFT HEADQUARTERS

Explore the Microsoft Campus during Visual Studio Live! Redmond 2017!

Scott Hanselman, Keynote Speaker

START TIME	END TIME
8:00 AM	9:00 AM
9:00 AM	6:00 PM

START TIME	END TIME
7:00 AM	8:00 AM
8:00 AM	12:00 PM
12:00 PM	2:00 PM
2:00 PM	5:30 PM
7:00 PM	9:00 PM

START TIME	END TIME
7:30 AM	8:00 AM
8:00 AM	9:15 AM
9:30 AM	10:45 AM
10:45 AM	11:15 AM
11:15 AM	12:15 PM
12:15 PM	1:30 PM
1:30 PM	2:45 PM
3:00 PM	4:15 PM
4:15 PM	5:45 PM

START TIME	END TIME
7:30 AM	8:00 AM
8:00 AM	9:15 AM
9:30 AM	10:45 AM
11:00 AM	12:00 PM
12:00 PM	1:30 PM
1:30 PM	2:45 PM
2:45 PM	3:15 PM
3:15 PM	4:30 PM
6:15 PM	8:30 PM

START TIME	END TIME
7:30 AM	8:00 AM
8:00 AM	9:15 AM
9:30 AM	10:45 AM
11:00 AM	12:15 PM
12:15 PM	2:15 PM
2:15 PM	3:30 PM
3:45 PM	5:00 PM

START TIME	END TIME
7:30 AM	8:00 AM
8:00 AM	5:00 PM

MICROSOFT SET LIST: DETAILS DROPPING SOON!

With all of the announcements sure to come out of Build, we'll be finalizing the FULL Track of Microsoft-led sessions shortly.

Microsoft Speakers are noted with a

Be sure to check vslive.com/redmondmsdn for session updates!

ROCK YOUR CODE TOUR 2017

Register by June 9 and Save $400*

Use promo code VSLRED2

*Savings based on 5 day packages only.

REGISTER NOW

EVENT PARTNER
Microsoft

GOLD SPONSORS
smartsheet
Syncfusion

SUPPORTED BY
Visual Studio
msdn magazine
Visual Studio MAGAZINE

PRODUCED BY
1105 MEDIA
YOUR GROWTH. OUR BUSINESS.

REDMOND AGENDA AT-A-GLANCE

Microsoft Set List	ALM / DevOps	Cloud Computing	Database and Analytics	Native Client	Software Practices	Visual Studio / .NET Framework	Web Client	Web Server

NEW Full Day Hands-On Labs: Sunday, August 13, 2017 *(Separate entry fee required)*

Pre-Conference HOL Workshop Registration - Coffee and Morning Pastries

HOL01 Full Day Hands-On Lab: Busy Developer's HOL on Angular - *Ted Neward*	**HOL02** Full Day Hands-On Lab: DevOps with ASP.NET Core and EF Core - *Benjamin Day & Brian Randell*	**HOL03** Full Day Hands-On Lab: Developer Dive into SQL Server 2016 - *Leonard Lobel*

Visual Studio Live! Pre-Conference Workshops: Monday, August 14, 2017 *(Separate entry fee required)*

Pre-Conference Workshop Registration - Coffee and Morning Pastries

M01 Workshop: Modern Security Architecture for ASP.NET Core - *Brock Allen*	**M02** Workshop: Distributed Cross-Platform Application Architecture - *Jason Bock & Rockford Lhotka*	**M03** Workshop: Big Data, BI and Analytics on The Microsoft Stack - *Andrew Brust*

Lunch @ The Mixer - Visit the Microsoft Company Store & Visitor Center

M01 Workshop Continues - *Brock Allen*	**M02** Workshop Continues - *Jason Bock & Rockford Lhotka*	**M03** Workshop Continues - *Andrew Brust*

Dine-A-Round Dinner

Visual Studio Live! Day 1: Tuesday, August 15, 2017

Registration - Coffee and Morning Pastries

T01 Go Mobile With C#, Visual Studio, and Xamarin - *James Montemagno*	**T02** Angular 101: Part 1 - *Deborah Kurata*	**T03** New SQL Server 2016 Security Features for Developers - *Leonard Lobel*	**T04** What's New in Visual Studio 2017 - *Robert Green*	**T05** Microsoft Set List: Details Dropping Soon
T06 Building Connected and Disconnected Mobile Apps - *James Montemagno*	**T07** Angular 101: Part 2 - *Deborah Kurata*	**T08** No Schema, No Problem! Introduction to Azure DocumentDB - *Leonard Lobel*	**T09** Getting to the Core of .NET Core - *Adam Tuliper*	**T10** Microsoft Set List: Details Dropping Soon

Sponsored Break - Visit Exhibitors

KEYNOTE: To Be Announced – ***Scott Hanselman**, Principal Community Architect for Web Platform and Tools, Microsoft*

Lunch - Visit Exhibitors

T11 Take the Tests: Can You Evaluate Good and Bad Designs? - *Billy Hollis*	**T12** Assembling the Web - A Tour of WebAssembly - *Jason Bock*	**T13** Unit Testing & Test-Driven Development (TDD) for Mere Mortals - *Benjamin Day*	**T14** To Be Announced	**T15** Microsoft Set List: Details Dropping Soon
T16 A Developers Introduction to HoloLens - *Billy Hollis & Brian Randell*	**T17** To Be Announced	**T18** Spans, Memory, and Channels - Making .NET Code Fast - *Jason Bock*	**T19** Entity Framework Core for Enterprise Applications - *Benjamin Day*	**T20** Microsoft Set List: Details Dropping Soon

Microsoft Ask the Experts & Exhibitor Reception—Attend Exhibitor Demos, *Sponsored by* smartsheet

Visual Studio Live! Day 2: Wednesday, August 16, 2017

Registration - Coffee and Morning Pastries

W01 Roll Your Own Dashboard in XAML - *Billy Hollis*	**W02** Migrating to ASP.NET Core - A True Story - *Adam Tuliper*	**W03** Hacker Trix - Learning from OWASP Top 10 - *Mike Benkovich*	**W04** Distributed Architecture: Microservices and Messaging - *Rockford Lhotka*	**W05** Microsoft Set List: Details Dropping Soon
W06 Customizing Your UI for Mobile Devices: Techniques to Create a Great User Experience - *Laurent Bugnion*	**W07** User Authentication for ASP.NET Core MVC Applications - *Brock Allen*	**W08** From Containers to Data in Motion, Tour d'Azure 2017 - *Mike Benkovich*	**W09** Agile: You Keep Using That Word... - *Philip Japikse*	**W10** Microsoft Set List: Details Dropping Soon

General Session: To Be Announced

Birds-of-a-Feather Lunch - Visit Exhibitors

W11 Building Cross-platform App. Dev. with CLSA.NET - *Rockford Lhotka*	**W12** Securing Web APIs in ASP.NET Core - *Brock Allen*	**W13** Tactical DevOps with VSTS - *Brian Randell*	**W14** Agile Failures: Stories from The Trenches - *Philip Japikse*	**W15** TypeScript and the Future of JavaScript - *Jordan Matthiesen & Bowden Kelly*

Sponsored Break - Exhibitor Raffle @ 2:55 pm (Must be present to win)

W16 Building Truly Universal Applications with Windows, Xamarin and MVVM - *Laurent Bugnion*	**W17** Integrating AngularJS & ASP.NET MVC - *Miguel Castro*	**W18** Get Started with Git and GitHub - *Robert Green*	**W19** SOLID – The Five Commandments of Good Software - *Chris Klug*	**W20** Using Angular 2, JavaScript, and TypeScript to Build Fast and Secure Mobile Apps - *Jordan Matthiesen*

Set Sail! VSLive!'s Seattle Sunset Cruise - Advanced Reservation & $10 Fee Required

Visual Studio Live! Day 3: Thursday, August 17, 2017

Registration - Coffee and Morning Pastries

TH01 Lessons Learned from Real World Xamarin.Forms Projects - *Nick Landry*	**TH02** Build Real-Time Websites and Apps with SignalR - *Rachel Appel*	**TH03** "Aurelia vs "Just Angular" a.k.a "The Framework Formerly Known as Angular 2" - *Chris Klug*	**TH04** Go Serverless with Azure Functions - *Eric D. Boyd*	**TH05** Microsoft Set List: Details Dropping Soon
TH06 Creating Great Looking Android Applications Using Material Design - *Kevin Ford*	**TH07** Hard Core ASP.NET Core - *Rachel Appel*	**TH08** Database Lifecycle Management and the SQL Server Database - *Brian Randell*	**TH09** Breaking Down Walls with Modern Identity - *Eric D. Boyd*	**TH10** Microsoft Set List: Details Dropping Soon
TH11 Software Engineering in an Agile Environment - *David Corbin*	**TH12** Enriching MVC Sites with Knockout JS - *Miguel Castro*	**TH13** Power BI: Analytics for Desktop, Mobile and Cloud - *Andrew Brust*	**TH14** Bots are the New Apps: Building Bots with ASP.NET WebAPI & Language Understanding - *Nick Landry*	**TH15** Microsoft Set List: Details Dropping Soon

Lunch @ The Mixer - Visit the Microsoft Company Store & Visitor Center

TH16 Classic Software Design Principles and Why They Are Still Important - *David Corbin*	**TH17** Getting Started with Aurelia - *Brian Noyes*	**TH18** Big Data with Hadoop, Spark and Azure HDInsight - *Andrew Brust*	**TH19** Extend and Customize the Visual Studio Environment - *Walt Ritscher*	**TH20** Microsoft Set List: Details Dropping Soon
TH21 End-to-End Dependency Injection & Testable Code - *Miguel Castro*	**TH22** Securing Client Apps with IdentityServer - *Brian Noyes*	**TH23** Continuous Integration and Deployment for Mobile using Azure Services - *Kevin Ford*	**TH24** Windows Package Management with NuGet and Chocolatey - *Walt Ritscher*	**TH25** Microsoft Set List: Details Dropping Soon

Visual Studio Live! Post-Conference Workshops: Friday, August 18, 2017 *(Separate entry fee required)*

Post-Conference Workshop Registration - Coffee and Morning Pastries

F01 Workshop: Building Modern Web Apps with Azure - *Eric D. Boyd*	**F02** Workshop: Data-Centric Single Page Apps with Aurelia, Breeze, and Web API - *Brian Noyes*

Speakers and sessions subject to change

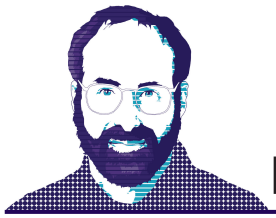
vslive.com/redmondmsdn

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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



DAVID S. PLATT

I'm Still Flying, Part 2

Last time, you remember, I was telling you about our UX design project with Zak Kohane of Harvard Medical School. Herewith the exciting conclusion:

Our project was designing the UX; not coding it, but determining what ought to be coded. We only had 10 sessions to get it all done. My students and I performed a straightforward, albeit intense, application of the Plattski Protocol™ that I published in my book, "The Joy of UX" (Addison-Wesley, 2016, joyofux.com).

We started, as always, with "Step 1: Who?" We analyzed the user population, quickly realizing that the patient isn't the user, the patient's caregiver is. We developed two personas: one for a family member caring for an aging relative (which we called the singular case), the other for a professional monitoring 100 patients as part of a medical practice (the plural case).

The next night we tackled "Step 2: What?" We wrote detailed UX stories, so very different from Agile development stories because they're told from the persona's viewpoint, not the program's. Then we moved to "Step 3: Sketch It." Each student used Balsamiq to sketch out a mockup design of each case, constantly receiving feedback from me and the other students.

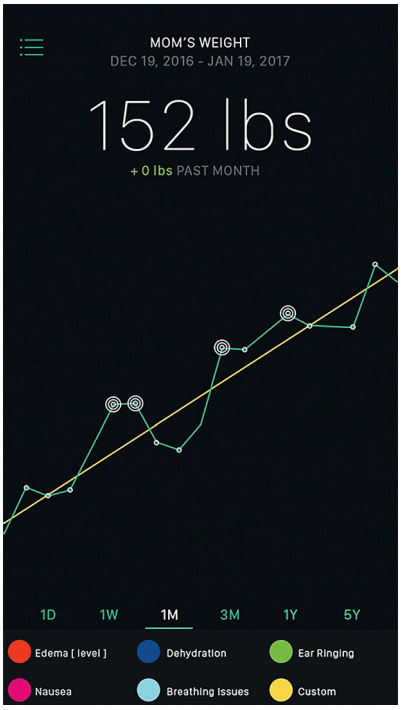


Figure 1 **Dr. Kohane raved about the ability to track daily weight changes.**

The most important step, and the one that gets omitted most often in practice, is "Step 4: Try It Out." There is no substitute, *none*, for trying your mockups on actual users. The sooner you do this, the easier and cheaper it is to make the corrections that you will surely need to make. As Disney's Pocahontas sings: "When you walk the footsteps of a stranger, you learn things you never knew you never knew."

How did we find test users? Easy. The students themselves knew plenty of good candidates. One student's mother had spent years caring for aging relatives. Another student's wife had a friend who worked with congestive heart failure (CHF) patients as a visiting nurse. They were happy to help us, honored—as most test users are—that we valued their opinion. We set up simple video connections with each of them using Google Hangouts, showing them the best of the student mockups, asking them to imagine themselves using it. "Mrs. X, imagine that you are monitoring your mother for weight gain, exacerbating her CHF. Here's the app, where do you start?" "Well, the weights are obviously shown here, and this one is obviously today's, so I look …"

As always, the test users gave us vital information: "I like [this]. I hate [that]. I don't understand [this other]. And by the way, did you ever consider [such and such]?" As always, we said, "Damn, I never for a microsecond imagined [whatever]." And as always, within 30 seconds, our amazement mutated into, "Well, duh (head slap), what could be more obvious?"

Then back to Balsamiq. Fix the sketches. What didn't the test user understand? It's not their job to decipher things, it's our job to clarify them. Change that label's wording. They don't need this control here; move it to another screen. Try it out. Present. Critique. Improve. Repeat. It was the most intense three weeks I've ever spent at Harvard.

Zak came to our last class to see our results. Each student presented their design (one of these examples, from Julie Dubela, is shown in **Figure 1**). Zak was astounded. After the first two, he said, "You know, we doctors don't have anything remotely like this." After four, he said, "We really need this." And at the end he gave us what I think he considers the ultimate accolade: "Something like this would make us better doctors."

So where to now? I'm not sure. This obviously has huge potential. We could start a monitoring business, telling insurance companies: "Don't pay us a nickel up front. We just want half the money that we save you." Anyone up for a startup?

I've written before of the huge economic pressure building up behind the dam of medical informatics. (See my February 2013 *MSDN Magazine* column, "What's Up, Doc?," at msdn.com/magazine/jj891060.) This project could be the first major crack. ■

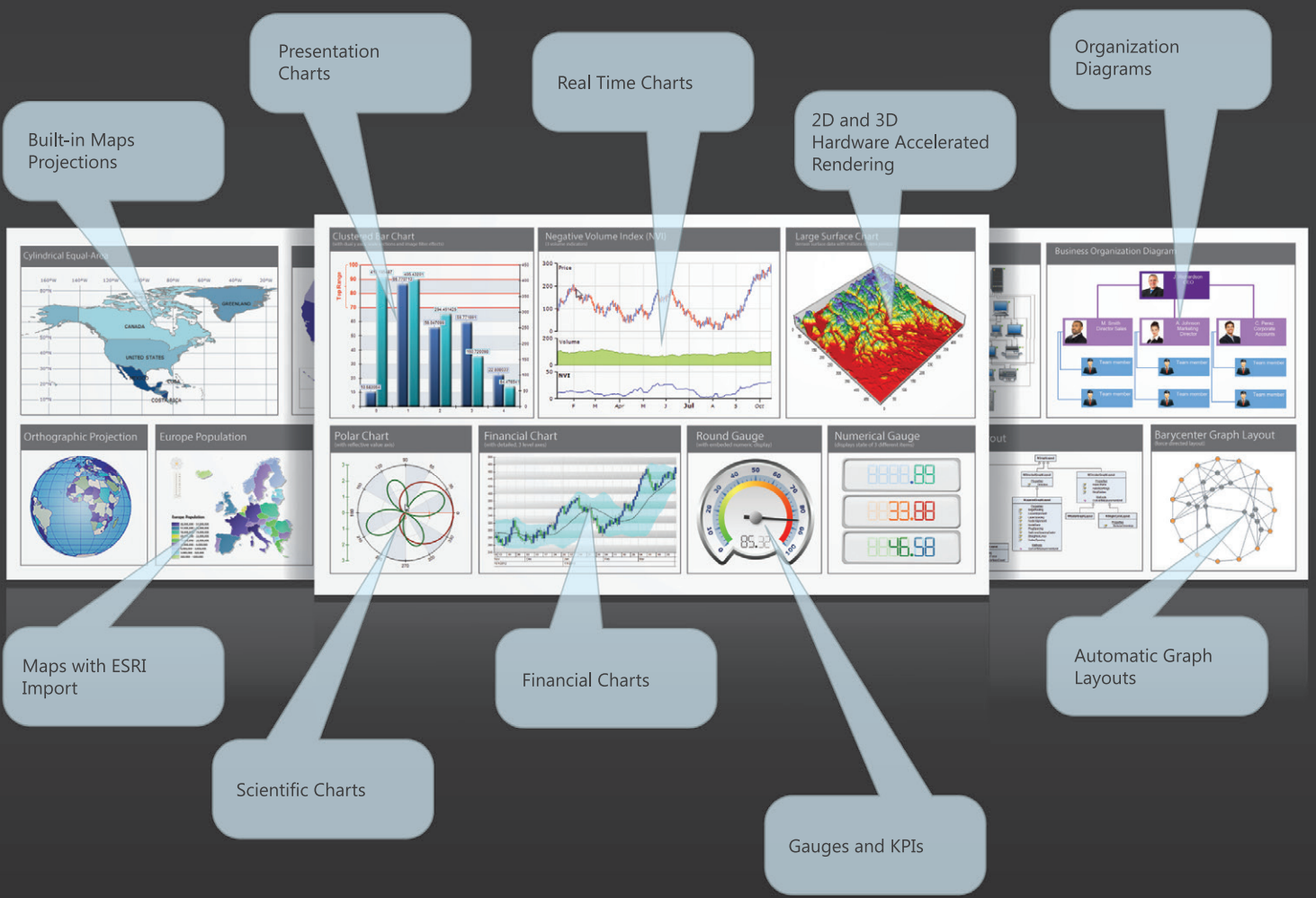
DAVID S. PLATT *teaches programming .NET at Harvard University Extension School and at companies all over the world. He's the author of 11 programming books, including "Why Software Sucks" (Addison-Wesley Professional, 2006) and "Introducing Microsoft .NET" (Microsoft Press, 2002). Microsoft named him 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.*

NEVRON

www.nevron.com

Nevron Data Visualization

The leading data visualization components for desktop and web development in a single package.



solutions available for

Microsoft ASP.net | Microsoft Silverlight | WPF | WinForms | SharePoint | Microsoft SQl Server | Xamarin

Learn more at **www.nevron.com** today

www.nevron.com | email@nevron.com | +1 888-201-6088 (Toll free, USA and Canada)