THE MICROSOFT JOURNAL FOR DEVELOPERS

JULY 2012 VOL 27 NO 7

msdn magazine

COLUMNS

Microsoft

Start a Revolution

Refuse to choose between desktop and mobile.

With the brand new NetAdvantage for .NET, you can create awesome apps with killer data visualization today, on any platform or device.

Get your free, fully supported trial today!

www.infragistics.com/NET

INFRAGISTICS™
DESIGN / DEVELOP / EXPERIENCE

Infragistics Sales US 800 231 8588 • Europe +44 (0) 800 298 9055 • India +91 80 4151 8042 • APAC (+61) 3 9982 4545


PopulationExplosion
VISUALIZING THE GROWTH OF THE WORLD COMMUNITY
XamPivotGrid
XamDataChart
Population
Measures
Country
List
Maria Anders
Ana Trujillo
Antonio Moreno
Thomas Hardy
Christina Berglund
Hanna Moos
Frédérique Citeaux
Martin Sommer
HTML
Compatible with
Microsoft® Visual Studio® 11 Beta

msdn magazine

JULY 2012 VOLUME 27 NUMBER 7

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

Corporate Address: 1105 Media, Inc., 9201 Oakdale Ave., Ste 101, Chatsworth, CA 91311, www.1105media.com

Media Kits: Direct your Media Kit requests to Matt Morollo, VP Publishing, 508-532-1418 (phone), 508-875-6622 (fax), mmorollo@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, Merit Direct. Phone: 914-368-1000; E-mail: 1105media@meritdirect.com; Web: www.meritdirect.com/1105

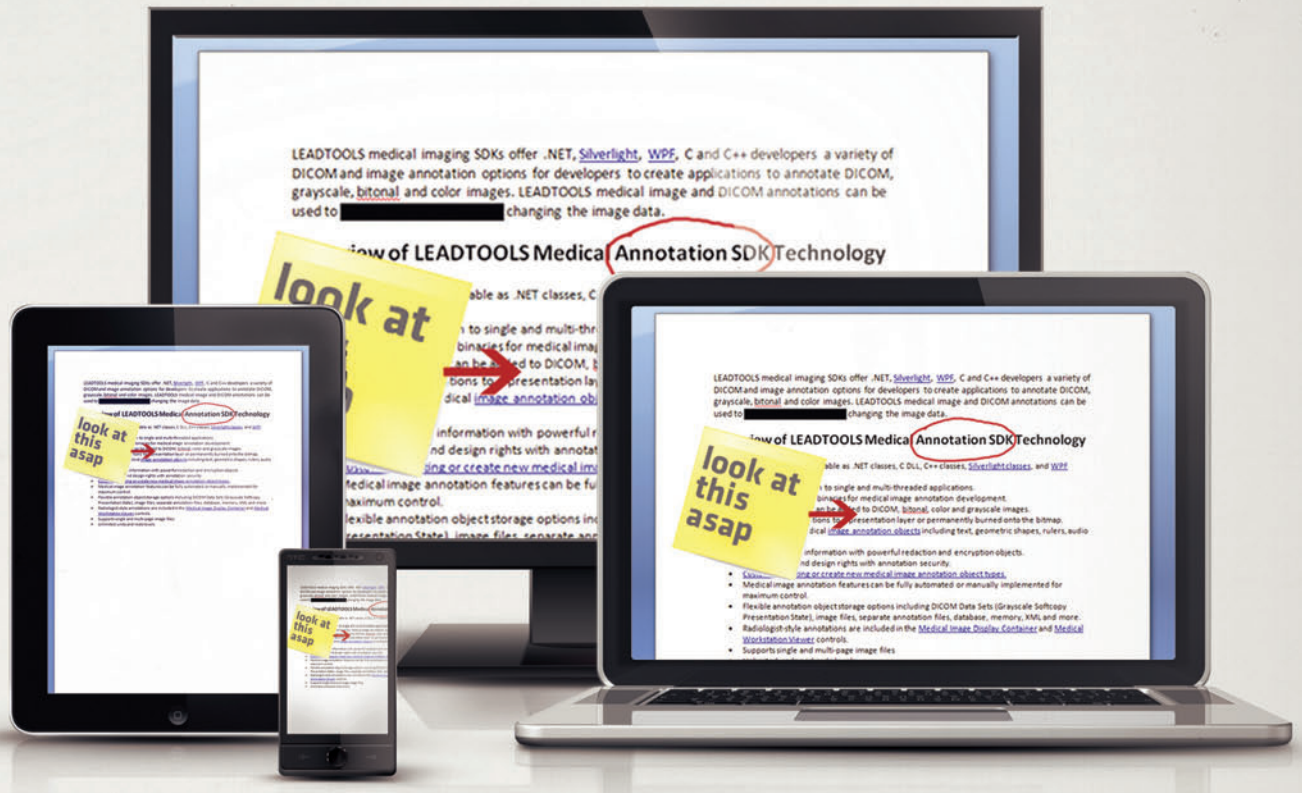
All customer service inquiries should be sent to MSDNmag@1105service.com or call 847-763-9560.

Microsoft

BPA WORLDWIDE

Printed in the USA

LEADTOOLS®
HTML5 SDKs



ZERO FOOTPRINT AND RICH CLIENT CONTROLS FOR WEB, MOBILE, & DESKTOP

MULTITOUCH SUPPORT FOR PAN, ZOOM, MAGNIFYING GLASS, ROTATE AND MORE

IMAGE MARKUP AND ANNOTATIONS

MEDICAL IMAGE DISPLAY FOR 12/16 BIT GRAYSCALE WITH WINDOW LEVEL

WEB SERVICES FOR IMAGE FORMATS, BARCODE, OCR, IMAGE PROCESSING AND MORE

DOWNLOAD OUR 60 DAY EVALUATION
WWW.LEADTOOLS.COM

LEAD TECHNOLOGIES INCORPORATED

SALES@LEADTOOLS.COM
800.637.1840

EDITOR'S NOTE

MICHAEL DESMOND

The Long Test Run

If you want to grow old in the magazine and Web editing business, you can't do it alone. Every editor needs a ringer—a trusted author who can step in and make things good even when they've gone really, really bad. Peter Vogel was that guy for me over at *Visual Studio Magazine* (and continues to be for current *Visual Studio Magazine* Editor in Chief Keith Ward), and Andrew Brust has saved my skin numerous times both at *Visual Studio Magazine* and, before that, at *Redmond Developer News*.

Nowadays, I've come to rely on James McCaffrey as my resident ringer at *MSDN Magazine*. McCaffrey writes the monthly Test Run column, often exploring some of the most technically challenging and fascinating concepts in each issue of the magazine. He also serves as a technical consultant, collaborating with editors to review article proposals.

Like some other longtime contributors to *MSDN Magazine*, McCaffrey got his start here "by accident." He had been exploring how to code for the Advanced Encryption Standard (AES) using the then-new C# programming language. His manager at the time read an article that McCaffrey had written based on his experience, and suggested that he submit it to *MSDN Magazine*.

"By sheer coincidence, *MSDN Magazine* was preparing a special issue on security and my article was accepted for publication," McCaffrey recalls. "The editors and production people helped me a lot and from that experience I was able to continue contributing to the magazine."

Indeed he has. Over the years, McCaffrey has proposed, written and reviewed innumerable article concepts. And, no surprise, he has some pointed thoughts on what makes an idea worthy of publication. McCaffrey says he looks for three things in an article pitch: new information, useful or interesting content, and a focus on code.

"I check to make sure the topic presents new, unpublished information. I also consider the scale of the proposed topic—some techniques are really interesting and new, but are better suited for a simple blog posting than an *MSDN Magazine* article," McCaffrey says, adding that he prefers articles "focus on actual development—architecture, design and coding—rather than a tutorial on a tool or a simple code-wrapper library."

As for his own work, McCaffrey says he looks for a "certain geek wow factor" when deciding what to explore in his column. That said, he works to ensure the topics include coding techniques and algorithms that can be used in normal software development situations. The April 2012 Test Run column, "Bacterial Foraging Optimization," is a case in point. It describes a fascinating algorithm that's based on the behavior of E. coli bacteria.

> Test Run wasn't always about bacterial super-algorithms and artificial intelligence.

"I find such algorithms really interesting and also often surprisingly useful. I think that as software development grows more sophisticated—especially with regard to the increased usage of cloud computing, big data and mobile devices—the AI topics in Test Run will move from interesting to essential," he says.

The thing is, Test Run wasn't always about bacterial super-algorithms and artificial intelligence. The column got its start when Microsoft released the Microsoft .NET Framework. The managed code framework created opportunities for software testing that simply did not exist with C++ and classic Visual Basic.

"Test Run was able to explore and explain techniques such as HTTP request-response testing, Windows Forms UI testing and so on," McCaffrey says. "By .NET 3.5 these techniques were established and quite well-known. Test Run gradually shifted to a new set of largely unexplored topics that generally fall into the category of artificial intelligence."

In a sense, McCaffrey has adapted in much the way the bacterial algorithms in his Test Run columns might. And along the way he's provided some valuable lessons for aspiring authors and overworked editors alike.

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

Brilliantly intuitive tools *for* dev and support teams.

Powered by HTML 5

OnTime Scrum

Agile project management & bug tracking

- product backlogs, releases, and sprints
- powerful user story and bug tracking
- automated burndown charts
- visual planning board

OnTime Help Desk

Customer support for software apps

- auto-generate tickets for support emails
- auto-responses & canned responses
- organize customers and contacts
- fully customizable Customer Portal

OnTime Team Wiki

Project wiki for collaborative dev teams

- hierarchical document organization
- configurable security controls
- HTML and WYSIWYG editor
- table of contents and search support

Pricing (11+ users, per product)

$7 *per user per month*

Special Small-team Pricing (up to 10 users)

$10 *per month per product*

Visit OnTimeNow.com/MSDN and....

Watch our popular *Scrum in Under 10 Minutes* video

Download and print our free Scrum Diagram

Sign up for a free trial of OnTime OnDemand

Develop in Visual Studio?

Our new **OnTime Visual Studio Extension** is as agile as your dev team, bringing the power of OnTime to your Visual Studio environment. Learn more at **ontimenow.com/addons**.

axosoft

axosoft.com
@axosoft
800.653.0024

DINO ESPOSITO

Mobile Site Development, Part 2: Design

There are a few myths about mobile site development that mostly derive from the idea that a mobile solution is a spin-off of an existing desktop Web site. I think the term spin-off is quite appropriate to describe the relationship between a mobile and a desktop Web site. The crucial issue concerns the details of how you actually spin the mobile site off the full desktop site.

You want to offer your users the best navigation experience for the device they have—whether it's a regular PC, a smartphone, a tablet or an old-fashioned cell phone. (I'm skipping over an emerging family of large-screen devices such as smart TVs, which will require their own ad hoc rendering in the near future.) You also want to present users with a set of pages that can be best viewed on the requesting device. How would you select the markup that best fits a given device? In my opinion, this is a sore point and deserves a bit of attention and analysis to keep you from believing the myths and ignoring real-world conditions.

The One-Site-Fits-All Myth

Most developers recognize the need to have a mobile and a desktop Web site that look different from one another. Screen sizes are different, computing power is different and use cases especially are going to be fewer and largely different in a mobile site. Nevertheless, the idea of building just one Web site that automatically adjusts to different resolutions is attractive to many developers. But it also may be a dangerous route with more cons than pros. Realistically, the pro/con balance is going to depend on the nature of the application: the more sophisticated the site, the more cons and the fewer pros. What does it mean, exactly, to have a single site that adapts to multiple devices? Many developers have bad memories of when—only a few years ago, if not still today—building a Web site was a matter of extricating yourself from the mess of different (desktop) browsers and different rendering capabilities. For this reason, most developers welcome the idea of a single site that users can visit with any device (laptops, tablets, phones, e-readers, smart TVs and so on), but with a different experience.

The point is that with desktop browsers most of the differences are in the rendering space. Mobile devices, instead, add one more dimension to the problem: different capabilities. So the goal should be to build a Web site that automatically serves the most appropriate *experience* to different devices rather than an over-emphasis on the most appropriate *rendering* of the content.

Sound like a subtle difference? Well, client-side adaptation versus server-side adaptation of mobile content is an open point these days.

To me, the idea of just one Web site that serves the same content to which the browser adjusts according to the device being used just doesn't work as a rule—but it might work as an exception. The one-site-fits-all myth builds on top of the browser capability of dynamically selecting the most appropriate CSS stylesheet given some physical device characteristics. Let's find out more.

CSS Media Queries

Introduced with CSS3, media queries are a browser feature meant to simplify the design of sites that might be consumed through devices of different screen sizes. It should be noted that the term *device* in this specific context also includes laptops and desktop PCs. For this reason, the screen size might range from the 24 inches of a desktop monitor to the 3 inches of most smartphones.

The idea behind CSS media queries is that you create one Web site with a single set of functions and then apply different CSS styles to it by loading a different stylesheet for different media. The great improvement brought by CSS3 is that the screen page medium now can be restricted to all devices that match a set of given rules. Here's an example of a media query:

```
<link type="text/css"
  rel="stylesheet"
  href="downlevel.css"
  media="only screen and (max-device-width: 320px)">
```

Once placed in a Web page (or ASP.NET MVC view), the preceding markup links the downlevel.css file only if the page is viewed through a browser with a width of 320 pixels or fewer. Using a few of the preceding elements lets you optimize the rendering of any page to devices of different screen size. Nicely enough, the stylesheet rule is applied dynamically and also adjusts the page content when the user resizes the browser window on the desktop.

It should be noted that in CSS media queries, there's no explicit check on the type of browser, whether mobile or desktop; all that matters is the effective width of the screen. However, when you use a media query to filter out browsers with a width larger than 320 pixels, you're surely selecting all mobile devices and cell phones with that screen size.

The keyword *only* in the media query should be added for the sole purpose of hiding the media query statement from browsers that don't support media queries. These browsers, in fact, don't understand the media type and blissfully go ahead with their rendering. It's a common thought these days that by simply adding media queries to a site you make it ready for mobile clients. My opinion is slightly different. CSS media queries help make the page content more mobile-friendly, but they don't help in any way in optimizing the mobile site for other critical aspects such as the number of HTTP requests per page and the amount of data downloaded.

Furthermore, media queries can discern only a limited number of browser properties. The full documentation about media queries can be found at bit.ly/yOuW7q.

CSS media query properties tell you something about the device—but not all you may need to know. For example, you have

Powerful Tools for Developers

v4.5!

High-Performance PDF Printer Driver

CITRIX ready

- Create accurate PDF documents in a fraction of the time needed with other tools
- WHQL tested for all Windows 32 and 64-bit platforms
- Produce fully compliant PDF/A documents
- Standard PDF features included with a number of unique features
- Interface with any .NET or ActiveX programming language

v4.5!

PDF Editor for .NET, now Webform Enabled

- Edit, process and print PDF 1.7 documents programmatically
- Fast and lightweight 32 and 64-bit managed code assemblies for Windows, WPF and Web applications
- Support for dynamic objects such as edit-fields and sticky-notes
- Save image files directly to PDF, with optional OCR
- Multiple image compression formats such as PNG, JBIG2 and TIFF

New!

PDF Integration into Silverlight Applications

- Server-side PDF component based on the robust Amyuni PDF Creator ActiveX or .NET components
- Client-side C# Silverlight 3 control provided with source-code
- Optimization of PDF documents prior to converting them into XAML
- Conversion of PDF edit-boxes into Silverlight TextBox objects
- Support for other document formats such as TIFF and XPS

OCR Module available with royalty-free licensing!

The new OCR Module from Amyuni enables developers to:

- Convert non-searchable PDF files into searchable PDFs
- Create searchable PDF documents out of various image formats such as multi-page TIFF, JPEG or PNG while applying text recognition
- Compress image based PDF documents using high compression JBIG2 or more standard CCITT, JPEG and PNG compression formats

The Amyuni OCR module is based on the Tesseract Library with the Amyuni PDF technology being used to process and create the PDF documents.

Learn more at www.amyuni.com

Windows 7

CERTIFIED FOR Windows Server 2008

More Development Tools Available at:

www.amyuni.com

USA and Canada
Toll Free: 1 866 926 9864
Support: (514) 868 9227
Info: sales@amyuni.com

Europe
Sales: (+33) 1 30 61 07 97
Support: (+33) 1 30 61 07 98
Customizations: management@amyuni.com

AMYUNI Technologies

no clue about the OS, whether the device is mobile or not, whether it's a tablet or a smart TV or perhaps a bot. Likewise, with CSS media queries, you have no idea if the requesting browser supports Asynchronous JavaScript and XML and Document Object Model manipulation, how it deals with HTML5 and jQuery Mobile, or which type of markup it prefers.

Being a CSS feature, media queries are processed by the browser's CSS subsystem. It turns out that media queries can only select a set of element styles that hide or resize some elements that are either too big or that can be reduced on a small screen. With CSS media queries, you still pay the costs of downloading and keeping these elements in memory even when they aren't displayed. You can use some JavaScript in the pages to programmatically download or configure images. In this way, heavy elements can be managed in a more optimized manner. This extra work, however, is up to you.

Last but not least, media queries require a browser that supports CSS3. This means CSS media queries work great on most smartphones but not on older devices—and not even on Windows Phone 7. You can opt for an all-browser solution for media queries by getting a jQuery plug-in from bit.ly/JcwwFY. However, in this case as well, there's no guarantee that the mobile browser where you might be using this plug-in can really run jQuery.

Overall, CSS media queries are not a technology specifically aimed at mobile development. However, the flexibility of CSS media queries makes them compelling to use to serve different devices with a single codebase. They might be a solution in some cases, but CSS media queries are not the way to go for just any scenario.

Multi-Serving

The server-side route to mobile development is based on the idea that the Web site receives a request, gets the user agent and uses that information to figure out the effective capabilities of the browser. Next, armed with that knowledge, the Web site intelligently decides what would be the most appropriate content for the requesting device.

How would you find out about browser capabilities? At present, the most effective strategy seems to be using a Device Description Repository (DDR)—namely a database that stores nearly all possible properties of nearly all devices and that is constantly updated as new devices hit the market. In what way is a DDR different from the ASP.NET Request.Browser object? In a way, the Request.Browser object can be seen as a simple DDR that contains good information about desktop browsers but not particularly accurate information about mobile devices.

The number of different models of mobile devices is in the order of thousands and growing. The properties that a developer can find interesting for each device profile in a particular use case might vary significantly, but if you take the union of all of them you easily reach a few hundred. What options do you have today for getting accurate device information?

The DDR called Wireless Universal Resource File (WURFL) is an XML database that currently contains more than 15,000 profiles of mobile devices and matches half a million user agent strings. Each profile contains more than 500 capabilities. WURFL is an open source framework released under a classic dual-licensing scheme, AGPL v3 and commercial. WURFL is also available as a cloud service and some basic plans, limited to a few properties, are free. For more information, visit scientiamobile.com. WURFL is not the only DDR out there, but WURFL can be considered the de facto standard. Among other things, WURFL is employed in the mobile platform of Facebook and Google. WURFL has APIs for PHP, Java and the Microsoft .NET Framework, and it plugs smoothly into ASP.NET applications via NuGet.

> Having just one Web site that morphs into different rendering for different devices is your goal, but it won't be attained free with just the magic of a few CSS files.

As far as the ASP.NET platform is concerned, another interesting DDR solution is 51Degrees (51degrees.codeplex.com). 51Degrees originally relied on WURFL as the source for device information, but it has been relaunched recently with a new internal architecture and vocabulary. 51Degrees is a purely commercial initiative, but it offers a free version of the framework limited to four properties: isMobile, ScreenPixelWidth, ScreenPixelHeight and LayoutEngine, which refers to the browser rendering engine. As far as their free offerings are concerned, WURFL and 51Degrees are nearly equivalent, apart from the licensing.

Understanding Wants and Needs

Mobile development is mostly about understanding what your customers want and need. This is a crucial point and is solved when you have a well-chosen selection of use cases. For mobile sites in particular, I estimate that getting a good selection of use cases to implement is the largest share of the work. Beyond this, mobile development requires recognizing that mobile browsing is different. This means that having just one Web site that morphs into different rendering for different devices is your goal, but it won't be attained free with just the magic of a few CSS files. CSS contains almost no logic, and you probably want to have logic involved in deciding which content to serve to a given class of devices.

In the next article, I'll discuss how to switch from a desktop to a mobile site, thus delivering two sites but possibly one experience to the user. Later on, I'll delve deeper into browser segmentation and alternatives to media queries. ■

DINO ESPOSITO *is the author of "Architecting Mobile Solutions for the Enterprise" (Microsoft Press, 2012) and "Programming ASP.NET MVC 3" (Microsoft Press, 2011), and coauthor of "Microsoft .NET: Architecting Applications for the Enterprise" (Microsoft Press, 2008). Based in Italy, Esposito is a frequent speaker at industry events worldwide. Follow him on Twitter at twitter.com/despos.*

THANKS to the following technical expert for reviewing this article: *Steve Sanderson*

telerik

The 1st commercial suite for Windows 8!!!

RadControls for Metro

Build for Windows 8 using your language of choice.

XAML HTML

www.telerik.com/metro

telerik

deliver more than expected

JULIE LERMAN

Create and Consume JSON-Formatted OData

In my June column, "Data Bind OData in Web Apps with Knockout.js" (msdn.microsoft.com/magazine/jj133816), I had some fun with Knockout.js and OData. As I learned how to data bind Knockout to the results of the OData service, I also found out more about OData and JSON than I had previously known. In this month's column, I'll shift my focus to consuming JSON from OData and creating JSON-friendly WCF Data Services.

> OData is a specification for ensuring that data service consumers can rely on a consistent experience from the services they consume.

I'll show you how to consume JSON directly from JavaScript, how to take advantage of the OData JavaScript SDK (called datajs) and how to ensure your feed is flexible enough to support any style of client-side coding that will need JSON.

OData is a specification for ensuring that data service consumers can rely on a consistent experience from the services they consume. One of the rules of the spec is that OData results are output by default in ATOM format (which is a specific way of formatting XML), and that it can also output results in JSON format.

As the OData specification evolves, it introduces new features that you may want users of your service to leverage. After I discuss the various ways to consume and create JSON in OData feeds, I'll make sure you understand how upcoming changes to the OData spec will affect OData and what you can do today to be prepared. I'll begin by exposing a Customer with the following schema in my service:

```
public class Customer
{
  public int Id { get; set; }
  public string FirstName { get; set; }
  public string LastName { get; set; }
  public string AccountNumber { get; set; }
}
```

By default, a Customer result from my service would be ATOM data and look (as formatted by a browser) as shown in **Figure 1**.

Code download available at code.msdn.microsoft.com/mag201207DataPoints.

The same result output as JSON would be:

```
{"d":[
{"__metadata":{"id":"http://localhost:43447/DataService.svc/Customers(1)",
               "uri":"http://localhost:43447/DataService.svc/Customers(1)",
               "type":"DataAccessMSDNJuly2012.Customer"},
  "Id":1,
  "FirstName":"Julie",
  "LastName":"Lerman",
  "AccountNumber":"A123"}
]}
```

If you're working with the raw ATOM or JSON results, the format of the output can make a big difference in the ease of coding. For example, if you're consuming the feed in JavaScript, it's much easier to work directly with a JSON object than have to parse XML.

There are two ways to ensure your results come back in JSON format: You can either specify application/json in the request header or you can add an OData query parameter. When composing requests in Fiddler to test, it's pretty easy to add a header, as shown in **Figure 2**.

You can also add a header in JQuery when you build your request, but thanks to a couple of other options, you don't have to go to that length. One of these options is to use datajs, the JavaScript library for OData, which provides many other benefits for coding against OData as well. The other option is to use the OData query parameter, $format.

> If you're working with the raw ATOM or JSON results, the format of the output can make a big difference in the ease of coding.

Get JSON by Default with datajs

You can download datajs from datajs.codeplex.com. As of the time of writing, the current version is 1.0.3.

If you're using Visual Studio, you can add datajs directly to your project using the NuGet package manager. NuGet will add a scripts folder to your project and put the current version of datajs (datajs-1.0.3.js) and its related min script files in that folder.

The script library provides a class called OData that makes it easy to read and write from OData-compliant feeds, and provides

```
- <entry>
    <id>http://localhost:43447/DataService.svc/Customers(1)</id>
    <category
      scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme"
      term="DataAccessMSDNJuly2012.Customer"/>
    <link title="Customer" href="Customers(1)" rel="edit"/>
    <title/>
    <updated>2012-04-17T14:00:21Z</updated>
  - <author>
      <name/>
    </author>
  - <content type="application/xml">
    - <m:properties>
        <d:Id m:type="Edm.Int32">1</d:Id>
        <d:FirstName>Julie</d:FirstName>
        <d:LastName>Lerman</d:LastName>
        <d:AccountNumber>A123</d:AccountNumber>
      </m:properties>
    </content>
  </entry>
```

Figure 1 **Results as ATOM**

other features. Although you can easily insert a header into the request via the OData.read function, it's not necessary. By default, datajs will automatically add the header because if you're using this library, it's likely you want JSON.

> Not everyone needing JSON will be using the jsdata library.

Here's some JavaScript that calls OData.read, passes in the Uri that represents my data service query, and specifies what to do with the results (in this case, I'm storing the results in a variable named customer):

```
<script src="Scripts/datajs-1.0.2.js" type="text/javascript"></script>
<script type="text/javascript" charset="utf-8">
  var customer;
  OData.read({ requestUri:"http://localhost:43447/DataService.svc/
Customers?$top=1"
            },
            function (data, response) {
              customer = data.results[0];
            },
            function (err) {
              alert("Error occurred: " + err.message);
            });

</script>
```

As I debug through this script, I can see that the result of the query is an object, as shown in **Figure 3**.

With Fiddler running to capture HTTP traffic, I can see that the Accept header, specifying application/json, was part of this request.

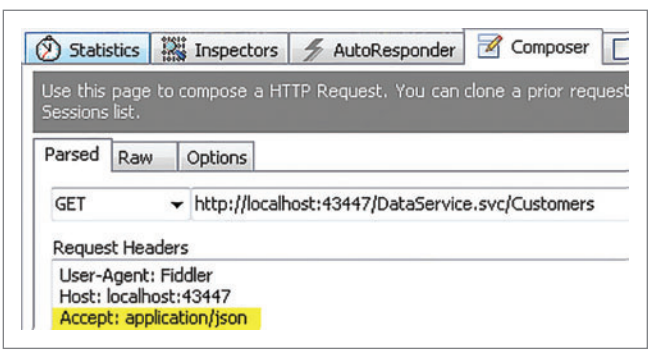


Figure 2 **Specifying JSON in the Request Header**

If you're a PHP developer, you should check out the related library, OData SDK for PHP (odataphp.codeplex.com). Just as the datajs library does for JavaScript developers, the PHP SDK ensures that the results are relevant. But in this case, it does so by requesting ATOM format and then materializing the resulting ATOM data into PHP objects.

Requesting JSON with the $format Parameter

Not everyone needing JSON will be using the datajs library. But that doesn't necessarily mean you have to use the request header to ask for JSON. The OData spec also has a query parameter, $format, that accepts json as one of its values.

Here's the modified Uri that requests JSON directly:

```
http://localhost:43447/DataService.svc/Customers(1)?$format=json
```

You can add the $format parameter to any query. Here I'm combining the format request with a filter:

```
http://localhost:43447/DataService.svc/Customers?$filter=FirstName eq
'Julie'&$format=json
```

Forcing a Data Service to Honor the $format Parameter

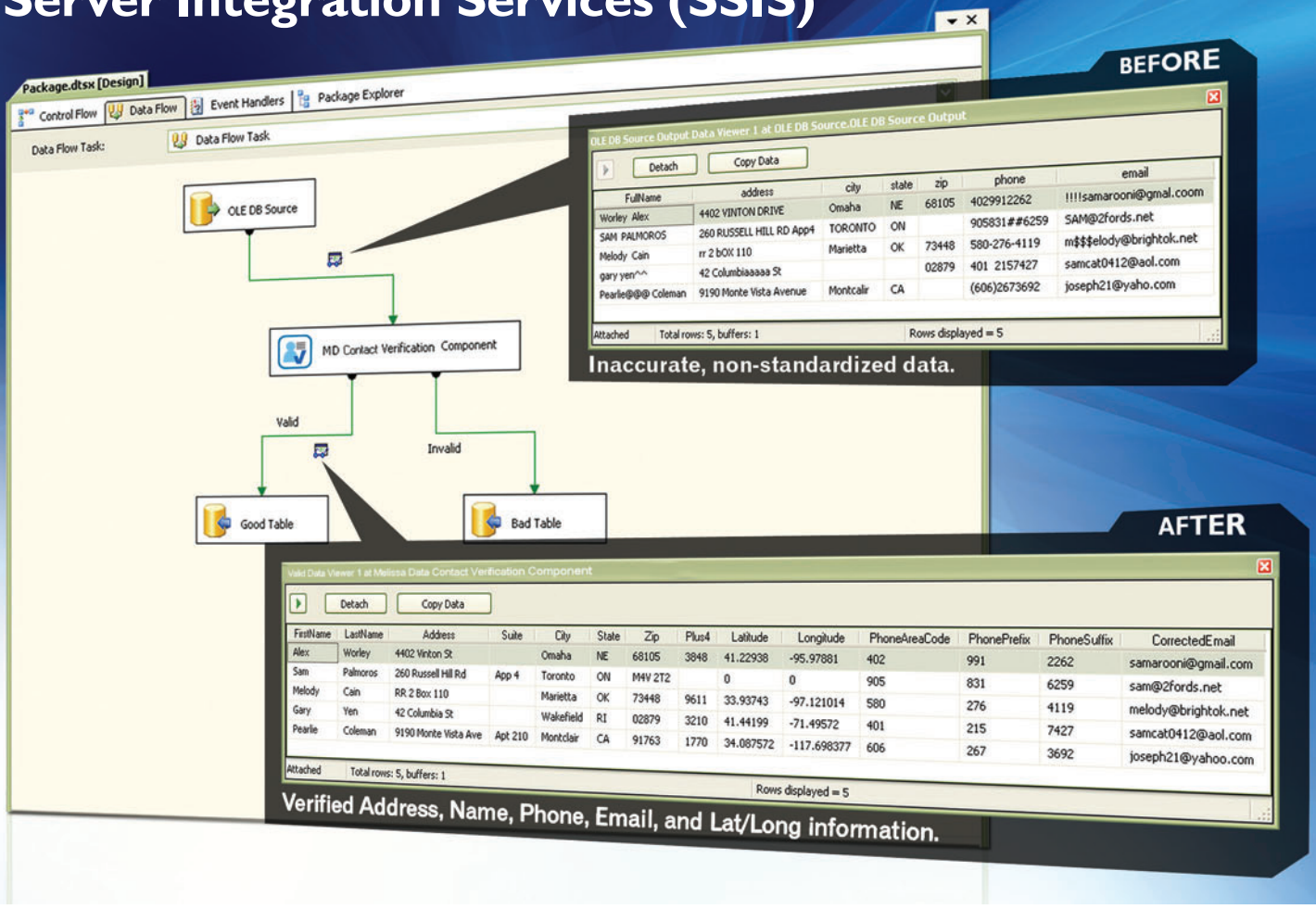
Although using this format parameter to request JSON is part of the OData specification, not every data service knows how to process the parameter. In fact, when you create a .NET WCF Data Service, it returns ATOM by default. So while the service will accept the $format=json parameter, it will still return ATOM in the response. But some developers on the OData team have shared a simple solution (available at archive.msdn.microsoft.com/DataServicesJSONP) you can add into your application that enables WCF Data Services to process the $format command and return JSON.

> The OData spec also has a query parameter, $format, that accepts json as one of its values.

The extension is provided in the form of a class named JSONPSupportInspector and an attribute named JSONPSupportBehavior. Both classes leverage logic from System.ComponentModel. You can add the classes directly to your project or create an assembly from them to reference. Once your WCF Data Service has access to these classes, all you need to do is add the JSONPSupportBehavior attribute to the service class, like so:

```
[JSONPSupportBehavior]
public class DataService : DataService<SalesContext>
{
  public static void InitializeService(DataServiceConfiguration config)
  {
    config.SetEntitySetAccessRule("Customers", EntitySetRights.All);
    config.SetEntitySetAccessRule("Orders", EntitySetRights.All);
  }
```

Data Cleansing Tools for SQL Server Integration Services (SSIS)



Contact Verification Component

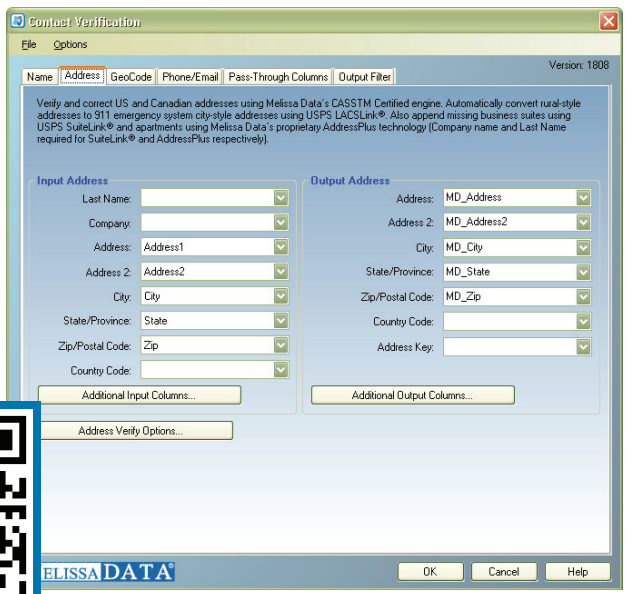
Parse, validate, correct and geocode addresses, phone numbers, email addresses, and full names.

SmartMover Component

Identify and update the addresses of people or businesses that have moved in the last 48 months.

MatchUp Component

Use advanced fuzzy matching algorithms and a custom lexicon to accurately dedupe contact data.



An easy-to-use interface enables you to easily select the fields from your input data and direct them to the correct fields in your output data.

Data Quality Components for SSIS

SCAN TO WATCH A SHORT VIDEO

For a Free Trial, Go to MelissaData.com/msdn-dqc
Call 1-800-MELISSA (635-4772)

MELISSA DATA®
Your Partner in Data Quality

With this attribute in place, my service will respond to the $format=json parameter when I add it into the query, and my service will return JSON.

I'm writing this column on the heels of the April 2012 release of WCF Data Services 5, which still requires that you explicitly add the JSONP support to your data services. If you'd like to see this wrapped into the WCF Data Services API in the future, you can add a vote to the team's UserVoice Feature Suggestions at bit.ly/ImeTQt.

JSON Support and OData Versions

The OData specification is currently at version 2 and evolving. Work on version 3 is underway, with beta documentation already available at OData.org. By default, your WCF Data Service will target version 1. Unfortunately, the default won't work if you want to use an OData version 2 feature, such as server-side paging, that's available in WCF Data Services. I've added the SetEntitySetPageSize configuration to my service to demonstrate:

```
public static void InitializeService(DataServiceConfiguration config)
  {
    config.SetEntitySetAccessRule("Customers", EntitySetRights.All);
    config.SetEntitySetAccessRule("Orders", EntitySetRights.All);
    config.SetEntitySetPageSize("Customers", 3);
  }
```

With this version 2 feature in place, the service will throw an exception and tell you that you have to specify MaxProtocolVersion greater than version 1. Here's the error associated with the SetEntitySetPageSize configuration:

"Server-side paging cannot be used when the MaxProtocolVersion of the data service is set to DataServiceProtocolVersion.V1."

To correct the problem, it's easy enough to set the version in the service code:

```
public static void InitializeService(DataSeviceConfiguration config)
  {
    config.SetEntitySetAccessRule("Customers", EntitySetRights.All);
    config.SetEntitySetAccessRule("Orders", EntitySetRights.All);
    config.SetEntitySetPageSize("Customers", 3);
    config.DataServiceBehavior.MaxProtocolVersion =
DataServiceProtocolVersion.V2;
  }
```

And, in this case, it's still possible to request JSON output either in the request header or using the $format parameter in the Uri.

However, you may want to design your service to be forward-compatible, setting the MaxProtocolVersion to DataServiceProtocolVersion.V3. But in version 3, the JSON format will be changing to a more streamlined output format called JSON Light (bit.ly/JrM6RQ). This can be a big problem if your client application is not expecting the streamlined JSON format. From the perspective of the consuming application, it will appear that the request for JSON has been ignored because ATOM will be returned and not JSON.

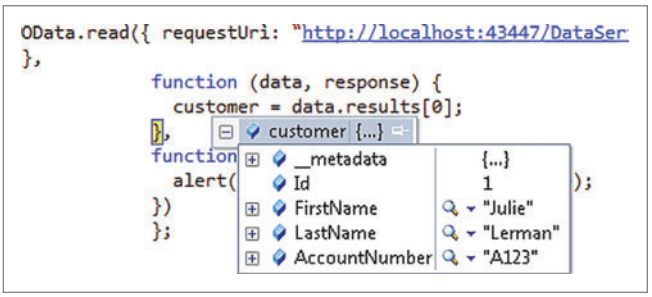


Figure 3 **Debug View of JSON Result**

That's because your client must either explicitly request the more verbose format or specify the OData version to target. Without either of these rules satisfied, the service will return ATOM. Therefore, you must do one or the other of the required tasks in order to get JSON results when the service is set to version 3.

Here's a request header modified to specifically request the verbose format of JSON:

```
Accept: application/json;odata=verbose
```

And here's an example of the header with a specific request that the OData response use version 2 formatting:

```
Accept: application/json
MaxDataServiceVersion: 2.0
```

What if you're using the $format=json parameter in the query instead of requesting JSON in the header? Again, the service won't comprehend the request and you'll get an error stating that the MIME type is not supported. In this case you must include the MaxDataServiceVersion in your request header.

> Thanks to its streamlined format, JSON is an increasingly important format for developers.

However, if you're using datajs, you don't need to worry. Starting with version 1.0.3 , the library is aware of the need for the version information and supplies it in its requests. With this in place, your service is ready for OData version 3 and consumers have the flexibility to request whichever version format they like.

JSON in the OData Pipeline

Thanks to its streamlined format, JSON is an increasingly important format for developers. You may recall that even some of the new document databases I wrote about in the November 2011 Data Points column, "What the Heck are Document Databases?" (msdn.microsoft.com/magazine/hh547103), store data using JSON or some twist on JSON.

If you check out last month's Data Points column, you'll find examples of reading and writing to data services using JSON and Knockout.js.

As we move into an age when more and more applications are disconnected, you'll appreciate the ability to consume OData as JSON (in either verbose or the more efficient new format). And if you're creating services, your consumers will certainly be grateful if you can make it easy for them to access your data this way. ■

Julie Lerman *is a Microsoft MVP, .NET mentor and consultant who lives in the hills of Vermont. You can find her presenting on data access and other Microsoft .NET topics at user groups and conferences around the world. She blogs at thedatafarm.com/blog and is the author of "Programming Entity Framework" (O'Reilly Media, 2010) and "Programming Entity Framework: Code First" (O'Reilly Media, 2011). Follow her on Twitter at twitter.com/julielerman.*

Thanks to the following technical expert for reviewing this article: *Alejandro Trigo*

JOSEPH FULTZ

Mixing Node.js into Your Windows Azure Solution

Node.js has been getting a lot of press as of late, and it's highly touted for its asynchronous I/O model, which releases the main thread to do other work while waiting on I/O responses. The overarching rule of Node.js is that I/O is expensive, and it attempts to mitigate the expense by forcing an asynchronous I/O model. I've been thinking about how it might be incorporated into an already existing framework. If you're starting from scratch, it's relatively easy to lay out the technology choices and make a decision, even if the decision is pretty much based on cool factor alone. However, if the goal is to perform a technology refresh on one part of a solution, the trick is picking something that's current, has a future, doesn't come with a lot of additional cost and will fit nicely with the existing solution landscape.

That's exactly what I'm going to demonstrate in this column. I'll take an existing solution that allows viewing of documents in storage but requires a shared access signature to download them. To that solution I'll add a simple UI using Node.js. To help with that implementation, I'll take advantage of some commonly used frameworks for Node.js. The solution, therefore, will include:

- Node.js—the core engine
- Express—a Model-View-Controller (MVC)-style framework
- Jade—a rendering and templating engine

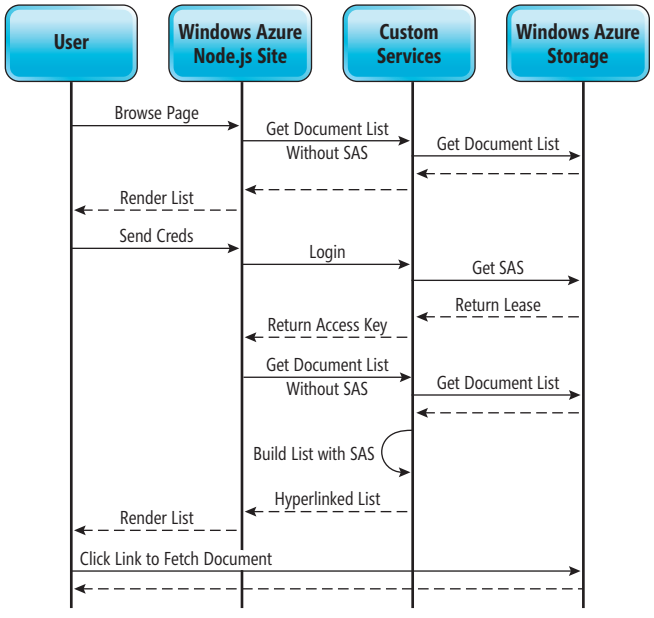


Figure 1 **The Request Sequence**

Together these three tools will provide a rich framework from which to build the UI, much like using ASP.NET MVC 3 and Razor.

Getting Started

If you're new to Node.js, it's probably best to start with the walkthroughs available from Microsoft at windowsazure.com/develop/nodejs. You'll also need to install the Node.js SDK for Windows Azure. Additionally, you'll probably want to spend a little time poking around Express (expressjs.com) and Jade (jade-lang.com). If you're new to these tools, you'll find some familiar concepts and a mix of familiar and unfamiliar syntax.

For this scenario, my existing services will do the work on the Windows Azure side, and the Windows Azure-hosted Node.js-based site will call those services to render a list of documents for access. Generally, it's a useful practice to put a layer of indirection between the client and the back-end services. This isolates the services from any interface changes, but the real value is often in the extra functional flexibility and the way you can include and exclude back-end service providers.

In the existing solution, as represented in **Figure 1**, the goal was to give a user access only if he was authenticated, which generates a Shared Access Signature (SAS). The idea was to grant read access to those who authenticated and then subsequently grant full Create, Read,

Figure 2 **Method for Getting the Shared Access Signature**

```
public string GetSharedAccessSignature()
{
  string sas = "";
  sas = (string) System.Web.HttpContext.Current.Cache.Get("sas");

  // If no SAS then get one
  if (sas == null)
  {
    // TODO: hardcoded container, move to config
    CloudBlobContainer container = blobClient.GetContainerReference("documents");

    // Ask the container for SAS passing in the a newly initialized policy
    sas = container.GetSharedAccessSignature(new SharedAccessPolicy()
    {

      SharedAccessStartTime = DateTime.Now,
      SharedAccessExpiryTime = DateTime.Now.AddMinutes(MaxMinutes),
      Permissions = SharedAccessPermissions.Read | SharedAccessPermissions.List

    });

    // Add to cache for reuse because this isn't a per-user SAS
    System.Web.HttpContext.Current.Cache.Add("sas", sas, null,
      DateTime.Now.AddMinutes(MaxMinutes), new TimeSpan(0,0,5,0,0),
      CacheItemPriority.High, null);
  }
  return sas
}
```

5 YEARS OF EXCELLENCE

XCEED
DataGrid
for WPF

Mature, feature-packed, and lightning-fast. The most adopted and trusted WPF datagrid.

IBM®
U2 SystemBuilder™

"IBM U2 researched existing third-party solutions and identified Xceed DataGrid for WPF as the most suitable tool. The datagrid's performance and solid foundation take true advantage of WPF and provide the extensibility required for IBM U2 customers to take their applications to the next level in presentation design and styling."

Vincent Smith
U2 Tools Product Manager at IBM

Microsoft®
Visual Studio® Team System 2010

"Using Xceed DataGrid for WPF in Microsoft Visual Studio System 2010 helped us greatly reduce the time and resources necessary for developing all the data presentation feature we needed. Working with Xceed has been a pleasure."

Norman Guadagno
Director of Product Marketing
for Microsoft Visual Studio Team System

XCEED Professional Themes for WPF

Theme your entire app in minutes. Flawless styles for all official WPF controls.

XCEED DataGrid for Silverlight

Incredible streaming technology. Speed up your app and say goodbye to paging.

XCEED Ultimate ListBox for WPF

NEW

The world's first streaming listbox. Simple, drop-in upgrade to the WPF listbox.

XCEED Ultimate ListBox for Silverlight

NEW

Fast and fluid, with ground-breaking streaming technology.

Visual Studio PARTNER

XCEED MULTI-TALENTED COMPONENTS

Update, Delete (CRUD) access to a particular document based on role and membership level. Here I'll focus solely on the read permissions.

Creating the Services

I'll mock up an authentication service to return an ID. The subsequent service call will retrieve a list of files. The Windows Azure Storage container I'm using ("documents") has the public permissions restricted. I want to provide a list of documents even if the user isn't authenticated, but I don't want an unauthenticated user to be able to open the files. My two calling signatures for the API I've created are:

```
http://[host]/admin/GetAccess?user=[user]&pwd=[password]
http://[host]/admin/files?accessId=[authId]
```

Of course, you'll want a more realistic auth service that doesn't use the querystring and does use SSL; I won't cover that piece of the solution here.

First, I need a method for creating a SAS (see **Figure 2**). I'll need this method shortly when I create the method that builds the document list.

For the most part, this is fairly typical Windows Azure Storage code until the call to GetSharedAccessSignature. There isn't a shared access policy here, so I need to pass information regarding when to start or stop allowing access and the type of permissions. All I want to provide via the SAS is the ability to read and list files. Also, because the SAS will theoretically be used by anyone who is authenticated, I add it to the cache for reuse in order to avoid collision and to reduce the amount of churn in generating access keys.

The service interface will be configured as a WebMethod:

```
[OperationContract]
[WebGet(UriTemplate = "Files?accessId={accessId}")]
List<BlobInfo> GetFiles(string accessId);
```

Note the use of the custom class BlobInfo—again, I'm using indirection. I have specific fields that I want to return and IListBlobItem doesn't necessarily represent them. So, I'll marshal the information returned from IListBlobItems into a list of my type, as shown in **Figure 3**.

It's important to note in **Figure 3** that I'm using the SAS if the user is authenticated in order to return a list that respects the access policy on the container.

With the Representational State Transfer (REST) service in place I can run a quick test via a browser window. By setting up the service interface this way, it becomes easy for me to mock authentication by using a well-known value until I have the for loop generating the list and the SAS running properly. Once that's done, the VerifyId(string) simply checks to see if I have a credential cached with a key equal to the accessId. **Figure 4** shows a list returned without being authenticated. Because the list returned by the service wasn't authenticated, the SAS value is set to nil. Thus, I can use the data to render the list, but I can't give a working link to the user, because there is no SAS.

Figure 5 shows the authenticated list, which does include the SAS.

It will be the job of the Node.js client to sort through what the service returns from an authenticated call and render the hyperlinks with the SAS postfixed to the URI. To help with that, I've provided a CombinedUri element so that the client needs to access only that one element. Finally, while the XML is great, because I'm working in Node.js, it makes sense to change the attribution on the interface to return JSON so that the service response can be directly consumed as an object:

```
[WebGet(UriTemplate = "Files?accessId={accessId}",
  ResponseFormat=WebMessageFormat.Json)]
```

Here's what the JSON output looks like:

```
[{"CombinedUri":"https:\/\/footlocker.blob.core.windows.net\/documents\/
AzureKitchen-onesheet.docx?st=2012-03-05T05%3A22%3A22Z&se=2012-03-05T05%
3A27%3A22Z&sr=c&sp=rl&sig=Fh41ZuV2y2z5ZPHi9OIyGMfFK%2F4zudLUOx5bg25iJas%
3D","Name":"\/documents\/AzureKitchen-onesheet.docx","Sas":"?st=2012-03-
05T05%3A22%3A22Z&se=2012-03-05T05%3A27%3A22Z&sr=c&sp=rl&sig=Fh41ZuV2y2z
5ZPHi9OIyGMfFK%2F4zudLUOx5bg25iJas%3D","Uri":"https:\/\/footlocker.blob.
core.windows.net\/documents\/AzureKitchen-onesheet.docx"}]
```

As noted, JSON is what we ultimately want here, as it's directly consumable within Express and Jade.

Figure 3 **GetFiles Implementation**

```
public List<BlobInfo> GetFiles(string accessId)
{
  List<BlobInfo> blobs = new List<BlobInfo>();
  CloudBlobClient sasBlobClient = default(CloudBlobClient);
  CloudStorageAccount storageAccount =
    CloudStorageAccount.FromConfigurationSetting(
    "StorageAccountConnectionString");
  string sas = default(string);

  if(accessId.Length > 0)
  {
    // For the mock just make a simple check
    if(VerifyId(accessId))
    {
      sas = GetSharedAccessSignature();

      // Create the blob client directly, using the SAS
      sasBlobClient = new CloudBlobClient(storageAccount.BlobEndpoint,
        new StorageCredentialsSharedAccessSignature(sas));
    }
  }
  else
  {
    sasBlobClient = storageAccount.CreateCloudBlobClient();
  }

  CloudBlobContainer blobContainer =
    sasBlobClient.GetContainerReference("documents");

  foreach (IListBlobItem blob in blobContainer.ListBlobs())
  {
    BlobInfo info = new BlobInfo();
    info.Name = blob.Uri.LocalPath;
    info.Uri = blob.Uri.AbsoluteUri;
    info.Sas = sas;
    info.CombinedUri = blob.Uri.AbsoluteUri + sas;
    blobs.Add(info);
  }

  return blobs;

}
```

```
<?xml version="1.0"?>
- <ArrayOfBlobInfo xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://schemas.datacontract.org/2004/07/WebRole1">
    - <BlobInfo>
        <CombinedUri>https://footlocker.blob.core.windows.net/documents/AzureKitchen-
          onesheet.docx</CombinedUri>
        <Name>/documents/AzureKitchen-onesheet.docx</Name>
        <Sas i:nil="true"/>
        <Uri>https://footlocker.blob.core.windows.net/documents/AzureKitchen-onesheet.docx</Uri>
      </BlobInfo>
    + <BlobInfo>
    + <BlobInfo>
  </ArrayOfBlobInfo>
```

Figure 4 **An Unauthenticated List**

DevExpress

Bridging tomorrow. **Today.**

The technology landscape is quickly changing. New platforms are emerging and more than ever we find great user experience and agile design at the center of the development conversation. Developers are looking to deliver next-generation user experiences on the desktop, on the Web or across a broad array of Touch-enabled mobile devices. DXv2 is the next generation of tools by DevExpress, ready to take on these new challenges using your existing skills & the technologies available today.

Your users are ready.
Ensure you're ready, too.

Download your
free 30-day trial at
DevExpress.com

DXv2

The next generation of inspiring tools. **Today.**

u e win asp wpf sl cr vcl

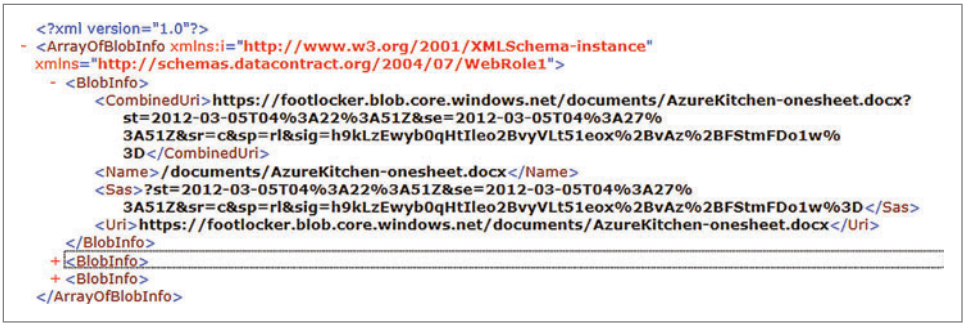


Figure 5 **An Authenticated List with the SAS**

Node.js UI

I've already set up Node.js, Express and Jade, so I'm ready to create the UI. I've gone through the process of getting deployment of Node.js roles up and running in Visual Studio, but that's a fairly detailed and completely manual process. So, because there's no tools integration for the Node.js part of this, I'll use Sublime Text 2 to do my editing and Chrome to debug (as described on Tomasz Janczuk's blog at bit.ly/uvufEM).

I should mention a few housekeeping items. For the uninitiated, the frameworks I'm employing provide some easy-to-use wrappers around certain functionality, MVC and a template-rendering engine:

- Restler for easy REST calls (think simplified WebClient)
- Express for the general MVC-style application framework
- Jade for template-rendering, similar to the Razor engine used in ASP.NET MVC

These are all considered modules in Node.js (like DLLs in .NET) and are generally installed from GitHub via Node Package Manager (NPM). For example, to install Restler, use the command "npm install restler" from within the project folder. This takes the place of manually installing a module and adding a reference to it in the project.

One last bit of information for the unfamiliar. You'll notice a lot of anonymous functions nested in other functions. My best advice is to just reformat the code enough to see the nesting as you work with it until you naturally parse it without having to reformat it. I'll attempt to nest my samples for readability as well as use screenshots from Sublime, which are nicely colored and help with readability.

I used the commands New-AzureService and New-AzureWebRole to create an app named AzureNodeExpress. I've also made a couple of other modifications. In server.js I added routes to get to the index page; the ASP.NET analog is the MapRoutes method used in MVC projects.

Figure 6 **Jade Template**

```
html
head
  title Default
body
  h1 File List Home Page
  br
  label Welcome #{username}

  form(method='post', action='/index')
    label Username:
      input(name='username', type='text')
    br
    label Password:
      input(name='password', type='password')
    br
    button(type='submit') Login
  h2 Files
  form
    table(border="1")
      tr
        td Name
        td Uri
        each doc in docList
          tr
            td #{doc.Name}
            td
              a(href=#{doc.CombinedUri}) #{doc.Name}
```

Server.js Modifications

Much like using statements in C#, I need to tell Node.js which libraries I'll be using. In Node.js, I'll set those references by assigning a variable the value of the return of the require('[lib name]') function. Once the references are set, I do a bit of configuration to set some of the engine variables (for example, setting "view engine" to "jade"). Of particular interest are the "view engine," router, bodyParser, cookieParser and session.

I'll skip some of the more mundane elements, but I do want to set up my routing. For the Get verb on my Index page I'll simply render the view directly:

```
app.get('/index',
  function(req, res){
    res.render('index.jade', {title: 'index'});
  }
);
```

However, for the Post verb I want to pass the handling over to the index model. To accomplish that I have to "bind" a defined method of the model:

```
app.post('/index', index.GetAccessKey.bind(index));
```

With the routing in place I'll need to set up both the view and the model.

The View—Index.jade

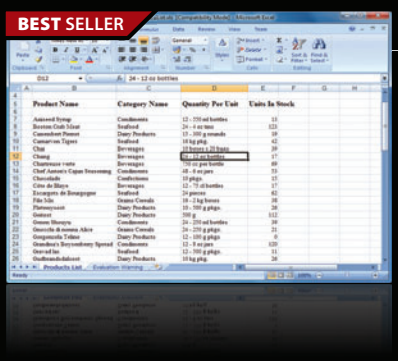
In a sense, I'm skipping from the beginning to the end by going from the controller to the view, but when working in an MVC style I like to create a simplified view to work against first. Jade syntax is basically HTML without the decoration of brackets. My entire Jade template is shown in **Figure 6**.

Of note here are the use of #{[var]} to reference variables and the table template with the loop in it, which is a kind of abbreviated foreach. I've arbitrarily named the list of items that I want to iterate over docList. This is important because in the index.js page where I ask Jade to render this view, I'll need to pass in the value for docList. Things are pretty basic here, because I'm just creating a developer UI—simple and plain.

The Model—Index.js

Having set up the runtime infrastructure in server.js and the final view template in index.jade, I'm left with the meat of the execution, which happens in index.js. Remember I set up a binding for app.Post to the index page. That binding will load and run the

BEST SELLER



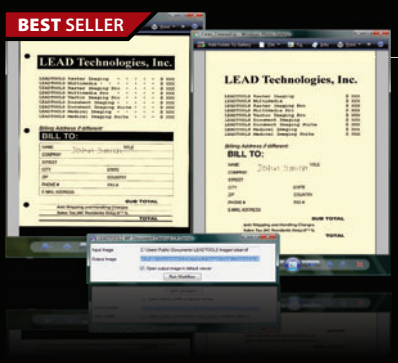
Aspose.Total for .NET | from $2,449.02

ASPOSE
Your File Format Experts

Every Aspose .NET component in one package.

- Programmatically manage popular file formats including Word, Excel, PowerPoint and PDF
- Add charting, email, spell checking, barcode creation, OCR, diagramming, imaging, project management and file format management to your .NET applications
- Common uses also include mail merge, adding barcodes to documents, building dynamic Excel reports on the fly and extracting text from PDF files

BEST SELLER



LEADTOOLS Document Imaging SDK V17.5 | from $2,245.50

LEAD TECHNOLOGIES

Add powerful document imaging functionality to Windows, Web & Mobile applications.

- Recognition Add-ons including OCR, Asian OCR, Arabic OCR, ICR, OMR and MICR
- Barcode Add-ons available: 1D Barcode and 2D Barcode
- PDF Add-ons available: Raster PDF Read and Write, and Advanced PDF
- Full featured, zero footprint Image Viewers and annotations for any device that supports HTML5

BEST SELLER

ActiveReports 6 | from $685.02

ComponentOne
a division of GrapeCity

Latest release of the best selling royalty free .NET report writer.

- Fast and flexible reporting engine
- Flexible event-driven API to completely control the rendering of reports
- Wide range of export and preview formats including viewers for WinForms, Web, Flash, PDF
- XCopy deployment
- Royalty-free licensing for Web and Windows, plus support for Azure

BEST SELLER

GdPicture.NET | from $3,135.02

GdPicture
imaging technologies

A full-featured document-imaging and management toolkit for software developers.

- Acquire, process, create, view, edit, annotate, compose, split, merge and print documents within your Windows & Web applications
- Read, write and convert vector & raster images in more than 90 formats, including PDF, PDF/A, TIFF , GIF, JPEG, PNG, JBIG2, WMF, BMP, WBMP, ICO, PCX, PNM, XPM, JPEG 2000, HDR, PSD, TGA, PICT, EXR, DDS, PPM, SGI, PBM, PGM, PFM, XBM, IFF and RAW camera files

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
30 Greyfriars Road
Reading
Berkshire
RG1 1PE
United Kingdom

Asia / Pacific Headquarters
ComponentSource
3F Kojimachi Square Bldg
3-3 Kojimachi Chiyoda-ku
Tokyo
Japan
102-0083

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

GSA Schedule
Contract GS-35F-0188R

```
index.prototype = {
    ShowDocs: function (req, res) {
        var uri = "http://jofultzazure.cloudapp.net//admin/files?accessId=" + accesskey;
        rest.get(uri).on('complete',
            function(result) {
                if (result instanceof Error) {
                    sys.puts('Error: ' + result.message);
                    this.retry(5000); // try again after 5 sec
                } else {
                    //sys.puts(result);
                    res.render ('index', {title: "Doc List",
                        layout: false,
                        docList: result,
                        username: req.BODY.username});
                }
            });
    },
    GetAccessKey: function (req, res) {
        var uri = "http://jofultzazure.cloudapp.net/admin/GetAccess?user=" + req.BODY.username;
        uri = uri + "&pwd=" + req.BODY.password;
        rest.get(uri).on ('complete',
            function(result){
                if (result instanceof Error) {
                    sys.puts('Error: ' + result.message);
                    this.retry(5000); // try again after 5 sec
                } else {
                    //sys.puts(result);
                    accesskey = result;
                    this.ShowDocs(req, res);
                }
            })
```

Figure 7 **Index.js Functions**

prototype I've created in index.js. To do this, I'll add functions to the index prototype, as shown in **Figure 7**. In essence, I'm creating a named function (for example, GetAccessKey) and defining an anonymous function as its execution body. In each of these functions I'll use the Restler module to simplify the REST calls I need to make.

The first call once the Post bind happens is to GetAccessKey, which simply takes the username and password I submitted via the form post, appends them to the URI as part of the querystring and uses Restler to do a Get. Remember, in Node.js all communication happens asynchronously, which is one of the reasons you'll see the proliferation of highly nested anonymous functions. Staying true to that pattern in the call to rest.get, I define an anonymous function that's executed once the request is complete. Without the error-handling code, the line simplifies to:

```
rest.get (uri).on('complete',
  function(result){
    accesskey = result;
this.ShowDocs (req, res);}
  )
```

Hopefully, that reformatting will help give a sense of what's going on. Once I've gotten the key from my service, I'll postfix it to the URI in the method to fetch the document list. Now things get a little different from the usual. In the anonymous function handling the return of the REST call to get the document list, I ask Jade to render the results for me:

```
res.render ('index', {title: "Doc List",
  layout: false,
  docList: result,
  username: req.BODY.username});
```

I noted earlier that in the template I created the variable name "docList." I now need to make sure I'm using that correct name. The call to res.render tells the Express framework to render the "index" resource and then passes parameters in via a list of colon- and comma-separated name:value pairs.

Runtime

If I attempt to browse to one of the files to download it, I'm presented with nothing. The Web page isn't found. You might expect an unauthorized error from Windows Azure Storage, but if you try to access something that's marked private, what's returned is that the resource doesn't exist. This is by design, and it's desirable because something that's "private" shouldn't exist to the public, even in concept. If a 401 error were returned instead, it would indicate that something is actually there, violating the security policy represented by private.

Because I've secured the storage location, no direct access is allowed. However, once I run the sample code, the story is a little different. I publish the application using the Windows PowerShell Publish-AzureService command, browse to the page and enter my credentials; I'm then presented with a list of links to the files (see **Figure 8**).

Because my service is brokering the calls to storage, I was able to list the files even though I can't list them directly. Also, because each link is postfixed with the SAS, when I click on it I'm prompted to Open or Save the target document.

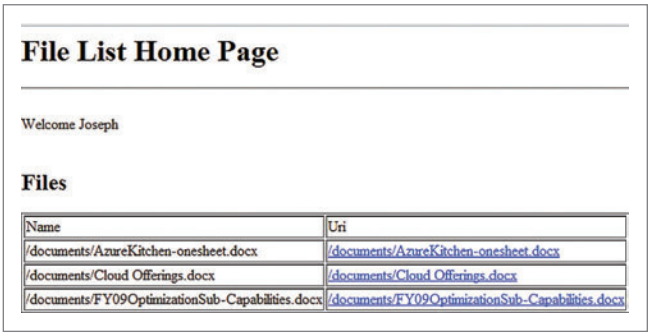


Figure 8 **Links to the Files**

Wrapping Up

If you're interested in new or trending technologies as a way to evolve your Windows Azure application, and you're a true believer of what's going on in the Node.js domain, Windows Azure has you covered—not only with hosting the solution, but also on the development side with options such as a client library for Node.js, hitting the REST API directly or through indirection as I demonstrated here. Development certainly would be a lot better and easier if Node.js had proper tooling support, and I'm sure we'll eventually see some sort of integration with Visual Studio if the popularity of Node.js continues to grow. ■

Joseph Fultz *is a software architect at Hewlett-Packard Co., working as part of the HP.com Global IT group. Previously he was a software architect for Microsoft, working with its top-tier enterprise and ISV customers to define architecture and design solutions.*

Thanks to the following technical experts for reviewing this article: *Bruno Terkaly*

Pragmatic Tips for Building Better Windows Phone Apps

Andrew Byrne

I love building apps for Windows Phone. I love creating code samples and writing about the Windows Phone platform as part of my day job, though I've had my ups and downs along the way. Throughout all of this work I've stumbled and picked myself up, and dug myself into holes, but climbed back out. What I want to do is smooth out this obstacle course for you with some pointers in this article. I'll talk about controls, User Datagram Protocol (UDP) Multicast, the Ad Control, isolated storage, tools and 256MB devices. So sit back, relax and enjoy.

Specify the TargetName when Using a HyperlinkButton

The HyperlinkButton control is a button control that displays a hyperlink. I use this all the time to help the user navigate my app, or the Web.

This article discusses:

- The HyperlinkButton
- Handling a tap-and-hold
- Dealing with UDP Multicast errors
- Ad Control
- Using IsolatedStorageSettings
- Using FileMode.Append
- The isolated storage explorer
- Marketplace Test Kit
- Using WPConnect
- Handling 256MB devices

Technologies discussed:

Windows Phone

In one of my apps I was using a HyperlinkButton to show some help information, and I wanted a hyperlink to take the user to more information on the Web. So I did the following (details omitted):

```
<HyperlinkButton NavigateUri="http://www.msdn.com">Help</HyperlinkButton>
```

Simple, right? I ran my app, navigated to the page containing my little help button and clicked on the link. Nothing happened. I then tried to debug the app and clicked on the link again. This time I got a NavigationFailed exception that said:

Navigation is only supported to relative URIs that are fragments, or begin with '/', or which contain ';component/'. Parameter name: uri.

The head scratching began. Reading the documentation a little more carefully revealed the problem. I had not set the TargetName property on the HyperlinkButton control. This attribute specifies the target window for the hyperlink. Not setting it meant I was actually using the default value of "", an empty string. This translates into the HyperlinkButton attempting to load the contents into the page in which the link was clicked. That can't work, because the page from where it came is not a browser. Here's what I should've done:

```
<HyperlinkButton NavigateUri="http://www.msdn.com"
  TargetName="_blank">Help</HyperlinkButton>
```

This loads the linked document into a new browser instance. Clicking on the link now worked.

The RichTextBox control can also display hyperlinks using the HyperLink inline element. This will have the same issue. Set the TargetName of the HyperLink or head scratching will ensue! For more information on the controls available for Windows Phone, see wpdev.ms/windowsphonecontrols.

Use DataContext to Reference a Bound Object in a ListBox

Sometimes you'll want to perform an action when the user taps and holds. For example, a user can tap and hold an item in a list.

Figure 1 **A Simple ViewModel**

```
public class SimpleViewModel : INotifyPropertyChanged
{
  private string _myText = "Initial Value";
  public string MyText
  {
    get
    {
      return _myText;
    }
    set
    {
      if (value != _myText)
      {
        _myText = value;
        NotifyPropertyChanged("MyText");
      }
    }
  }
  public event PropertyChangedEventHandler PropertyChanged;

  private void NotifyPropertyChanged(string PropertyName)
  {
    if (PropertyChanged != null)
    {
      PropertyChanged(this, new PropertyChangedEventArgs(PropertyName));
    }
  }
}
```

Determining which item is being held can be a little tricky. I'll illustrate using a simplified ViewModel as shown in **Figure 1**.

I want to expose a collection of these as shown here:

```
public ObservableCollection<SimpleViewModel> Items { get; private set; }
```

In the page's constructor I instantiate this collection and add some test data, as shown in **Figure 2**.

I want to display the data in **Figure 2** in a list on a page and add a little graphic to each item listed. I do this with standard XAML Binding to a ListBox control, as follows:

```
<ListBox ItemsSource="{Binding}">
  <ListBox.ItemTemplate>
    <DataTemplate>
      <StackPanel Orientation="Horizontal">
        <Image Source="duck.png" Height="60" Width="60"/>
        <TextBlock Text="{Binding MyText}"/>
      </StackPanel>
    </DataTemplate>
  </ListBox.ItemTemplate>
</ListBox>
```

The end result is a list of items with a cute image and the MyText value for each item, as shown in **Figure 3**.

Next, I want to bring up a context menu when the user taps and holds an item in the list. This isn't selecting an item, it's just tapping and holding that item. I want the context menu to change based on the current context—that is, the underlying data for whichever item I'm pressing.

To do this, I had to change the ListBox XAML to add an eventHandler for the Hold event, as follows:

```
<ListBox ItemsSource="{Binding}" Hold="ListBox_Hold">
```

How do I figure out what the data context is? This magic happens in the Hold event handler:

```
private void ListBox_Hold(object sender, GestureEventArgs e)
{
  SimpleViewModel data =
   ((FrameworkElement)e.OriginalSource).DataContext as SimpleViewModel;
  MessageBox.Show(data.MyText);
}
```

So, what's happening here? Although ListBox is the control handling the Hold event, the event is actually fired on one of the UI elements of ListBoxItem, such as the Image or the TextBlock. Because I didn't set a DataContext on each ListBoxItem, they inherit the DataContext from their parent UI element. So, by casting OriginalSource to FrameworkElement, we can access the DataContext, which is the underlying data object—or the SimpleViewModel in this case.

You Can't Use Multicast on a Cellular Connection

UDP Multicast isn't supported on a cellular connection. What does that mean? BeginJoinGroup begins the join operation to a Multicast group. If you call BeginJoinGroup when the phone is only connected to a cellular network, a SocketException will occur. You can catch this as shown in **Figure 4**.

Obviously, in the real world you would handle this exception with a little more panache than displaying a MessageBox. You improve the situation by using the SetNetworkRequirement extension method. Then at least you're declaring up front that you know what you're doing, but you'll still get the NetworkDown exception in this case, too. You'll find all the details about networking on Windows Phone at wpdev.ms/windowsphonenetworking.

Keep Your App Launch Time in Check

My first app was a paid app and I gave it a Trial mode so the user could try the key features before splashing out on the app. I was pleased with the number of downloads, and the conversion rate from Trial to paid was healthy, too. All I needed was big volume! While waiting for the floodgates to open, I thought I'd try shipping a free app, with some ads placed in it, to see what kind of return I would get on my investment.

The ad-based revenue model is a popular way for developers to monetize their development efforts. I initially hesitated to go this route in my own app development, mainly because the first app I wrote was targeted at young children, and I didn't believe in ads in an app for that target demographic. They get enough of that from TV. My other reason for shying away from this model was my ignorance. I didn't know how to advertise and I didn't know how to implement advertising in an app. On looking back, I had no reason to be so timid. Making your app ad-based and learning about

Figure 2 **Instantiating a Collection and Adding Test Data in the Page Constructor**

```
// Constructor
public MainPage()
{
  InitializeComponent();

  // Crete some test data
  Items = new ObservableCollection<SimpleViewModel>();
  Items.Add(new SimpleViewModel
  {

    MyText = "My first item",
  });

  Items.Add(new SimpleViewModel
  {
    MyText = "My second item",
  });

  Items.Add(new SimpleViewModel
  {
    MyText = "My third item",
  });

  this.DataContext = Items;
}
```

impressions, effective cost per thousand impressions (eCPMs) and ad units is a breeze, and the Microsoft Advertising SDK for Windows Phone is a shining example of just how accessible this world of advertising can be.

The Microsoft Ad Control for Windows Phone is built right into the Windows Phone SDK, and you can be up and running with it in no time. All you need to do is create an app id and ad unit in Pub Center, drop the Ad Control control onto your page, set the ApplicationId and AdUnitId properties in that control to the values you got from Pub Center, and you're done. You can find more information at wpdev.ms/adsdk. Although the Advertising SDK ships with the Windows Phone SDK, be sure to check for Advertising SDK updates at wpdev.ms/adsdkupdates because they can ship independently.

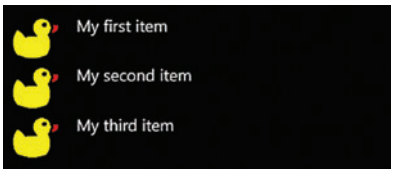


Figure 3 **A List of Items with an Image**

However, with all this power comes responsibility. You want to make sure that, as with all UI elements, loading the Ad Control doesn't impact the responsiveness of your app, particularly at app launch time. When someone taps your app tile, you want them in your app before they change their minds.

The Marketplace Test Kit, built right into the Visual Studio IDE, is a simple way to test your application's launch time. Using the monitored tests in the kit, you can start your application on your phone, play with it and then close it down. The kit then analyzes the metrics it gathered while your app was running and gives you feedback on launch time, peak memory usage and more. Failing any of these tests when you submit your app would prevent it from passing certification. The Technical Certification Requirements for Windows Phone include requirements for application launch time. At the time of this writing, the requirement was that an application must render the first screen within five seconds after launch. To see the most recent list of certification requirements, see wpdev.ms/techcert.

Always monitor your launch time and make sure it doesn't reach the limit of five seconds. Adding an Ad Control to your app's main page can also impact launch time. This is true for anything you do on that page, but the Ad Control is nice way to illustrate this point. Fortunately, there's a way to avoid a control such as the Ad Control impacting your launch time, as long as you don't mind a slight delay in the first ad appearing on the page. I don't mind, because responsiveness and a good user experience of my app trump those few extra ad impressions any day. The pattern I use is to delay loading the Ad Control. While I'm at it, I also make my code more robust by handling errors that may arise if an ad fails to load due to network issues, a lack of ads to actually load and so on.

Instead of placing the Ad Control into my page in XAML, I instead put a Grid on my page that will eventually contain the Ad Control:

```
<!—Ad Control is added dynamically in codebehind-->
<Grid Grid.Row="0" Height="90" x:Name="AdGrid"/>
```

In my codebehind for the page, I then add a member variable to the class for the Ad Control (don't forget to add a reference to Microsoft.Advertising.Mobile.UI in your project):

```
// Ad Control to add dynamically to AdGrid
private AdControl adControl = null;
```

In the page constructor I hook up the loaded event as follows:

```
public MainPage()
{
  InitializeComponent();
  this.Loaded += new RoutedEventHandler(MainPage_Loaded);
}
```

In the MainPage_Loaded method I add the following:

```
// Load Ad Control in Loaded event, so it won't count against the app's launch time
void MainPage_Loaded(object sender, RoutedEventArgs e)
{
  LoadAdControl();
}
```

The LoadAdControl method will be used to add a new Ad Control to the AdGrid that I created in XAML, as shown in **Figure 5**.

Note that the APPID and ADUNITID in **Figure 5** should be replaced by the ids that you get when you create an ad unit on pub center.

You'll notice that I also hooked up the AdRefreshed and ErrorOccurred events in this method. It's a good practice to have something in place for handling these events. Errors can occur when retrieving ads over the network, and it's important that you handle these cases. In my case, I simply want to display a piece of text, such as the name of my app. Others have been more adventurous and actually filled this spot with ads for their other Windows Phone Marketplace apps or used ads from an alternate provider. **Figure 6** shows how I handle an Ad Control error.

Figure 4 **Attempting to Join a UDP Multicast Group**

```
try
{
  // Make a request to join the group.
  _client.BeginJoinGroup(
    result =>
    {
      // Complete the join
      _client.EndJoinGroup(result);

    // Send or Receive on the multicast group
    }, null);
}
catch (SocketException socketException)
{
  if (socketException.SocketErrorCode == SocketError.NetworkDown)
  {
    MessageBox.Show("UDP Multicast works only over WiFi/Ethernet");
  }
}
```

Figure 5 **Adding an Ad Control to a Grid Programmatically**

```
private void LoadAdControl()
{
  // Create Ad Control if it doesn't exist already
  if (adControl == null || !AdGrid.Children.Contains(adControl))
  {
    adControl = new AdControl(APPID, ADUNITID, true)
    {
      Width = 480,
      Height = 80,
      VerticalAlignment = System.Windows.VerticalAlignment.Top
    };

    // Hook up some interesting events
    adControl.AdRefreshed += new EventHandler(adControl_AdRefreshed);
    adControl.ErrorOccurred +=
      new EventHandler<Microsoft.Advertising.AdErrorEventArgs>(
      adControl_ErrorOccurred);

    // Add it to the AdGrid
    AdGrid.Children.Add(adControl);
  }
}
```

What do you get when you build smartphones with world class hardware and software?

You get the newest Windows Phones.

You get Nokia Lumia!

See how you can create the Amazing Everyday at
http://developer.nokia.com/Lumia

NOKIA Developer

Figure 6 **Handling an Ad Control Error**

```
// If there's an error, display some text
void adControl_ErrorOccurred(object sender, Microsoft.Advertising.AdErrorEventArgs e)
{
  AdControl ad = (AdControl)sender;
  Dispatcher.BeginInvoke(() =>
  {
    // Hide the Ad Control
    ad.Visibility = System.Windows.Visibility.Collapsed;

    // Place something in its place. I chose to add a TextBlock
    // containing the name of my app. You could instead instantiate
    // an ad control from another provider, show static ads for your other
    // apps or do nothing.
    if (tbBrand == null)
    {
      tbBrand = new TextBlock()
      {
        Text = "My Application",
        Foreground = (Brush)Resources["PhoneForegroundBrush"],
        FontSize = (double)Resources["PhoneFontSizeMedium"],
        HorizontalAlignment = System.Windows.HorizontalAlignment.Center,
        VerticalAlignment = System.Windows.VerticalAlignment.Center,
        Margin = new Thickness(10)
      };

    }

    If (!AdGrid.Children.Contains(tbBrand))
    {
      AdGrid.Children.Add(tbBrand);
    }

    tbBrand.Visibility = System.Windows.Visibility.Visible;
  });
}
```

I added the AdRefreshed handler because I want to make sure the Ad Control is visible once it's up and running again, following an error (see **Figure 7**).

This is good practice and makes my Ad Control usage more robust, but what impact does it have on launch time?

To demonstrate, I created a new project using File | New Project | Windows Phone Application. I first dropped an Ad Control directly into XAML. I then ran the Marketplace Test Kit, having deployed my app to my phone. I ran my tiny app multiple times and looked at the launch times. It averaged out at 2.7 seconds. This is still an acceptable launch time. Next, I applied the delay load pattern I outlined previously. I deployed my device again through Marketplace Test Kit and tested it the same number of times as I had done in the first test case. The average launch time was now only 1.9 seconds. Say no more.

Use IsolatedStorageSettings to Store Simple Settings

I'm officially addicted to IsolatedStorageSettings as my preferred way to store application settings. Given its key-value pair structure, it should only be used for the simplest of settings. I typically use it to store things such as whether the user has enabled or disabled sound in my app, the number of objects to display per page, the last time data was updated and more. Simple data should go in here, and it's really easy to use. Check out the procedure in the MSDN Library docs that shows you how to create your very own Settings Page at wpdev.ms/settingspage. I promise, once you try this, you'll be hooked.

Figure 7 **Adding an AdRefreshed Handler**

```
// Make sure the Ad Control is visible if the ad was refreshed
void adControl_AdRefreshed(object sender, EventArgs e)
{
  AdControl ad = (AdControl)sender;
  Dispatcher.BeginInvoke(() =>
  {
    ad.Visibility = System.Windows.Visibility.Visible;
    if (tbBrand != null)
    {
      tbBrand.Visibility = System.Windows.Visibility.Collapsed;
    }
  });
}
```

When it comes to storing large amounts of data, you can't avoid writing to files in isolated storage. Keeping app launch time in mind, you should adhere to the guidelines to load data asynchronously (BackgroundWorker, an async Web call or whatever your async call du jour may be). I also adopt a pattern of only saving deltas to disk, and I do so more or less as soon as they occur. For more information on state on Windows Phone, check out bit.ly/oR96Ux.

Don't Use FileMode.Append on a File that Doesn't Exist

As mentioned previously, great care should be taken to optimize your data transfers to and from isolated storage. The responsiveness of your app should be paramount—otherwise you'll find the application downloads you fought so hard to get being uninstalled worldwide. In one of my creations I was able to store and load my data pretty snappily until I discovered that what went in wasn't necessarily coming out. I was losing data somewhere! I didn't spot it for a number of reasons. One reason was because of a long-lost Debug.WriteLine statement I had dropped into a catch block late one night. So there I was, thinking I was writing everything to isolated storage, when all the time the Error List was going bonkers. I thought I was appending data to a text file, but the Error List was delivering the message:

Operation not permitted on IsolatedStorageFileStream.

I'm not a big fan of displaying the wrong code in an article like this. I learned that habit from my math teacher back in high school, at a time when legwarmers, "Family Ties" and the Commodore 64 were all the rage. Yes, he would never show the wrong answer to a problem for fear it would stick. (Tangent: Strangely enough my teacher's initials were D.O.S.—I don't think I got the reference at the time!) So, here's what I *should* have been doing.

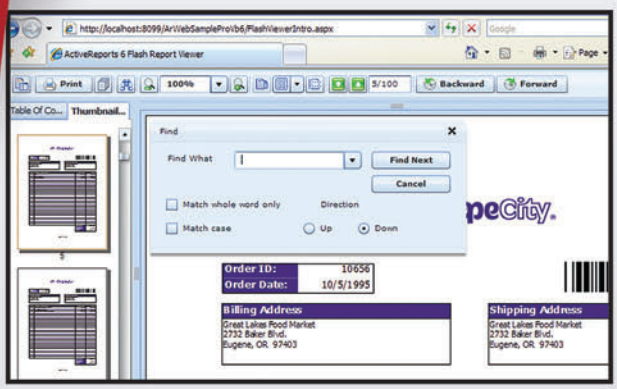
Figure 8 **Writing to a File in Isolated Storage**

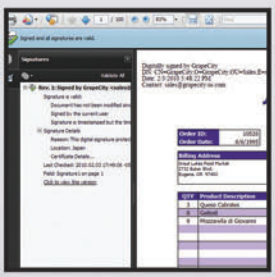
```
if (!_store.FileExists("mydata.data"))
{
  // Open for writing
  using (var textFile = _store.OpenFile("mydata.data", FileMode.CreateNew))
  {
    WriteRecordsToFile(textFile, dataString);
  }
}
else
{
  using (var textFile = _store.OpenFile("mydata.data", FileMode.Append))
  {
    WriteRecordsToFile(textFile,dataString);
  }
}

private void WriteRecordsToFile(IsolatedStorageFileStream textFile, string data)
{
  using (var writer = new StreamWriter(textFile))
  {
    writer.WriteLine(data);
  }
  writer.Flush();
  writer.Close()
}
```

<activereports>

platforms supported: windows forms · asp.net · silverlight · windows azure





ActiveReports 7
NEW VERSION

- Fast & flexible reporting engine
- Fixed page layout: create advanced reports
- Full report customization
- More controls: tables, matrices, charts, barcodes, lists & more

Follow the ActiveReports 7 road to success

It starts with a seamless Microsoft Visual Studio integration, advances with easy report creation and design from simple invoices and sales reports to complex tax forms and financial analysis reports, and continues with effortlessly exporting your .NET reports to popular formats. The road to success is wide open with endless customization options given the extensive APIs and integrated designers. Plus, maintenance is a cinch. Download your free trial and start delivering professional reports today.

FREE TRIAL @: http://c1.ms/reporting

"There's nothing like it right now in the market."

Dani Diaz
Developer Evangelist, Microsoft

EXPECT MORE.
GET MORE.

ComponentOne
a division of GrapeCity

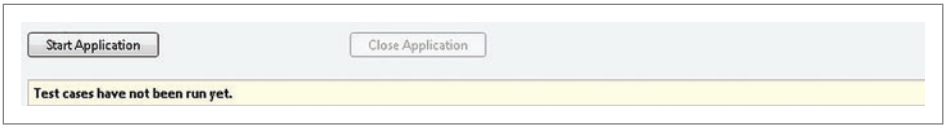


Figure 9 **Before Running Monitored Tests**

Figure 10 **A Monitored Test Failure**

Never attempt to call FileMode.Append on a nonexistent file. Create it, write to it and then append with subsequent transfers, as shown in **Figure 8**.

Use an Isolated Storage Explorer to Test Storage in Your App

The tooling space for Windows Phone is gathering momentum, and I'm thrilled to see high-quality tools filling gaps in the Microsoft offering. Mind you, these gaps aren't chasms, the size of which you'd find on other toolsets for other mobile platforms. No, these are gaps that Microsoft simply hasn't gotten around to filling yet. For example, Microsoft made headway by delivering the Isolated Storage Explorer command-line tool. Out in the wild you'll find wrappers around this that make life so much easier. One such tool is the Windows Phone 7 Isolated Storage Explorer found on CodePlex (wp7explorer.codeplex.com). Browse around and find the one that suits you, or stick with the command line. But get involved with isolated storage exploration and use the tools to—for example—drop test data into your app's isolated storage. It's so much easier than finding these bugs when you've already shipped.

Use the Marketplace Test Kit

The Marketplace Test Kit is a great tool for keeping you in check while you work on your app. Run directly from within the Visual Studio IDE, it offers sets of tests that do useful things such as verify that you have the right iconography for your app, detect the actual phone capabilities your app is using, report what your app launch time is like and so on. There are even manual tests that give you a good indication of what's been looked at during the app certification process. This is one of those features that every developer thinks about knocking out some night and posting on CodePlex for everyone to use. Microsoft got here first, which hopefully means there's a roadmap for this kit that will build on this foundation and make it rock even further down the road.

Apart from one failure due to a misunderstanding about how the Windows Phone Marketplace test team would test one of my apps, I've never failed application certification. I could chalk this up to my own genius or diligence, but really I got by with a little help from my Marketplace Test Kit friend, and you should too.

If I had one gripe—and this is something for you to watch out for—it's that the monitored tests fail pretty silently when you try to run them on the emulator. This failure simply takes you from **Figure 9** to **Figure 10** when you click "Start Application" and have the emulator selected as the target device.

I think it informs you all too subtly that you can only run these monitored tests on a real phone. A little more in-your-face notification on this one would be a nice improvement, in my opinion. You can read more about the Marketplace Test Kit at wpdev.ms/toHcRb.

Test Your Media App with WPConnect

When I began tinkering with media apps in general, I would hack for a couple of hours and then give up as soon as my first test run annoyed me by telling me I couldn't view the media library while my phone was connected to my PC. So, away I'd go into a more comfortable area of the platform, just shaking my head and wondering at how patient those media app developers really were. That was, until I found the Windows Phone Connect Tool (WPConnect). Yes, it took me a while, but when I did come across this nugget in the docs it made me smile ear to ear. What is it? Simply put, it enables you to debug media apps while connected to your PC. The instructions are very straightforward, so go check them out at wpdev.ms/wpconnect.

Test Responsiveness on 256MB Devices

Phones with 256MB of RAM exist. Because of their reduced memory capacity, you should monitor your own app's memory footprint and decide what to do if your app were to land on one of these phones. You could decide to detect whether a host device was a 256MB device and turn the bells and whistles down on your app. You could also come to the conclusion that your app simply wasn't built for these devices, and you'd rather take the hit of your app not showing up on these than take the risk of one-star ratings showing up because of the perceived performance problems of your app.

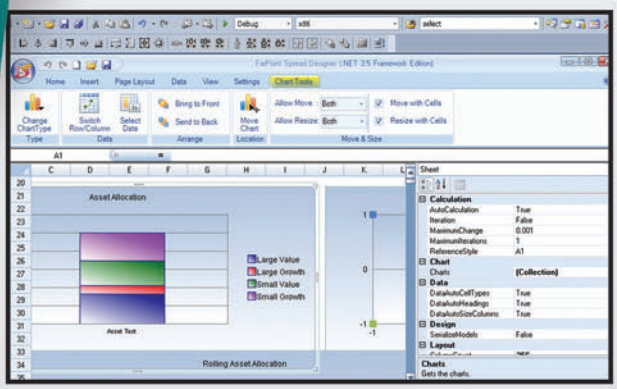
With the Windows Phone SDK 7.1.1, Microsoft helps you make informed decisions in this space. You can check what's known as the host device's ApplicationWorkingSetLimit to find out whether the phone is of the 256MB class. You can even opt out of making your app visible in the Windows Phone Marketplace to these devices. Check out the details at wpdev.ms/256devices.

In this article I took you on a journey through a broad set of issues I discovered while building apps. I also highlighted some tools that have helped me greatly. It should be clear that application responsiveness is a core value to uphold when building apps. I hope you found something that will help you on your Windows Phone app-building adventure. Windows Phone is a powerful platform and I still love to build for it and write for it. I enjoy browsing the Windows Phone Marketplace every day for all the new goodness that creative folks like you are delivering in a steady stream. If I haven't tried your app yet, I look forward to doing so. ■

ANDREW BYRNE *is a senior programming writer on the Windows Phone team. His knowledge and passion for software come from his 21 years in the software development industry working for many multinational organizations, as well as his own startup.*

THANKS to the following technical experts for reviewing this article: *Kim Cameron, Mark Hopkins, Robert Lyon, Nitya Ravi, Cheryl Simmons and Matt Stroshane*

<spread>

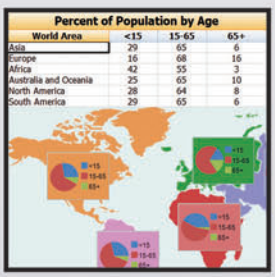


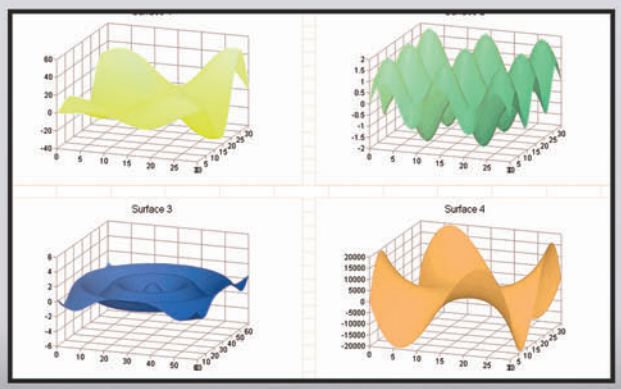
Spread's powerhouse components make it as easy as 1-2-3:

Import Microsoft Excel documents, preserving complete formatting

Interact with the data in Spread from within your application

Then export your spreadsheets to Microsoft Excel for portable distribution to your users





Where Easy & Excel Come Together in .NET – Spread.NET & Spread WPF-Silverlight

Ever been asked to add financial risk models, a budget analysis, or an engineering structural stress analysis to your application, and don't know how? With Spread.NET and Spread WPF-Silverlight you get the power, familiarity, and flexibility of Microsft Excel-compatible spreadsheets to harness in your business, engineering, and scientific applications. A vast Excel-functional library, data visualization, and enhanced file format support make these tasks effortless.

FREE TRIAL @: http://c1.ms/spreadsheets

"It's fast. It does what we need it to do. It just works."

Jeffrey Baron
VP App Software Development

EXPECT MORE.
GET MORE.

ComponentOne
a division of GrapeCity

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

YOUR MAP TO THE .NET DEVELOPMENT PLATFORM

CODE ON CAMPUS

Intense Take-Home Training for Developers, Software Architects and Designers

LIVE!
MICROSOFT HQ
Redmond, WA, August 6-10

PLATINUM SPONSOR

esri

SUPPORTED BY

Microsoft

Microsoft Visual Studio

msdn

Visual Studio MAGAZINE

Register Today Before Seats Sell Out!

Use Promo Code REDJULY

It All Happens Here!

Visual Studio Live! is returning to Microsoft Headquarters for the third year running! Developers, software architects and designers will connect for five days of unbiased and cutting-edge education on the Microsoft platform – all while getting an insider's look into the house that Bill built.

Topics include:

- Windows 8 / WinRT
- Cross Platform Mobile
- Silverlight / WPF
- Web
- Visual Studio / .NET
- SharePoint
- Cloud Computing and Services
- Data Management
- HTML5
- Windows Phone

Register today for Visual Studio Live! Redmond and become a more valuable part of your company's development team.

vslive.com/redmond

Scan this code to register and learn more about what Visual Studio Live! Redmond has to offer.

Redmond, WA

August 6-10

Microsoft Headquarters

PRODUCED BY

1105 MEDIA

MEDIA SPONSOR

THE CODE PROJECT
WWW.CODEPROJECT.COM

Check Out the 2012 Tracks for Visual Studio Live! Redmond

Cloud Computing and Services
Microsoft launched the Azure cloud computing platform in 2009 and continues to introduce major enhancements to it today. Even if you aren't ready for Azure yet, you owe it to yourself to become familiar with cloud computing and the services approach to development.

Data Management
With recent enhancements to SQL Azure and the release of SQL Server 2012, there's a lot of ground to cover in the data management sphere. New options, like Entity Framework Code First and Hadoop/Big Data require review and strategic evaluation. This track covers all of these technologies so you can integrate them into your environment sensibly and pragmatically.

Silverlight / WPF
This track is all about XAML in both WPF and Silverlight, and offers you the opportunity to build your skills and knowledge around this key smart client technology.

Web
Web development technologies continue to evolve and improve, offering rich and powerful capabilities for software developers. Today, Microsoft provides ASP.NET Web Forms and MVC, with traditional aspx and Razor page markup languages. These are supported by Visual Studio, WebMatrix, and Expression Web. In the near future, .NET 4.5 will provide substantial improvements to Web Forms, as well as other web technologies.

Windows Phone
Windows Phone (WP) version 7.5 now has more than 70,000 apps in its marketplace. Nokia's adoption of Windows Phone as its primary smartphone standard lends even more momentum to WP.

Cross Platform Mobile
Mobile clients are becoming a common way for users to interact with each other, their organizations, and their business applications. Mobile apps might be native client apps or mobile Web sites, and they often make use of cloud-based data and services.

HTML5
Learn more about HTML5, an industry-wide phenomenon. It is becoming indispensible to developing in almost any vendor environment and on every device form factor, along with JavaScript and jQuery. Its influence extends not just to the Web but to every major mobile platform, and even to Windows 8.

SharePoint
What's new in SharePoint 15? We'll have a special session to brief you on the latest features and developer hooks.

Visual Studio / .NET
Get the information you need to understand and leverage the power of .NET, Visual Studio and LightSwitch today and into the future.

Windows 8 / WinRT
Windows 8 and the Windows Runtime (WinRT) are coming. Are you ready for them? We'll prepare you to build Windows 8 Metro-style apps, and understand the finer points of WinRT that will make them great.

LIVE!
MICROSOFT HQ
Redmond, WA, August 6-10

Register at vslive.com/redmond
Use Promo Code REDJULY

VISUAL STUDIO LIVE! REDMOND AGENDA AT-A-GLANCE

Cross Platform Mobile	HTML5	Web	Visual Studio / .NET	Windows Phone	Data Management	Silverlight / WPF	Windows 8 / WinRT	Cloud Computing and Services

Visual Studio Live! Pre-Conference Workshops: Monday, August 6, 2012 *(Separate entry fee required)*

MWKS1 **Workshop: SQL Server 2012 for Developers** *Andrew Brust & Leonard Lobel*	MWKS2 **Workshop: HTML5 + Cloud - Reach Everyone, Everywhere** *Eric D. Boyd*	MWKS3 **Workshop: Services - Using WCF and ASP.NET Web API** *Miguel Castro*

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

MWKS1 **Workshop: SQL Server 2012 for Developers** *Andrew Brust & Leonard Lobel*	MWKS2 **Workshop: HTML5 + Cloud - Reach Everyone, Everywhere** *Eric D. Boyd*	MWKS3 **Workshop: Services - WCF and ASP.NET Web API** *Miguel Castro*

Visual Studio Live! Day 1: Tuesday, August 7, 2012

Keynote: To Be Announced

T01 **A Lap Around WPF 4.5** *Pete Brown*	T02 **MVC For Web Forms Developers: Comparing and Contrasting** *Miguel Castro*	T03 **Best Kept Secrets in Visual Studio 2010+ and .Net 4+** *Deborah Kurata*	T04 **Building Secured, Scalable, Low-latency Web Applications with the Windows Azure Platform** *Ido Flatow*	T05 **Microsoft Session To Be Announced**
T06 **Introduction to Silverlight 5** *Pete Brown*	T07 **Embracing HTTP with ASP.NET Web APIs** *Ido Flatow*	T08 **What's New in the .NET 4.5 BCL** *Jason Bock*	T09 **Tips & Tricks on Building Event Driven Architectures in Windows Azure with Service Bus** *Nuno Godinho*	T10 **Microsoft Session To Be Announced**

Lunch - Visit Exhibitors

T11 **WPF Validation - Techniques & Styles** *Miguel Castro*	T12 **IE 10 and HTML5: Tips for Building Fast Multi-Touch Enabled Web Sites** *Ben Hoelting*	T13 **Writing Asynchronous Code using .NET 4.5 and C# 5.0** *Brian Peek*	T14 **What's New in Windows Azure 1** *Vishwas Lele*	T15 **Microsoft Session To Be Announced**

Sponsored Break - Visit Exhibitors

T16 **Improve Your Code with Anonymous Types and Lambda Expressions** *Deborah Kurata*	T17 **Working with Client-Side HTML5 Storages** *Gil Fink*	T18 **Managing the .NET Compiler** *Jason Bock*	T19 **What's New in Windows Azure 2** *Vishwas Lele*	T20 **Microsoft Session To Be Announced**

Microsoft Ask the Experts & Exhibitor Reception

Visual Studio Live! Day 2: Wednesday, August 8, 2012

Keynote: To Be Announced

W01 **Metro Style Apps for Silverlight Developers** *Billy Hollis*	W02 **Introducing SQL Server Data Tools** *Leonard Lobel*	W03 **Team Foundation Service: TFS Goes to the Cloud** *Brian Randell*	W04 **Visual Studio for Mobile Apps on iOS, Android and WP7** *Miguel de Icaza*	W05 **Microsoft Session To Be Announced**
W06 **Windows 8 Metro Style Apps for the Enterprise** *Ben Hoelting*	W07 **T-SQL Enhancements in SQL Server 2012** *Leonard Lobel*	W08 **Top 10 Ways to Go from Good to Great Scrum Master** *Benjamin Day*	W09 **Using HTML 5 to build Mobile Web Sites and Apps** *Jon Flanders*	W10 **Microsoft Session To Be Announced**

Birds-of-a-Feather Lunch - Visit Exhibitors

W11 **WinRT for Web Devs** *Ben Dewey*	W12 **SQL Azure Intro and What's New** *Eric D. Boyd*	W13 **10 Ways to Get Your Project Started Right** *Benjamin Day*	W14 **Maps in Microsoft Silverlight 5** *Al Pascual*	W15 **Microsoft Session To Be Announced**

Sponsored Break - Visit Exhibitors

W16 **Windows 8, the Live SDK, and SkyDrive** *Rockford Lhotka*	W17 **Tips & Tricks to Build Multi-Tenant Databases with SQL Azure** *Nuno Godinho*	W18 **Storyboarding and User Feedback in Visual Studio 11** *Brian Randell*	W19 **What's New in WCF 4.5** *Ido Flatow*	W20 **Microsoft Session To Be Announced**

Evening Event @ Lucky Strikes Bellevue

Visual Studio Live! Day 3: Thursday, August 9, 2012

TH01 **Filling up Your Charm Bracelet** *Ben Dewey*	TH02 **Microsoft's Big Play for Big Data** *Andrew Brust*	TH03 **Win8 Azure Services** *Rockford Lhotka*	TH04 **Getting Started with Windows Phone 7** *Scott Golightly*	TH05 **Microsoft Session To Be Announced**
TH06 **Leveraging XAML for a Great User Experience** *Billy Hollis*	TH07 **Big Data/Hadoop/ MapReduce** *Andrew Brust*	TH08 **I'm Not Dead Yet! AKA The Resurgence of Web Forms** *Philip Japikse*	TH09 **Porting iOS Applications to Windows Phone 7** *Al Pascual*	TH10 **Microsoft Session To Be Announced**
TH11 **Blissful Separation of Concerns with MVVM in WPF** *Brian Noyes*	TH12 **Power View: Analysis and Visualization for Your Application's Data** *Andrew Brust*	TH13 **Building RESTful Services with WCF** *Jon Flanders*	TH14 **Cloud Application Performance Tips** *Stephen Burton*	TH15 **Microsoft Session To Be Announced**

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

TH16 **Building Extensible XAML Client Apps** *Brian Noyes*	TH17 **Consuming and Publishing Data for Any Platform with OData** *Eric D. Boyd*	TH18 **Controlling ASP.NET MVC 4** *Philip Japikse*	TH19 **To Be Announced**	TH20 **Microsoft Session To Be Announced**
TH21 **Maps in Silverlight, HTML 5 and WinRT** *Al Pascual*	TH22 **Not Just a Designer: Code First and Entity Framework** *Gil Fink*	TH23 **To Be Announced**	TH24 **Making Money with Windows Phone** *Scott Golightly*	TH25 **Microsoft Session To Be Announced**

Visual Studio Live! Post-Conference Workshops: Friday, August 10, 2012 *(Separate entry fee required)*

FWKS1 **Workshop: Full Life Cycle with TFS and CSLA .NET** *Rockford Lhotka & Brian Randell*	FWKS2 **Workshop: XAML UX Design** *Billy Hollis*

Lunch

FWKS1 **Workshop: Full Life Cycle with TFS and CSLA .NET** *Rockford Lhotka & Brian Randell*	FWKS2 **Workshop: XAML UX Design** *Billy Hollis*

For the complete session schedule and full session descriptions, please check the Visual Studio Live! Las Vegas web site at ***vslive.com/redmond***
**Speakers and Sessions Subject to Change.*

Test-Driving ASP.NET MVC

Keith Burnell

At the core of the Model-View-Controller (MVC) pattern is the segregation of UI functions into three components. The model represents the data and behavior of your domain. The view manages the display of the model and handles interactions with users. The controller orchestrates the interactions between the view and the model. This separation of inherently difficult-to-test UI logic from the business logic makes applications implemented with the MVC pattern extremely testable. In this article I'll discuss best practices and techniques for enhancing the testability of your ASP.NET MVC applications, including how to structure your solution, architecting your code to handle the injection of dependencies and implementing dependency injection with StructureMap.

This article discusses a prerelease version of ASP.NET MVC 4. All information is subject to change.

This article discusses:

- Structuring your solution for maximum testability
- Introducing dependency injection into your architecture
- Taking dependency injection to the next level with StructureMap

Technologies discussed:

Visual Studio, ASP.NET MVC 4, ASP.NET MVC 3, StructureMap

Code download available at:

code.msdn.microsoft.com/mag201207MVC

Structuring Your Solution for Maximum Testability

What better way to start our discussion than where every developer starts a new project: creating the solution. I'll discuss some best practices for laying out your Visual Studio solution based on my experience developing large enterprise ASP.NET MVC applications using test-driven development (TDD). To start, I suggest using the empty project template when creating an ASP.NET MVC project. The other templates are great for experimenting or creating proofs of concept, but they generally contain a lot of noise that's distracting and unnecessary in a real enterprise application.

Whenever you create any type of complex application, you should use an *n*-tier approach. For ASP.NET MVC application development, I recommend using the approach illustrated in **Figure 1** and **Figure 2**, which contain the following projects:

- The Web project contains all the UI-specific code, including views, view models, scripts, CSS and so forth. This layer has the ability to access only the Controllers, Service, Domain and Shared projects.
- The Controllers project contains the controller classes used by ASP.NET MVC. This layer communicates with the Service, Domain and Shared projects.
- The Service project contains the application's business logic. This layer communicates with the DataAccess, Domain and Shared projects.
- The DataAccess project contains the code used to retrieve and manipulate the data that drives the application. This

layer communicates with the Domain and Shared projects.

- The Domain project contains the domain objects used by the application and is prohibited from communicating with any of the projects.
- The Shared project contains code that needs be available to multiple other layers, such as loggers, constants and other common utility code. It's allowed to communicate only with the Domain project.

How you name your test projects is as important as where you locate them.

I recommend placing your controllers into a separate Visual Studio project. For information on how this is easily accomplished, see the post at bit.ly/K4mF2B. Placing your controllers in a separate project lets you further decouple the logic that resides in the controllers from the UI code. The result is that your Web project contains only code that's truly related to the UI.

Where to Put Your Test Projects Where you locate your test projects and how you name them is important. When you're developing complex, enterprise-level applications, the solutions tend to get pretty large, which can make it difficult to locate a specific class or piece of code in Solution Explorer. Adding multiple test projects to your existing code base only adds to the complexity of navigation in Solution Explorer. I highly recommend physically separating your test projects from your actual application code. I suggest placing all test projects in a Tests folder at the solution level. Locating all your test projects and tests in a single solution folder significantly reduces the noise in your default Solution Explorer view and allows you to easily locate your tests.

Next you'll want to separate the types of tests. More than likely, your solution will contain a variety of test types (unit, integration, performance, UI and so on), and it's important to isolate and group each test type. Not only does this make it easier to locate specific test types, but it also lets you easily run all tests of a specific type. If you're using either of the most popular Visual Studio productivity tool suites, ReSharper (jetbrains.com/ReSharper) or CodeRush (devexpress.com/CodeRush), you get a context menu that allows you to right-click any folder, project or class in Solution Explorer and run all tests contained in the item. To group tests by test type, create a folder for each type of test you plan to write inside the Tests solution folder.

Figure 3 shows an example Tests solution folder containing a number of test type folders.

Naming Your Test Projects How you name your test projects is as important as where you locate them. You want to be able to easily distinguish what part of your application is under test in each test project and what type of tests the project contains. For this, it's a good idea to name your test projects using the following convention: *[Full Name of Project Under Test].Test.[Test Type]*. This allows you to, at a glance, determine exactly what layer of your project is under test and what type of test is being performed. You may be thinking that putting the test projects in type-specific folders and including the test type in the name of the test project is redundant, but remember that solution folders are used only in Solution Explorer and are not included in the namespaces of the projects files. So although the Controllers unit test project is in the Tests\Unit solution folder, the namespace—TestDrivingMVC.Controllers.Test.Unit—doesn't reflect that folder structure. Adding the test type when naming the project is necessary to avoid naming collisions and to determine what type of test you're working with in the editor. **Figure 4** shows Solution Explorer with test projects.

Introducing Dependency Injection to Your Architecture

You can't get very far unit testing an *n*-tier application before you encounter a dependency in your code under test. These dependencies could be other layers of your application, or they could be completely external to your code (such as a database, file system or Web services). When you're writing unit tests, you need to deal with this situation correctly and use test doubles (mocks, fakes or stubs) when you encounter an external dependency. For more information on test doubles, refer to "Exploring the Continuum of Test Doubles" (msdn.microsoft.com/magazine/cc163358) in the September 2007 issue of *MSDN Magazine*. Before you can take advantage of the flexibility that test doubles offer, however, your code has to be architected to handle the injection of dependencies.

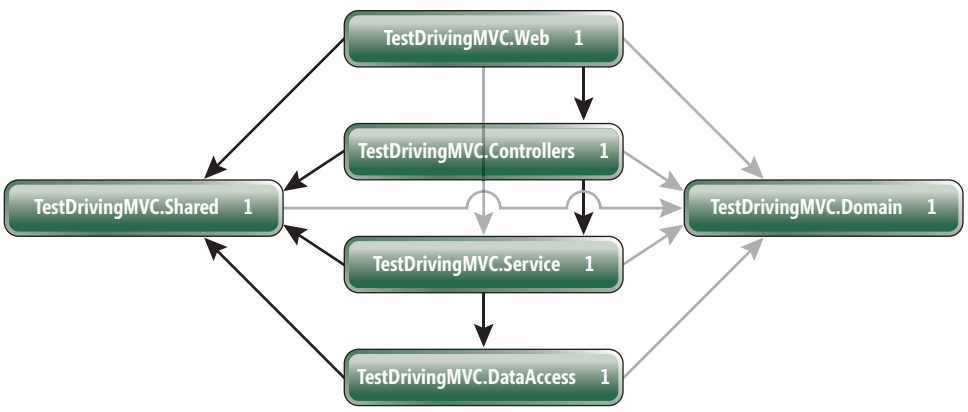


Figure 1 **Interaction Among Layers**

Dependency Injection Dependency injection is the process of injecting the concrete implementations a class requires rather than the class directly instantiating the dependency. The consuming class is not aware of an actual concrete implementation of any of its dependencies, but knows only of the interfaces that back the

dependencies; the concrete implementations are provided either by the consuming class or by a dependency injection framework.

The goal of dependency injection is to create extremely loosely coupled code. The loose coupling lets you easily substitute test double implementations of your dependencies when writing unit tests.

There are three primary ways to accomplish dependency injection:

- Property injection
- Constructor injection
- Using a dependency injection framework/Inversion of Control container (referred to from this point as a DI/IoC framework)

With property injection, you expose public properties on your object to enable its dependencies to be set, as shown in **Figure 5**. This approach is straightforward and requires no tooling.

There are three downsides to this approach. First, it puts the consumer in charge of supplying the dependencies. Next, it requires you to implement guarding code in your objects to ensure the dependencies are set before being used. Finally, as the number of dependencies your object has increases, the amount of code required to instantiate the object increases as well.

Implementing dependency injection using constructor injection involves supplying dependencies to a class via its constructor when the constructor is instantiated, as shown in **Figure 6**. This approach is also straightforward but, unlike property injection, you are assured that the class's dependencies are always set.

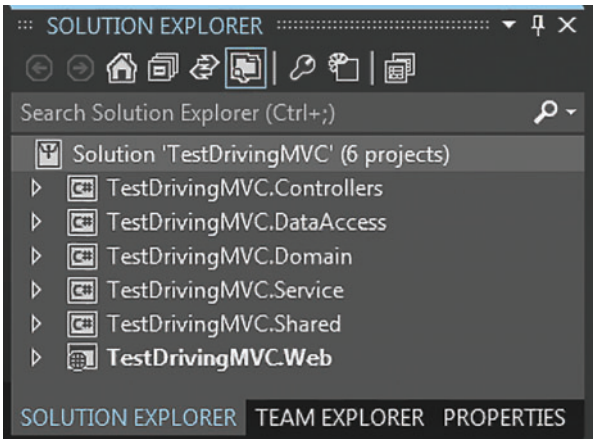


Figure 2 **Example Solution Structure**

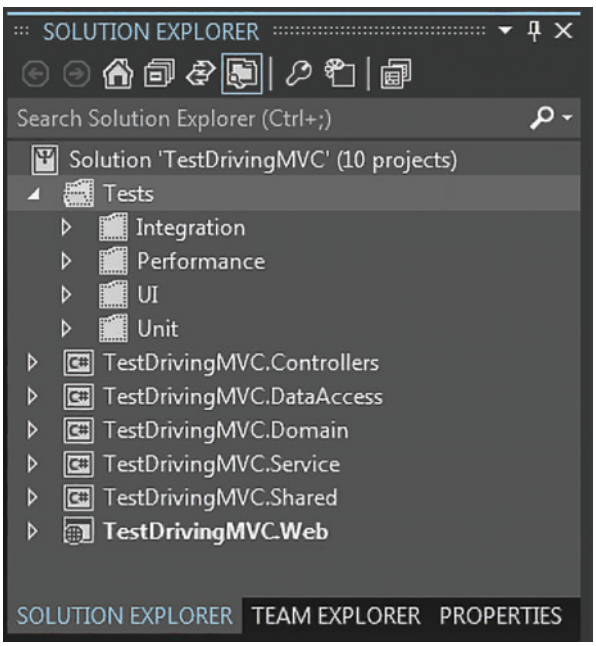


Figure 3 **An Example Tests Solution Folder**

> The goal of dependency injection is to create extremely loosely coupled code.

Unfortunately, this approach still requires the consumer to supply the dependencies. Moreover, it's really suitable only for small applications. Larger applications generally have too many dependencies to supply them via the object's constructor.

The third way to implement dependency injection is to use a DI/IoC framework. A DI/IoC framework completely removes the responsibility of supplying dependencies from the consumer and lets you configure your dependencies at design time and have them resolved at run time. There are many DI/IoC frameworks available for .NET, including Unity (Microsoft's offering), StructureMap, Castle Windsor, Ninject and more. The concept underlying all the different DI/IoC frameworks is the same, and choosing one typically comes down personal preference. To demonstrate DI/IoC frameworks in this article, I'll use StructureMap.

Taking Dependency Injection to the Next Level with StructureMap

StructureMap (structuremap.net) is a widely adopted dependency-injection framework. You can install it via NuGet with either the Package Manager Console (Install-Package StructureMap) or with the NuGet Package Manager GUI (right-click your project's references folder and select Manage NuGet Packages).

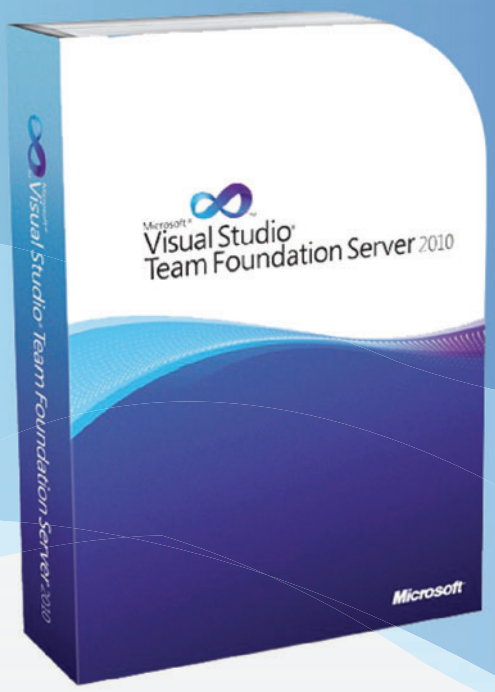
Configuring Dependencies with StructureMap The first step in implementing StructureMap in ASP.NET MVC is to configure your dependencies so StructureMap knows how to resolve them. You do this in the Application_Start method of the Global.asax in either of two ways.

The first way is to manually tell StructureMap that for a specific abstract implementation it should use a specific concrete implementation:

```
ObjectFactory.Initialize(register => {
  register.For<ILoggingService>().Use<LoggingService>();
  register.For<IEmployeeService>().Use<EmployeeService>();
});
```

A drawback of this approach is that you have to manually register each of the dependencies in your application, and with large applications this could become tedious. Moreover, because you register your dependencies in the Application_Start of your ASP.NET MVC site, your Web layer must have direct knowledge of every other layer of your application that has dependencies to wire up.

You can also use the StructureMap auto registration and scanning features to inspect your assemblies and wire up dependencies automatically. With this approach, StructureMap scans your



VISUAL STUDIO TEAM FOUNDATION SERVER HOSTING

30 DAY FREE TRIAL + NO SETUP FEES

WWW.DISCOUNTASP.NET/TFS/MSDN

As a Microsoft Gold Hosting Partner and Microsoft Visual Studio Partner, we offer all the tools development teams require to effectively manage their software development projects without the aggravation or expense of running TFS on their own.

discountASP.net
Team Foundation Server Hosting

Microsoft Partner
Gold Hosting

Microsoft Visual Studio
PARTNER

TFS HOSTING FEATURES

$20/MO PER USER

Monthly Billing

Daily Backups

Unlimited Projects

5gb of Disk Space

Source Control

Work Item Tracking

Team Web Access

Custom Process Template

Secure Access via HTTPS

Visual Studio 2010/2008 Integration

TFS HOSTING ADDONS & MIGRATION SERVICES

- TFS Build Server
- UrbanTurtle Scrum Tools
- VSS to TFS Migration
- TFS 2010 to Hosted TFS Migration

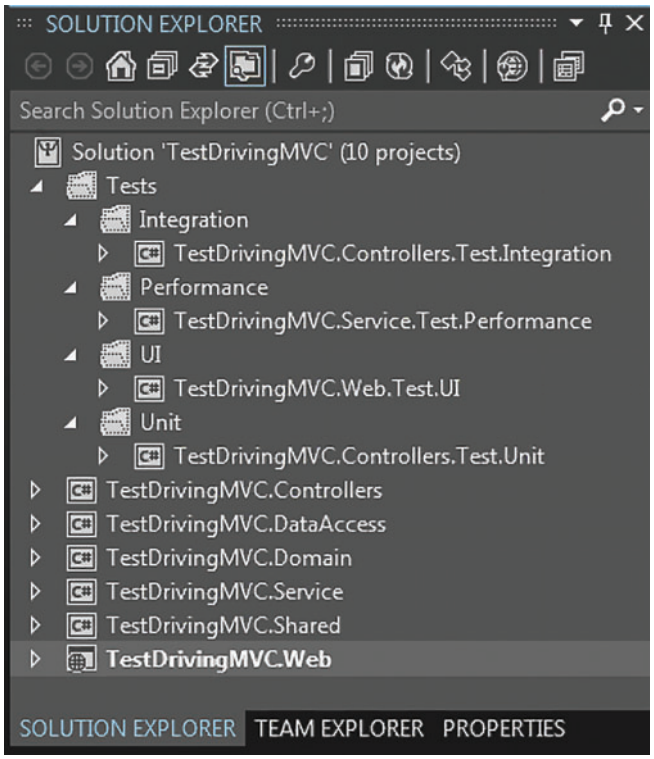


Figure 4 **Test Projects in Solution Explorer**

assemblies, and when it encounters an interface it looks for an associated concrete implementation (based on the notion that an interface named *IFoo* would by convention map to the concrete implementation *Foo*):

```
ObjectFactory.Initialize(registry => registry.Scan(x => {
  x.AssembliesFromApplicationBaseDirectory();
  x.WithDefaultConventions();
}));
```

StructureMap Dependency Resolver Once you have your dependencies configured, you need to be able to access them from your code base. This is accomplished by creating a dependency resolver and locating it in the Shared project (because it will need to be accessed by all layers of the application that have dependencies):

```
public static class Resolver {
  public static T GetConcreteInstanceOf<T>() {
    return ObjectFactory.GetInstance<T>();
  }
}
```

The Resolver class (as I like to call it, because Microsoft introduced a DependencyResolver class with ASP.NET MVC 3, which I'll discuss in a bit) is a simple static class that contains one function. The function accepts a generic parameter T that represents the interface for which you're seeking a concrete implementation, and returns T, which is the actual implementation of the passed-in interface.

Before I jump into how to use the new Resolver class in your code, I want to address why I wrote my own homegrown dependency resolver instead of creating a class that implements the IDependencyResolver interface introduced with ASP.NET MVC 3. The inclusion of the IDependencyResolver functionality is an awesome addition to ASP.NET MVC and a great stride in promoting proper software practices. Unfortunately, it resides in the System.Web.MVC DLL, and I don't want to have references to a Web-technology-specific library in the non-Web layers of my application architecture.

Figure 5 **Property Injection**

```
// Employee Service
public class EmployeeService : IEmployeeService {

  private ILoggingService _loggingService;

  public EmployeeService() {}

  public ILoggingService LoggingService { get; set; }

  public decimal CalculateSalary(long employeeId) {
    EnsureDependenciesSatisfied();
    _loggingService.LogDebug(string.Format(
      "Calculating Salary For Employee: {0}", employeeId));
    decimal output = 0;
    /*
    * Complex logic that needs to be performed
    * in order to determine the employee's salary
    */
    return output;
  }

  private void EnsureDependenciesSatisfied() {
    if (_loggingService == null)
      throw new InvalidOperationException(
        "Logging Service dependency must be satisfied!");
    }
  }
}

// Employee Controller (Consumer of Employee Service)
public class EmployeeController : Controller {

  public ActionResult DisplaySalary(long id) {
    EmployeeService employeeService = new EmployeeService();
    employeeService.LoggingService = new LoggingService();
    decimal salary = employeeService.CalculateSalary(id);
    return View(salary);
  }
}
```

Figure 6 **Constructor Injection**

```
// Employee Service
public class EmployeeService : IEmployeeService {

  private ILoggingService _loggingService;

  public EmployeeService(ILoggingService loggingService) {
    _loggingService = loggingService;
  }

  public decimal CalculateSalary(long employeeId) {
    _loggingService.LogDebug(string.Format(
      "Calculating Salary For Employee: {0}", employeeId));
    decimal output = 0;
    /*
    * Complex logic that needs to be performed
    * in order to determine the employee's salary
    */
    return output;
  }
}

// Consumer of Employee Service
public class EmployeeController : Controller {

  public ActionResult DisplaySalary(long employeeId) {
    EmployeeService employeeService =
      new EmployeeService(new LoggingService());
    decimal salary = employeeService.CalculateSalary(employeeId);
    return View(salary);
  }
}
```

Raising the Bar...

And Pie, and Spline and Scatter... on Mobile Business Intelligence

Compatible with
Microsoft® Visual Studio® 11 Beta

Visualize your BI in 50+ ways, and on just as many devices. From candlesticks to polar and radial charts, nobody offers the breadth and depth of dynamic, high fidelity, totally mobile charting solutions that you get from Infragistics' NetAdvantage. **Try a free, fully supported trial of NetAdvantage for .NET today!**

www.infragistics.com/NET

INFRAGISTICS™
DESIGN / DEVELOP / EXPERIENCE

Infragistics Sales US 800 231 8588 • Europe +44 (0) 800 298 9055 • India +91 80 4151 8042 • APAC (+61) 3 9982 4545

HTML 5

Figure 7 **Resolving Dependencies in Code**

```
public class EmployeeService : IEmployeeService {

  private ILoggingService _loggingService;

  public EmployeeService() {
    _loggingService = Resolver.GetConcreteInstanceOf<ILoggingService>();
  }

  public decimal CalculateSalary(long employeeId) {
    _loggingService.LogDebug(string.Format(
      "Calculating Salary For Employee: {0}", employeeId));
    decimal output = 0;
    /*
    * Complex logic that needs to be performed
    * in order to determine the employee's salary
    */
    return output;
  }
}
```

Resolving Dependencies in Code Now that all the hard work is done, resolving dependencies in your code is simple. All you need to do is call the static GetConcreteInstanceOf function of the Resolver class and pass it the interface for which you're seeking a concrete implementation, as shown in **Figure 7**.

Taking Advantage of StructureMap to Inject Test Doubles in Unit Tests Now that the code is architected so you can inject dependencies without any intervention from the consumer, let's get back to the original task of correctly dealing with dependencies in unit tests. Here's the scenario:

- The task is to write logic using TDD that generates the salary value to return from the CalculateSalary method of the EmployeeService. (You'll find the EmployeeService and CalculateSalary functions in **Figure 7**.)
- There's a requirement that all calls to the CalculateSalary function must be logged.
- The interface for the logging service is defined, but the implementation is not complete. Calling the logging service currently throws an exception.
- The task needs to be completed before work on the logging service is scheduled to start.

It's more than likely you've encountered this type of scenario before. Now, however, you have the proper architecture in place to be able to sever the tie to the dependency by putting a test double in its place. I like to create my test doubles in a project that can be shared among all of my test projects. As you can see in **Figure 8**, I've created a Shared project in my Tests solution folder.

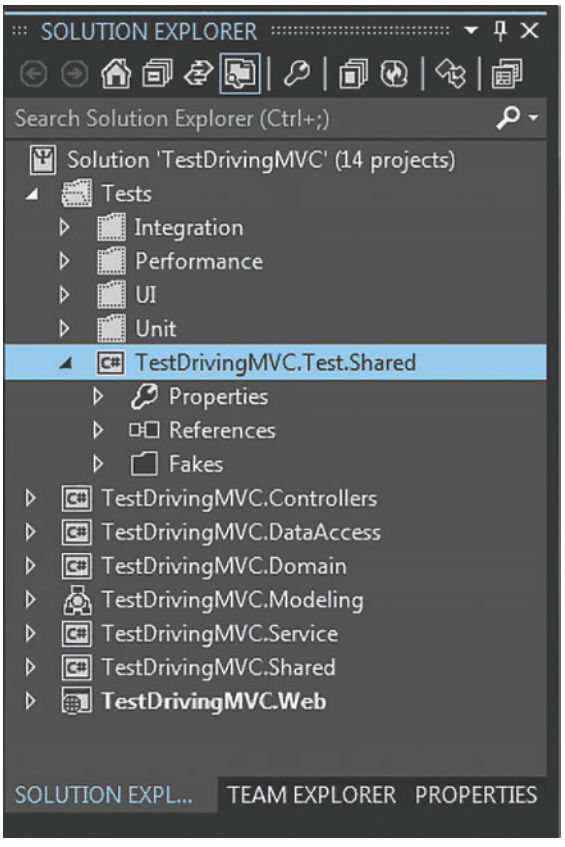


Figure 8 **Project for Shared Test Code and Fakes**

Inside the project I added a Fakes folder because, to complete my testing, I need a *fake* implementation of the ILoggingService.

Creating a fake for the logging service is easy. First, I create a class inside my Fakes folder named LoggingServiceFake. LoggingServiceFake needs to satisfy the contract that the EmployeeService is expecting, which means it needs to implement ILoggingService and its methods. By definition, a fake is a stand-in that contains just enough code to satisfy the interface. Typically, this means it has empty implementations of void methods and the function implementations contain a return statement that returns a hardcoded value, like so:

> By definition, a fake is a stand-in that contains just enough code to satisfy the interface.

```
public class LoggingServiceFake : ILoggingService {

  public void LogError(string message, Exception ex) {}

  public void LogDebug(string message) {}

  public bool IsOnline() {
    return true;
  }
}
```

Now that the fake is implemented, I can write my test. To start out, I'll create a test class in the TestDrivingMVC.Service.Test.Unit unit test project and, per the naming conventions discussed earlier, I'll name it EmployeeServiceTest, as shown in **Figure 9**.

For the most part, the test class code is straightforward. The line you want to pay particularly close attention to is:

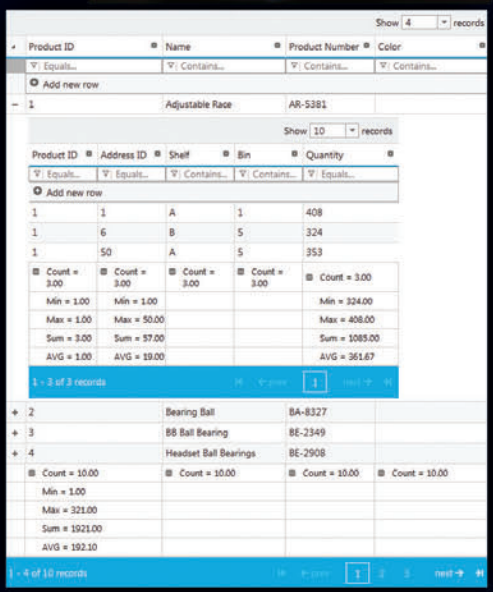
```
ObjectFactory.Initialize(x =>
  x.For<ILoggingService>().Use(
  _loggingService));
```

This is the code that instructs StructureMap to use the LoggingServiceFake when the Resolver class we created earlier attempts to resolve ILoggingService. I put this code in a method marked with TestInitialize, which tells the unit-testing framework to execute this method prior to running every test in the test class.

Using the power of DI/IoC and the StructureMap tool, I'm able to completely sever the tie to the Logging Service. Doing this enables me to complete my coding and unit testing without being affected by the state of the logging service, and to code true unit tests that don't rely on any dependencies.

MetroTactual

[me-troh tak-choo-uhl]



Compatible with
Microsoft® Visual Studio® 11 Beta

noun, adjective

1. Modern, clean, sleek, stylish, touch-friendly design and UX
2. Feng Shui for your apps
3. Available in NetAdvantage 12.1 toolsets

See also: NetAdvantage for .NET

Try your free, fully supported trial today.
www.infragistics.com/NET

INFRAGISTICS™
DESIGN / DEVELOP / EXPERIENCE

Infragistics Sales US 800 231 8588 • Europe +44 (0) 800 298 9055 • India +91 80 4151 8042 • APAC (+61) 3 9982 4545

HTML 5

Figure 9 **The EmployeeServiceTest Test Class**

```
[TestClass]
public class EmployeeServiceTest {

  private ILoggingService _loggingServiceFake;
  private IEmployeeService _employeeService;

  [TestInitialize]
  public void TestSetup() {
    _loggingServiceFake = new LoggingServiceFake();
    ObjectFactory.Initialize(x => x.For<ILoggingService>().Use(_
      loggingServiceFake));
    _employeeService = new EmployeeService();
  }

  [TestMethod]
  public void CalculateSalary_ShouldReturn_Decimal() {
    // Arrange
    long employeeId = 12345;
    // Act
    var result = _employeeService.CalculateSalary(employeeId);
    // Assert
    result.ShouldBeType<decimal>();
  }
}
```

Using StructureMap as the Default Controller Factory ASP.NET MVC provides an extensibility point that lets you add a custom implementation of how controllers are instantiated in your application. By creating a class that inherits from DefaultControllerFactory (see **Figure 10**), you can control how controllers are created.

In the new controller factory, I have a public StructureMap Container property that gets set based on the StructureMap ObjectFactory (which is configured in **Figure 10** in the Global.asax).

Next, I have an override of the GetControllerInstance method that does some type checking and then uses the StructureMap container to resolve the current controller based on the supplied controller type parameter. Because I used the StructureMap auto registration and scanning features when configuring StructureMap initially, there's nothing else I have to do.

Figure 10 **Custom Controller Factory**

```
public class ControllerFactory : DefaultControllerFactory {

  private const string ControllerNotFound = "The controller for path
'{0}' could not be found or it does not implement IController.";
  private const string NotAController = "Type requested is not a
controller: {0}";
  private const string UnableToResolveController = "Unable to resolve
controller: {0}";

  public ControllerFactory() {
    Container = ObjectFactory.Container;
  }

  public IContainer Container { get; set; }

  protected override IController GetControllerInstance(
    RequestContext context, Type controllerType) {
    IController controller;
    if (controllerType == null)
      throw new HttpException(404, String.Format(ControllerNotFound,
      context.HttpContext.Request.Path));

    if (!typeof (IController).IsAssignableFrom(controllerType))
      throw new ArgumentException(string.Format(NotAController,
      controllerType.Name), "controllerType");

    try {
      controller = Container.GetInstance(controllerType) as IController;
    }
    catch (Exception ex) {
      throw new InvalidOperationException(
      String.Format(UnableToResolveController, controllerType.Name), ex);
    }
    return controller;
  }
}
```

Figure 11 **Resolving the Controller**

```
public class HomeController : Controller {

  private readonly IEmployeeService _employeeService;

  public HomeController(IEmployeeService employeeService) {
    _employeeService = employeeService;
  }

  public ActionResult Index() {
    return View();
  }

  public ActionResult DisplaySalary(long id) {
    decimal salary = _employeeService.CalculateSalary(id);
    return View(salary);
  }
}
```

The benefit of creating a custom controller factory is that you're no longer limited to parameterless constructors on your controllers. At this point you might be asking yourself, "How would I go about supplying parameters to a controller's constructor?" Thanks to the extensibility of the DefaultControllerFactory and StructureMap, you don't have to. When you declare a parameterized constructor for your controllers, the dependencies are automatically resolved when the controller is resolved in the new controller factory.

The benefit of creating a custom controller factory is that you're no longer limited to parameterless constructors on your controllers.

As you can see in **Figure 11**, I've added an IEmployeeService parameter to the HomeController's constructor. When the controller is resolved in the new controller factory, any parameters required by the controller's constructor are automatically resolved. This means you don't need to add the code to resolve the controller's dependencies manually—but you can still use fakes, as discussed earlier.

By using these practices and techniques in your ASP.NET MVC applications, you'll position yourself for an easier and cleaner TDD process. ■

KEITH BURNELL *is a senior software engineer with Skyline Technologies. He's been developing software for more than 10 years, specializing in large-scale ASP.NET and ASP.NET MVC Web site development. Burnell is an active member of the developer community and can be found on his blog (dotnetdevdude.com) and on Twitter at twitter.com/keburnell.*

THANKS to the following technical experts for reviewing this article: *John Ptacek and Clark Sell*

LIVE! 360

IT EVENTS WITH PERSPECTIVE

December 10-14, 2012 | Royal Pacific Resort, Orlando, FL

CODE IN THE SUNSHINE!

Visual Studio Live! is back in Orlando, bringing attendees everything they love about this event: hard-hitting, practical .NET Developer training from the industry's best speakers and Microsoft insiders. But this year, there's an extra bonus – THREE extra co-located events that you can attend for free: SharePoint Live!, SQL Server Live! and Cloud & Virtualization Live! **live360events.com**

Who Should Attend?

- Developers
- IT Pros
- Database Administrators
- System Architects
- Integrators
- Engineers
- Systems Administrators

SAVE UP TO $400 –
Register Before October 10th

Use Promo Code DEVJULY

WHAT IS LIVE! 360?

LIVE! 360 is a new event brought to you by the publishers of *Visual Studio* Magazine, *MSDN Magazine*, *Redmond* Magazine, *Redmond Channel Partner* Magazine, and *Virtualization Review* Magazine; and the producers of Visual Studio Live! and TechMentor conferences. This event will bring together Visual Studio Live!, an established and respected independent software conference, with three new industry events: Cloud & Virtualization Live!, SharePoint Live!, and SQL Server Live!.

Live360events.com

Scan the QR code or visit live360events.com for more information.

LIVE! 360

IT EVENTS WITH PERSPECTIVE

Buy 1 Event, Get 3 Free!

Customize an agenda to suit YOUR needs – attend just one Live! event or all four for the same low price!

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

Code in the Sunshine!
Developers, software architects, programmers and designers will receive hard-hitting and practical .NET Developer training from industry experts and Microsoft insiders. **vslive.com/orlando**

Cloud & Virtualization LIVE!

THE FUTURE OF COMPUTING

Cloud and Virtualization for the Real World
The event for IT Professionals, Systems Administrators, Developers and Consultants to develop their skill sets in evaluating, deploying and optimizing cloud and virtual-based environments.**virtlive360.com**

SQL Server LIVE!

TRAINING FOR DBAs AND IT PROS

Bringing SQL Server to Your World
IT Professionals, DBAs and Developers – across a breadth of experience and organizations – come together for comprehensive education and knowledge share for SQL Server database management, performance tuning and troubleshooting. **sqllive360.com**

SharePoint LIVE!

TRAINING FOR COLLABORATION

Build. Develop. Implement. Manage.
Leading-edge knowledge and training for SharePoint administrators, developers, and planners who must customize, deploy and maintain SharePoint Server and SharePoint Foundation to maximize the business value. **splive360.com**

PRODUCED BY

1105 MEDIA INC

Writing a Compass Application for Windows Phone

Donn Morse

As a writer responsible for the sensor platform documentation for Windows 8, I want to see as many developers as possible adopt our new platform. And, because Metro-style apps can be written using XAML and C#, the Windows Phone developer is an ideal candidate for this migration. Windows Phone developers already have experience with XAML, and a number also have experience with sensors (because the accelerometer, compass, gyro and GPS sensors are exposed in the most recent Windows Phone release).

To help me better understand Windows Phone developers and their development platform, I decided to write a simple compass app last fall. Once I wrote it, I submitted a free version to the Windows Phone Marketplace using the App Hub. Since acceptance, the app has been downloaded by Windows Phone users from as far away as Switzerland and Malaysia.

This article covers the development of the application.

This article discusses:

- How the compass app works
- Designing the compass app UI
- The code behind the primary screen of the compass app

Technologies discussed:

Windows Phone

Code download available at:

code.msdn.microsoft.com/mag201207Compass

The Compass Application

The compass application uses the compass, or magnetometer, built in to the Windows Phone device. This app provides a heading with respect to true north as well as a reciprocal heading that can be useful when navigating in a boat or orienteering in a remote area with a map. In addition, the app allows the user to toggle from a numeric heading (for example, "090" degrees) to an alpha heading (for example, "E" for east). The app also allows the user to lock the current heading. This is useful when users need the pointer to remain stationary so they can take a bearing on a specific landmark or reference point on a map or chart.

Figure 1 shows the application running on a Samsung Focus. The image on the left displays a numeric heading and the image on the right displays an alpha heading.

Designing the UI

As a developer accustomed to writing apps for the PC, I initially felt limited by the reduced screen real estate on the phone. However, this wasn't a showstopper. I just needed to give a little more thought—and careful consideration—to the features in my app with respect to the new screen dimensions. My compass app has two screens: a calibration screen and the main navigation screen.

The Calibration Screen The compass, or magnetometer, installed on a Windows Phone device requires calibration after the device is powered up. In addition, these sensors might require periodic recalibration. To help you programmatically detect when calibration is necessary, the platform supports a HeadingAccuracy property

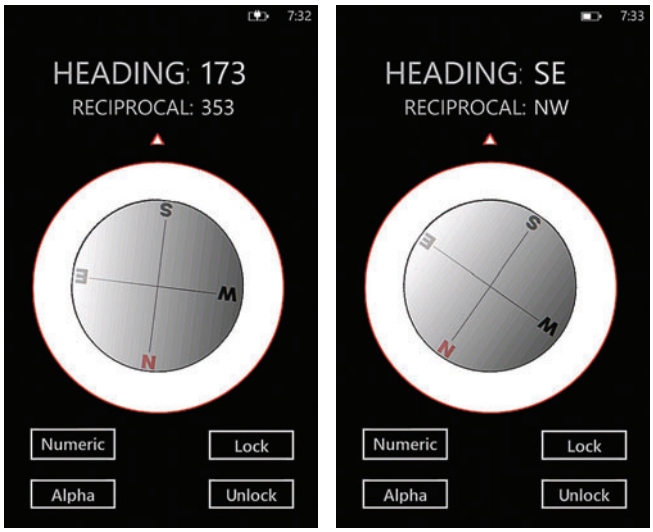


Figure 1 **The Running App, Showing a Numeric Heading (Left) and an Alpha Heading (Right)**

that you can use to detect the current calibration. In addition, the platform supports a Calibrate event that's fired if the compass requires calibration.

My app handles the Calibrate event, which, in turn, displays a calibration screen (calibrationStackPanel) that prompts the user to manually calibrate the device by sweeping the phone in a figure 8 motion. As the user sweeps the phone, the current accuracy is displayed in the CalibrationTextBlock with a red font until the desired accuracy is achieved, as shown in **Figure 2**. Once the returned accuracy is less than or equal to 10°, the numeric values are cleared and a green "Complete!" appears.

The corresponding code that supports calibration is found in the module MainPage.xaml.cs within the compass_CurrentValueChanged event handler, as shown in **Figure 3**.

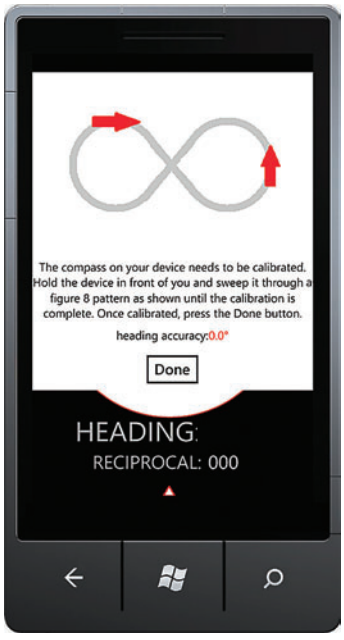


Figure 2 **The Calibration Screen**

Once the desired accuracy is achieved, the user is prompted to press the Done button, which hides the calibration screen and displays the app's primary screen.

The Primary Screen This screen displays a numeric or alpha heading and reciprocal value. In addition, it renders the face of a compass that's oriented with respect to true north. Finally, the primary screen displays the four controls (or buttons) that let the user alter the output as well as lock the heading value and compass face.

Figure 4 shows my application's primary screen, MainPage.xaml, as it appears in Visual Studio.

Most of the UI elements in the primary screen are simple TextBlock and Button controls. The text blocks identify the heading and its reciprocal value. The buttons enable the user to control the output. However, the compass face is a little more involved.

The Compass Face The compass face consists of three components: a background image, a foreground image, and a larger, bordering ellipse in which the foreground and background images rotate. The background image contains the letters corresponding to the four points on a compass, a horizontal line and a vertical line. The foreground image creates the smoked-glass effect.

The Background Image In the XAML, the background image is named CompassFace (this variable name is referenced later in the code that rotates the compass face):

```
<Image Height="263" HorizontalAlignment="Left" Margin="91,266,0,0"
  Name="CompassFace" VerticalAlignment="Top" Width="263"
  Source="/Compass71;component/compass.png" Stretch="None" />
```

The Foreground Image The foreground of the compass face, EllipseGlass, is defined in the XAML itself. The smoked-glass effect is created using a linear-gradient brush. I created this ellipse using Microsoft Expression Blend 4. The Expression Blend tool is compatible with Visual Studio and lets you load your application's XAML and enhance the UI by customizing the graphics. **Figure 5** shows the Expression Blend editor as it appeared when creating the shaded ellipse.

> Most of the UI elements in the primary screen are simple TextBlock and Button controls.

After I finished editing the ellipse in Expression Blend, the XAML in my Visual Studio project was updated with the following XML:

```
<Ellipse Height="263"  Width="263" x:Name="EllipseGlass"
  Margin="91,266,102,239" Stroke="Black" StrokeThickness="1">
  <Ellipse.Fill>
    <LinearGradientBrush EndPoint="1,0.5" StartPoint="0,0.5">
    <GradientStop Color="#A5000000" Offset="0" />
    <GradientStop Color="#BFFFFFFF" Offset="1" />
    </LinearGradientBrush>
  </Ellipse.Fill>
```

Note that the dimensions of EllipseGlass (263 x 263 pixels) are an exact match of the dimensions of compass.png. Also note that the object name, EllipseGlass, is referenced later in the code that performs the rotation of the compass face.

The Compass Face Border The compass face rotates within a larger, white ellipse with a red border. This ellipse is defined in the XAML and named EllipseBorder:

```
<Ellipse Height="385" HorizontalAlignment="Left" Margin="31,0,0,176"
  Name="EllipseBorder" Stroke="#FFF80D0D" StrokeThickness="2"
  VerticalAlignment="Bottom" Width="385" Fill="White" />
```

The Code Behind the UI

The code resides in the file MainPage.xaml.cs in the accompanying code download, and it accesses the namespaces required by the application, initializes the sensor, sets a reporting interval and handles the various application features: calibration,

Figure 3 **Calibrating the Compass**

```
...
else
{
 if (HeadingAccuracy <= 10)
 {
  CalibrationTextBlock.Foreground = new SolidColorBrush(Colors.Green);
  CalibrationTextBlock.Text = "Complete!";
 }
 else
 {
  CalibrationTextBlock.Foreground = new SolidColorBrush(Colors.Red);
  CalibrationTextBlock.Text = HeadingAccuracy.ToString("0.0");
 }
}
```

rotating the compass face, toggling between numeric and alpha output and so on.

Accessing the Compass in Your Code The first step in writing a compass application (or any application that accesses one of the phone sensors) is to obtain access to the sensor objects exposed by the namespace Microsoft.Devices.Sensors. This is accomplished with the following using directive in MainPage.xaml.cs:

```
using Microsoft.Devices.Sensors;
```

Once this using directive appears in the file, I can create a compass variable that gives me programmatic access to the actual device in the phone:

```
namespace Compass71
{
  public partial class MainPage : PhoneApplicationPage
  {
    Compass compass = new Compass();
```

I'll use this variable to start the compass, stop it, retrieve the current heading accuracy, set the report interval and so on.

Starting the Compass and Setting the Report Frequency Once I create the compass variable, I can begin invoking methods and setting properties on the object. The first method I invoked is the Start method, which allows me to begin receiving data from the sensor. After I start the compass, I set the report interval—the time between sensor updates—to 400 ms (note that the TimeBetweenUpdates property requires a multiple of 20 ms):

```
compass.TimeBetweenUpdates = TimeSpan.
FromMilliseconds(400);  // Must be multiple of 20
compass.Start();
```

The 400 ms value was chosen by trial and error. The default report interval is extremely short. If you attempt to run the application at this default value, the compass face is rotated so frequently that it appears to be unstable.

Establishing the Compass Event Handlers The compass app supports two event handlers: one that displays the calibration page (calibrationStackPanel), and another that renders the current heading and rotates the compass face.

Defining and Establishing the Calibration Event Handler The calibration event handler contains relatively few lines of code. This code, shown in **Figure 6**, accomplishes two primary tasks: First, it displays the calibration screen that's defined in the MainPage.xaml file; second, it sets a Boolean variable *calibrating* to true.

Figure 4 **The App's Primary Screen in Visual Studio**

Because this event handler is called from a background thread, it doesn't have direct access to the UI thread. So, to display the calibration screen, I need to call the BeginInvoke method on the Dispatcher object.

The Boolean variable *calibrating* is examined within the code for the value-changed event handler (compass_CurrentValueChanged). When this variable is true, I ignore the compass and update the calibration screen with the most recent calibration data. When the variable is false, I update the compass readings and perform rotations of the compass face.

> The first step in writing a compass application (or any application that accesses one of the phone sensors) is to obtain access to the sensor objects exposed by the namespace Microsoft.Devices.Sensors.

This event handler is established in the MainPage constructor with the following line of code:

```
compass.Calibrate += new EventHandler<CalibrationEventArgs>(compass_Calibrate);
```

Defining and Establishing the Value-Changed Handler The value-changed event handler (compass_CurrentValueChanged) is invoked each time a new reading arrives from the compass. And, depending on the state of the *calibrating* variable, it either updates the calibration screen or it updates the primary screen.

When it's updating the primary screen, the event handler performs the following tasks:

- It computes the true and reciprocal headings with respect to true north.
- It rotates the compass face.
- It renders the current and reciprocal headings.

Computing the Headings The following code demonstrates how the event handler retrieves the heading with respect to true north using the TrueHeading property on the SensorReading object:

```
TrueHeading = e.SensorReading.TrueHeading;
if ((180 <= TrueHeading) && (TrueHeading <= 360))
  ReciprocalHeading = TrueHeading - 180;
Else
  ReciprocalHeading = TrueHeading + 180;
```

Figure 7 demonstrates how the event handler updates the current and reciprocal headings.

Rotating the Compass Face The following code snippet demonstrates how the app rotates the two ellipses that constitute the background and foreground of the compass face:

```
CompassFace.RenderTransformOrigin = new Point(0.5, 0.5);
EllipseGlass.RenderTransformOrigin = new Point(0.5, 0.5);


transform.Angle = 360 - TrueHeading;
CompassFace.RenderTransform = transform;
EllipseGlass.RenderTransform = transform;
```

The variable CompassFace corresponds to the background image that contains the four points of the compass (N, E, W and S) and the horizontal and vertical line. The variable EllipseGlass corresponds to the smoked-glass layer.

The compass app supports two event handlers.

Before I can apply a rotation transform, I need to ensure that the transform is centered on the two objects that I'll rotate. This is done by invoking the RenderTransformOrigin method on each object and supplying the coordinates (0.5, 0.5). (For more information about this method and its use, refer to the MSDN Library page, "UIElement.RenderTransformOrigin Property," at bit.ly/KIn8Zh.)

Once I've centered the transform, I can compute the angle and perform the rotation. I compute the angle by subtracting the current heading from 360. (This is the heading I just received in the event handler.) I apply this new angle with the RenderTransform property.

Locking and Unlocking the Compass The locking and unlocking feature was intended for someone outdoors who's using the app to navigate (whether in a boat or hiking on a trail with map in hand). This feature is simple; it invokes the Stop method on the compass to lock the heading and then it invokes the Start method to resume heading retrieval.

The Stop method is invoked when the user presses LockButton:

```
private void LockButton_Click(object sender, RoutedEventArgs e)
{
  try
  {
    compass.Stop();
  }
  catch (Exception ex)
  {
    MessageBox.Show(ex.Message.ToString(), "Error!", MessageBoxButton.OK);
  }
}
```

The Start method is invoked when the user presses UnlockButton:

```
private void UnlockButton_Click(object sender, RoutedEventArgs e)
{
  try
  {
    compass.Start();
    compass.TimeBetweenUpdates =
      TimeSpan.FromMilliseconds(400);  // Must be multiple of 20
  }
  catch (Exception ex)
  {
    MessageBox.Show(ex.Message.ToString(), "Error!", MessageBoxButton.OK);
  }
}
```

Note that in addition to restarting the compass, I reset the report interval to 400 ms to ensure consistent behavior.

Toggling Between Numeric and Alpha Headings The code that toggles between alpha and numeric headings is controlled by a single Boolean variable named Alphabetic that's set when a user presses AlphaButton or NumericButton. When the user presses AlphaButton, this variable is set to True; when the user presses NumericButton, it's set to False.

Following is the code for the AlphaButton click event:

```
private void AlphaButton_Click(
  object sender, RoutedEventArgs e)
{
  try
  {
    Alphabetic = true;
  }
  catch (Exception ex)
  {
    MessageBox.Show(
      ex.Message.ToString(), "Error!",
      MessageBoxButton.OK);
  }
}
```

The code within the compass_CurrentValueChanged event handler examines Alphabetic to determine whether it should render the numeric or alpha headings.

Supporting the Light and Dark Visibility Themes After I created the app and submitted it to the App

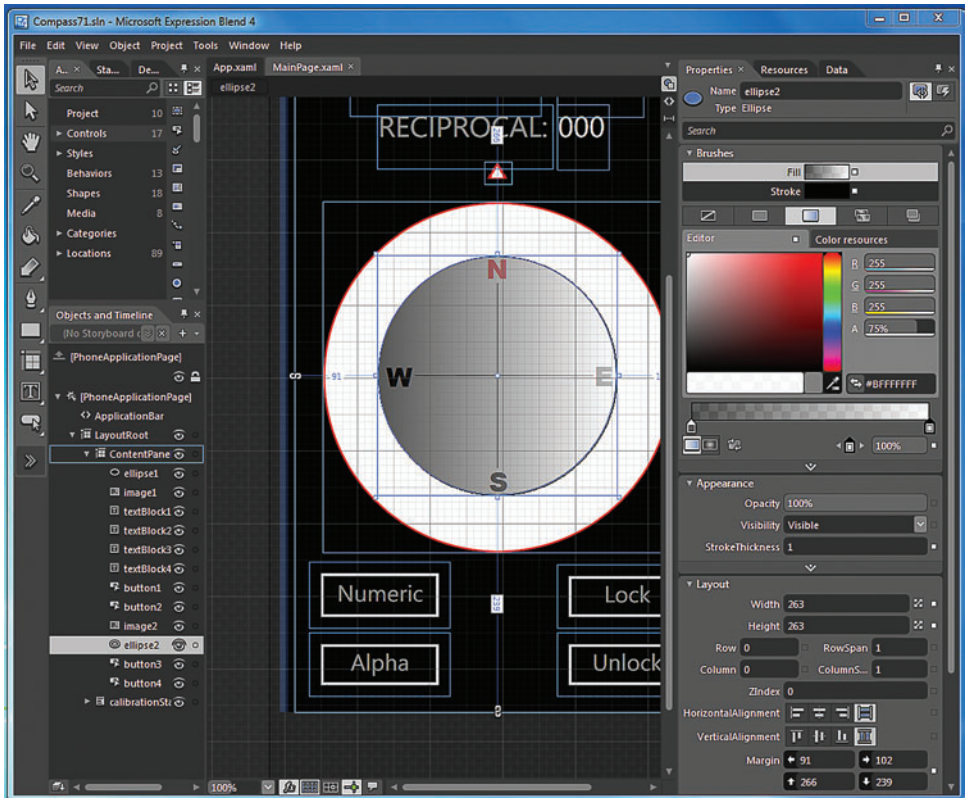


Figure 5 **Creating a Shaded Ellipse in Expression Blend**

Hub for certification, I was surprised to receive notification that it had failed because certain UI elements disappeared when the light visibility theme was tested. (I'd been running exclusively with the dark theme and had failed to test the light theme.)

To resolve this issue, I added code to the MainPage constructor, which retrieves the current theme and then sets the foreground color of the UI elements (text blocks and buttons) to work with the given theme. If the light theme is set, the foreground colors of the elements are set to black and red. If the dark theme is set, the foreground colors of the elements are set to dark and light gray. **Figure 8** shows this code.

Figure 6 **The Calibration Event Handler**

```
void compass_Calibrate(object sender, CalibrationEventArgs e)
{
  try
  {
    Dispatcher.BeginInvoke(() => { calibrationStackPanel.Visibility =
      Visibility.Visible;
    });
    calibrating = true;
  }
  catch (Exception ex)
  {
    MessageBox.Show(ex.Message.ToString(), "Error!", MessageBoxButton.OK);
  }
}
```

Figure 7 **Updating Current and Reciprocal Headings**

```
if (!Alphabetic) // Render numeric heading
{
  HeadingTextBlock.Text = TrueHeading.ToString();
  RecipTextBlock.Text = ReciprocalHeading.ToString();
}
else // Render alpha heading
{
  if (((337 <= TrueHeading) && (TrueHeading < 360)) ||
    ((0 <= TrueHeading) && (TrueHeading < 22)))
  {
    HeadingTextBlock.Text = "N";
    RecipTextBlock.Text = "S";
  }
  else if ((22 <= TrueHeading) && (TrueHeading < 67))
  {
    HeadingTextBlock.Text = "NE";
    RecipTextBlock.Text = "SW";
  }
  else if ((67 <= TrueHeading) && (TrueHeading < 112))
  {
    HeadingTextBlock.Text = "E";
    RecipTextBlock.Text = "W";
  }
  else if ((112 <= TrueHeading) && (TrueHeading < 152))
  {
    HeadingTextBlock.Text = "SE";
    RecipTextBlock.Text = "NW";
  }
  else if ((152 <= TrueHeading) && (TrueHeading < 202))
  {
    HeadingTextBlock.Text = "S";
    RecipTextBlock.Text = "N";
  }
  else if ((202 <= TrueHeading) && (TrueHeading < 247))
  {
    HeadingTextBlock.Text = "SW";
    RecipTextBlock.Text = "NE";
  }
  else if ((247 <= TrueHeading) && (TrueHeading < 292))
  {
    HeadingTextBlock.Text = "W";
    RecipTextBlock.Text = "E";
  }
  else if ((292 <= TrueHeading) && (TrueHeading < 337))
  {
    HeadingTextBlock.Text = "NW";
    RecipTextBlock.Text = "SE";
  }
}
```

Figure 8 **Coordinating Themes and Colors**

```
Visibility isLight = (Visibility)Resources["PhoneLightThemeVisibility"];
// For light theme
if (isLight == System.Windows.Visibility.Visible) // Light theme enabled
{

  // Constructor technique
  SolidColorBrush scb = new SolidColorBrush(Colors.Black);
  SolidColorBrush scb2 = new SolidColorBrush(Colors.Red);

  RecipLabelTextBlock.Foreground = scb;
  HeadingLabelTextBlock.Foreground = scb;

  RecipTextBlock.Foreground = scb2;
  HeadingTextBlock.Foreground = scb2;

  LockButton.Foreground = scb;
  UnlockButton.Foreground = scb;
  AlphaButton.Foreground = scb;
  NumericButton.Foreground = scb;
}
else // Dark color scheme is selected—set text colors accordingly
{
  // Constructor technique
  SolidColorBrush scb = new SolidColorBrush(Colors.DarkGray);
  SolidColorBrush scb2 = new SolidColorBrush(Colors.LightGray);

  RecipLabelTextBlock.Foreground = scb;
  HeadingLabelTextBlock.Foreground = scb;

  RecipTextBlock.Foreground = scb2;
  HeadingTextBlock.Foreground = scb2;

  LockButton.Foreground = scb;
  UnlockButton.Foreground = scb;
  AlphaButton.Foreground = scb;
  NumericButton.Foreground = scb;
}
```

Fun and Valuable

The creation of this app was a lot of fun, and it was valuable as well. Having worked with a sensor in the Windows Phone platform, I now have a clearer understanding of the differences between this platform and the sensor support in Windows 8. But what struck me the most were the similarities. My hunch is that if you're a Windows Phone developer who has spent any time with the sensor namespace, you're going to find the migration to Windows 8 exceptionally straightforward. And, in Windows 8, you'll find additional sensors such as the inclinometer, orientation sensor and simple orientation sensor. (The orientation sensor is a fusion of multiple sensors that returns a Quaternion, or rotation matrix, which you can use to control complex games. The simple orientation sensor lets you detect whether your device is in portrait or landscape mode, as well as faceup or facedown.)

The development opportunities afforded by all these sensors are exciting, and I look forward to seeing the imaginative ways our creative developer community can put them to use. ■

Donn Morse *is a senior programming writer on the Windows team at Microsoft, who has focused for the past several years on the sensor platform, from the application side to the driver. He's passionate about, and fascinated by, sensors and their use.*

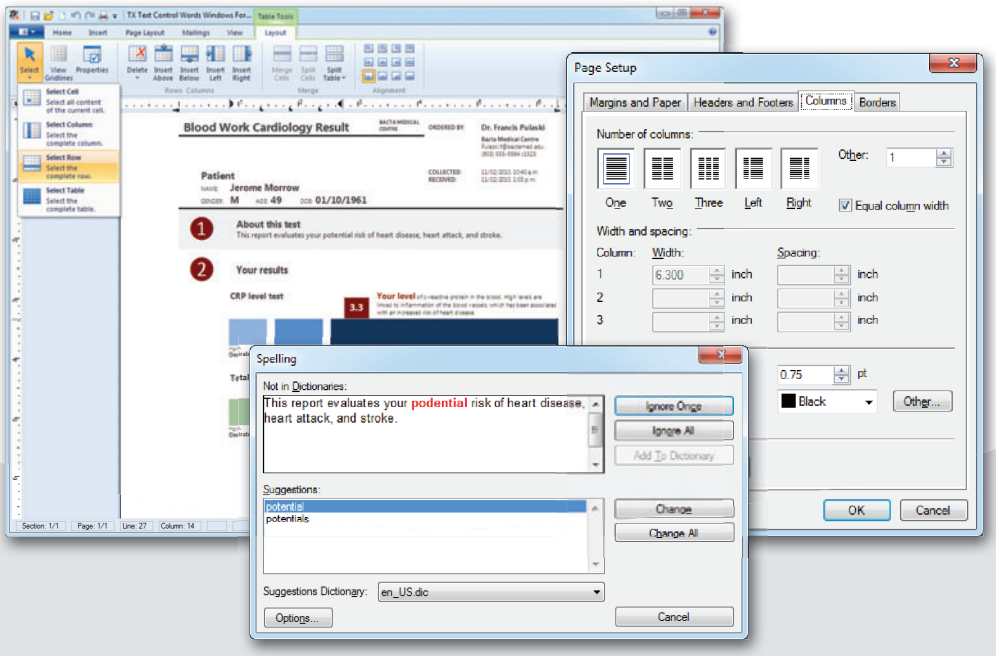
Thanks to the following technical expert for reviewing this article: *Jason Scott*

WINDOWS FORMS | WPF | ASP.NET | ACTIVEX

WORD PROCESSING COMPONENTS

Ⓘ Rich Text Editing

Integrate professional rich text editing into your .NET based applications.



Rich Text Editing

Add feature-complete word processing capabilities to your Microsoft .NET applications.

Free License

Download our 100% free Express version.

Spell Checking

Add the fastest spell checking engine to your Windows Forms, WPF or ASP.NET applications.

Reporting Redefined

Build MS Word compatible mail merge applications with Master-Detail views.

TX TEXT CONTROL® word processing components

Microsoft® Visual Studio® PARTNER

Word Processing Components
for Windows Forms, WPF & ASP.NET

WWW.TEXTCONTROL.COM

US +1 877-462-4772 (toll-free)
EU +49 421-4270671-0

Hadoop on Windows Azure

Lynn Langit

There's been a lot of buzz about Hadoop lately, and interest in using it to process extremely large data sets seems to grow day by day. With that in mind, I'm going to show you how to set up a Hadoop cluster on Windows Azure. This article assumes basic familiarity with Hadoop technologies. If you're new to Hadoop, see "What Is Hadoop?" As of this writing, Hadoop on Windows Azure is in private beta. To get an invitation, visit hadooponazure.com. The beta is compatible with Apache Hadoop (snapshot 0.20.203+).

Setting Up Your Cluster

Once you're invited to participate in the beta, you can set up your Hadoop cluster. Go to hadooponazure.com and log in with your authorized Windows Live ID. Next, fill out the dialog boxes on the Web portal using the following values:

1. **Cluster (DNS) name:** Enter name in the form "*<your unique string>*.cloudapp.net".
2. **Cluster size:** Choose the number of nodes, from 4 to 32, and their associated storage allocations, from 2TB to 16TB per cluster.
3. **Administrator username and password:** Enter a username and password; password complexity restrictions are listed on the page. Once this is set you can connect via remote desktop or via Excel.
4. **Configuration information for a SQL Azure instance:** This is an option for storing the Hive Metastore. If it's selected, you'll need to supply the URL to your SQL Azure server instance, as well as the name of the target database and login credentials. The login you specify must have the following permissions on the target database: ddl_ddladmin, ddl_datawriter, ddl_datareader.

After you've filled in this information, click Request cluster. You'll see a series of status updates in the Web portal as your cluster (called Isotope in the beta) is being allocated, created and started. For each cluster you allocate, you'll see many worker nodes and one head node, which is also known as the NameNode.

After some period of time (five to 30 minutes in my experience), the portal will update to show that your cluster is allocated and ready for use. You can then simply explore the Metro-style

Hadoop on Windows Azure is currently in private beta. All information is subject to change.

This article discusses:

- Setting up a Hadoop cluster on Windows Azure
- Alternatives for connecting to your data
- Running MapReduce jobs with Java, JavaScript and C# Streaming
- Using HiveQL to query a Hive table

Technologies discussed:

Windows Azure, Hadoop, Java, JavaScript, C# Streaming, HiveQL

What Is Hadoop?

Hadoop is an open source library designed to batch-process massive data sets in parallel. It's based on the Hadoop distributed file system (HDFS), and consists of utilities and libraries for working with data stored in clusters. These batch processes run using a number of different technologies, such as Map/Reduce jobs, and may be written in Java or other higher-level languages, such as Pig. There are also languages that can be used to query data stored in a Hadoop cluster. The most common language for query is HQL via Hive. For more information, visit hadoop.apache.org.

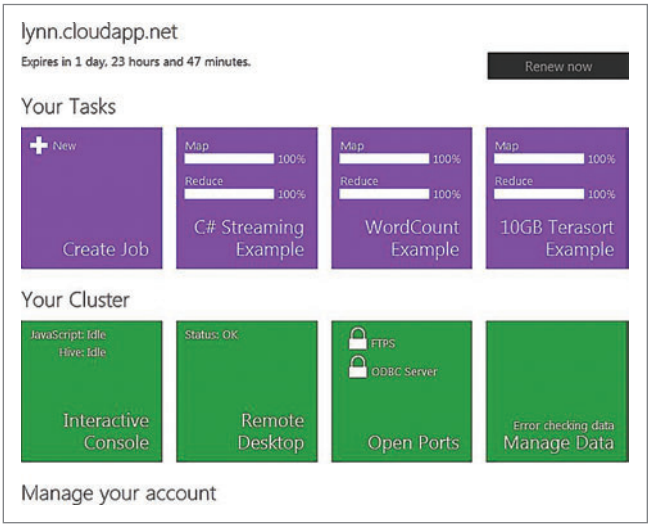


Figure 1 **The Windows Azure Hadoop Portal**

interface (by clicking the large buttons) to see what types of data processing and management tasks you can perform (see **Figure 1**). In addition to using the Web portal to interact with your cluster, you might want to open the available ports (closed by default) for FTP or ODBC Server access. I'll discuss some alternative methods of connecting in a bit.

In the Your Cluster section of the portal, you can perform basic administrative tasks such as configuring access to your cluster, importing data and managing the cluster via the interactive console. The interactive console supports JavaScript or Hive. As **Figure 1** shows, you can also access the Your Tasks section. Here you can run a MapReduce job (via a .jar file) and see the status of any MapReduce jobs that are running as well as those that have recently completed.

The portal buttons display information about three recently completed MapReduce jobs: C# Streaming Example, Word Count Example and 10GB Terasort Example. Each button shows the status of both the Map and the Reduce portion of each job. There are several other options for viewing the status of running (or completed) MapReduce jobs directly from the portal and via other means of connecting to your cluster, such as Remote Desktop Protocol (RDP).

Connecting to Your Data

You can make data available to your Hadoop on Windows Azure cluster in a number of ways, including directly uploading to your cluster and accessing data stored in other locations.

Although FTP allows uploading theoretically any size data files, best practice is to upload files that are in a lower gigabyte-size range. If you want to run batch jobs on data stored outside of Hadoop, you'll need to perform a couple of configuration steps first. To set up outside connections, click on the Manage Cluster button on the main portal and then configure the storage locations you wish to use, such as a Windows Azure Blob storage location, a Windows Azure Data Market query result or an Amazon Web Services (AWS) S3 storage location:

1. To configure a connection to an AWS S3 bucket, enter your security keys (both public and private) so you can access data stored on S3 in your Hadoop cluster.
2. To work with data from the Windows Azure Data Market, fill in values for username (WLID), passkey (for the data source you wish to query and import), source (extract) query and (destination) Hive table name. Be sure to remove the parameter for default query limit (100 rows) from the query generated by the tools in the Data Market before you enter the query into the text box on your cluster.
3. To access data stored in Windows Azure Blob storage, you'll need to enter the storage account name (URL) to the Blob storage locations and your passkey (private key) value.

Running a MapReduce Job

After setting up and verifying your Hadoop cluster, and making your data available, you'll probably want to start crunching this data by running one or more MapReduce jobs. The question is, how best to start? If you're new to Hadoop, there are some samples you can run to get a feel of what's possible. You can view and run any of these by clicking the Samples button on the Web portal.

If you're experienced with Hadoop techniques and want to run your own MapReduce job, there are several methods. The method you select will depend on your familiarity with the Hadoop tools (such as the Hadoop command prompt) and your preferred language. You can use Java, Pig, JavaScript or C# to write an executable MapReduce job for Hadoop on Windows Azure.

I'll use the Word Count sample to demonstrate how to run a MapReduce job from the portal using a .jar file. As you might expect, this job counts words for some input—in this example a large text file (the contents of an entire published book)—and outputs the result. Click Samples, then WordCount to open the job configuration page on the portal, as shown in **Figure 2**.

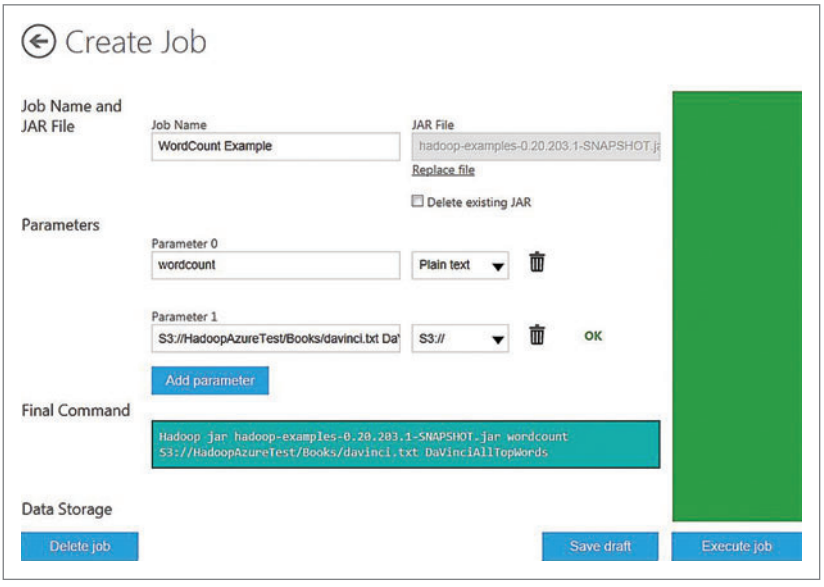


Figure 2 **Setting Up the WordCount Sample**

You'll see two configurable parameters for this job, one for the function (word count) and the other for the source data (the text file). The source data (Parameter 1) includes not only the name of the input file, but also the path to its location. This path to the source data file can be text, or it can be "local," which means that the file is stored on this Hadoop Windows Azure cluster. Alternatively, the source data can be retrieved from AWS S3 (via the S3n:// or the S3:// protocol), from the Windows Azure Blob storage (via the ASV:// protocol) or from the Windows Azure Data Market (by first importing the desired data using a query), or be retrieved directly from the HDFS store. After you enter the path to a remote location, you can click on the verification icon (a triangle) and you should get an OK message if you can connect using the string provided.

After you configure the parameters, click Execute job. You'll find a number of ways to monitor both job status as the job is executing and job results after the job completes. For example, on the main portal page, the Your Tasks section displays a button with the status of the most recent jobs during execution and after completion. A new button is added for each job, showing the job name, percentage complete for both the Map and the Reduce portions during execution, and the status (OK, failed and so forth) after job completion.

The Job History page, which you can get to from the Manage Your Account section of the main page, provides more detail about the job, including the text (script) used to run the job and the status, with date and time information. You can click the link for each job to get even more information about job execution.

If you decide to run a sample, be sure to read the detailed instructions for that particular sample. Some samples can be run from the Web portal (Your Tasks | Create Job); others require an RDP connection to your cluster.

Using JavaScript to Run Jobs

Click on the Interactive Console button to open the JavaScript console. Here you can run MapReduce jobs by executing .jar files (Java) by running a Pig command from the prompt, or by writing and executing MapReduce jobs directly in JavaScript.

You can also directly upload source data from the js> prompt using the fs.put command. This command opens a dialog box where you can choose a file to upload to your cluster. IIS limits the size of the file you can upload via the JavaScript console to 4GB.

You can also use source data from other remote stores (such as Windows Azure Blobs) or from other cloud vendors. To work with source data from AWS S3, you use a request in the format *s3n://<bucket name>/<folder name>*.

Using the JavaScript console, you can verify connectivity to your AWS S3 bucket by using the #ls command with the bucket address, like so:

```
js> #ls s3n://HadoopAzureTest/Books
Found 2 items
-rwxrwxrwx   1          0 2012-03-30 00:20 /Books
-rwxrwxrwx   1    1395667 2012-03-30 00:22 /Books/davinci.txt
```

When you do, you should get a list of the contents (both folders and files) of your bucket as in this example.

If you'd like to review the contents of the source file before you run your job, you can do so from the console with the #cat command:

```
js> #cat s3n://HadoopAzureTest/Books/davinci.txt
```

After you verify that you can connect to your source data, you'll want to run your MapReduce job. The following is the JavaScript syntax for the Word Count sample MapReduce job (using a .jar file):

```
var map = function (key, value, context) {
  var words = value.split(/[^a-zA-Z]/);
  for (var i = 0; i < words.length; i++) {
    if (words[i] !== "") {
      context.write(words[i].toLowerCase(), 1);
    }
  }
};var reduce = function (key, values, context) {
  var sum = 0;
  while (values.hasNext()) {
    sum += parseInt(values.next());
  }
  context.write(key, sum);
};
```

In the map portion, the script splits the source text into individual words; in the reduce portion, identical words are grouped and then counted. Finally, an output (summary) file with the top words by count (and the count of those words) is produced. To run this WordCount job directly from the interactive JavaScript console, start with the pig keyword to indicate you want to run a Pig job. Next, call the from method, which is where you pass in the location of the source data. In this case, I'll perform the operation on data stored remotely—in AWS S3.

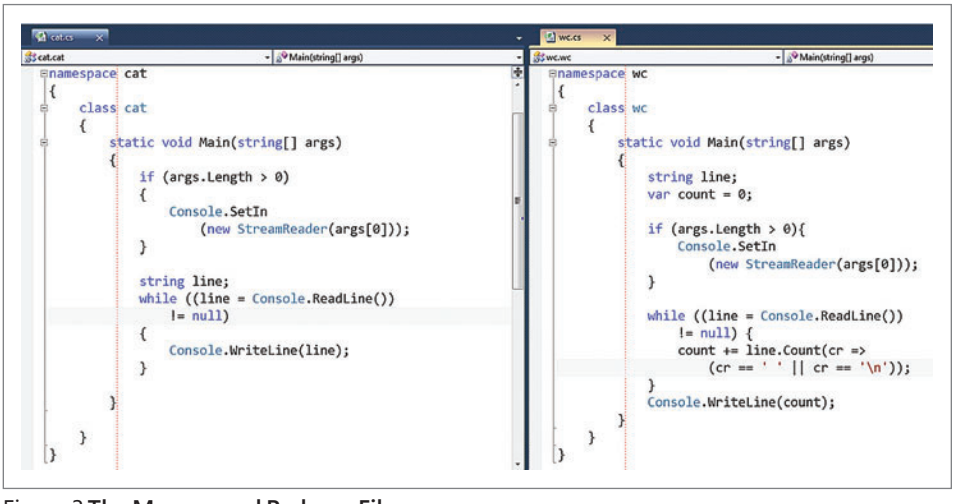


Figure 3 **The Mapper and Reducer Files**

Now you call the mapReduce method on the Pig job, passing in the name of the file with the JavaScript code for this job, including the required parameters. The parameters for this job are the method of breaking the text—on each word—and the value and datatype of the reduce aggregation. In this case, the latter is a count (sum) of data type long.

You then specify the output order using the orderBy method, again passing in the parameters; here the count of each group of words will be output in descending order. In the next step, the take method specifies

how many aggregated values should be returned—in this case the 10 most commonly occurring words. Finally, you call the to method, passing in the name of the output file you want to generate. Here's the complete syntax to run this job:

```
pig.from("s3n://HadoopAzureTest/Books").mapReduce("WordCount.js", "word,
count:long").orderBy("count DESC").take(10).to("DaVinciTop10Words.txt")
```

As the job is running, you'll see status updates in the browser—the percent complete of first the map and then the reduce job. You can also click a link to open another browser window, where you'll see more detailed logging about the job progress. Within a couple of minutes, you should see a message indicating the job completed successfully. To further validate the job output, you can then run a series of commands in the JavaScript console.

The first command, fs.read, displays the output file, showing the top 10 words and the total count of each in descending order. The next command, parse, shows the same information and will populate the data variable with the list. The last command, graph.bar, displays a bar graph of the results. Here's what these commands look like:

```
js> file = fs.read("DaVinciTop10Words.txt")
js> data = parse(file.data, "word, count:long")
js> graph.bar(data)
```

An interesting aspect of using JavaScript to execute MapReduce jobs is the terseness of the JavaScript code in comparison to the Java. The MapReduce WordCount sample Java job contains more than 50 lines of code, but the JavaScript sample contains only 10 lines. The functionality of both jobs is similar.

Using C# with Hadoop Streaming

Another way you can run MapReduce jobs in Hadoop on Windows Azure is via C# Streaming. You'll find an example showing how to do this on the portal. As with the previous example, to try out this sample, you need to upload the needed files (davinci.txt, cat.exe and wc.exe) to a storage location such as HDFS, ASV or S3. You also need to get the IP address of your Hadoop HEADNODE. To get the value using the interactive console, run this command:

```
js>#cat file:///apps/dist/conf/core-site.xml
```

Fill in the values on the job runner page; your final command will look something like this:

```
Hadoop jar hadoop-examples-0.20.203.1-SNAPSHOT.jar
-files "hdfs:///example/apps/wc.exe,hdfs:///example/apps/cat.exe"
-input "/example/data/davinci.txt"
-output "/example/data/StreamingOutput/wc.txt"
-mapper "cat.exe"
-reducer "wc.exe"
```

In the sample, the mapper and the reducer are executable files that read the input from stdin, line by line, and emit the output to stdout. These files produce a Map/Reduce job, which is submitted to the cluster for execution. Both the mapper file, cat.exe, and the reducer file, wc.exe, are shown in **Figure 3**.

Here's how the job works. First the mapper file launches as a process on mapper task initialization. If there are multiple mappers, each will launch as a separate process on initialization. In this case, there's only a single mapper file—cat.exe. On execution, the mapper task converts the input into lines and feeds those lines to the stdin portion of the MapReduce job. Next, the mapper gathers the line outputs from stdout and converts each line into a key/value pair. The default behavior (which can be changed) is that the key is created from the line prefix up to the first tab character and the value is created from the remainder of the line. If there's no tab in the line, the whole line becomes the key and the value will be null.

After the mapper tasks are complete, each reducer file launches as a separate process on reducer task initialization. On execution, the reducer converts key/value input pairs into lines and feeds those lines to the stdin process. Next, the reducer collects the line-oriented outputs from the stdout process and converts each line into a key/value pair, which is collected as the output of the reducer.

Using HiveQL to Query a Hive Table

Using the interactive Web console, you can execute a Hive query against Hive tables you've defined in your Hadoop cluster. To learn more about Hive, see hive.apache.org.

To use Hive, you first create (and load) a Hive table. Using our WordCount MapReduce sample output file (DavinciTop10-Words.txt), you can execute the following command to create and then verify your new Hive table:

```
hive> LOAD DATA INPATH
 'hdfs://lynnlangit.cloudapp.net:9000/user/lynnlangit/DaVinciTop10Words.txt'
 OVERWRITE INTO TABLE wordcounttable;
 hive> show tables;
 hive> describe wordcounttable:
 hive> select * from wordcounttable;
```

Hive syntax is similar to SQL syntax, and HiveQL provides similar query functionality. Keep in mind that all data is case-sensitive by default in Hadoop.

Other Ways to Connect to Your Cluster

Using RDP In addition to working with your cluster via the Web portal, you can also establish a remote desktop connection to the cluster's NameNode server. To connect via RDP, you click the Remote Desktop button on the portal, then click on the downloaded RDP connection file and, when prompted, enter your administrator username and password. If prompted, open firewall ports on your client machine. After the connection is established, you can work directly with your cluster's NameNode using the Windows Explorer shell or other tools that are included with the Hadoop installation, much as you would with the default Hadoop experience.

My NameNode server is running Windows Server 2008 R2 Enterprise SP1 on a server with two processors and 14GB of RAM, with Apache Hadoop release 0.20.203.1 snapshot installed. Note that the cluster resources consist of the name node and the associated worker nodes, so the total number of processors for my sample cluster is eight.

The installation includes standard Hadoop management tools, such as the Hadoop Command Shell or command-line interface (CLI), the Hadoop MapReduce job tracker (found at http://[namenode]:50030) and the Hadoop NameNode HDFS (found at http://[namenode]:50070). Using the Hadoop Command Shell you can run MapReduce jobs or other administrative tasks (such as managing your DFS cluster state) via your RDP session.

At this time, you can connect via RDP using only a Windows client machine. Currently, the RDP connection uses a cookie to enable port forwarding. The Remote Desktop Connection for Mac client doesn't have the ability to use that cookie, so it can't connect to the virtual machine.

Using the Sqoop Connector Microsoft shipped several connectors for Hadoop to SQL Server in late 2011 (for SQL Server 2008 R2 or later or for SQL Server Parallel Data Warehouse). The Sqoop-based SQL Server connector is designed to let you import or export data between Hadoop on Linux and SQL Server. You can download the connector from bit.ly/JgFmm3. This connector requires that the JDBC driver for SQL Server be installed on the same node as Sqoop. Download the driver at bit.ly/LAlU4F.

You'll find an example showing how to use Sqoop to import or export data between SQL Azure and HDFS in the samples section of the portal.

Using FTP To use FTP, you first have to open a port, which you can do by clicking the Configure Ports button on the portal and then dragging the slider to open the default port for FTPS (port 2226). To communicate with the FTP server, you'll need an MD5 hash of the password for your account. Connect via RDP, open the *users.conf* file, copy the MD5 hash of the password for the account that will be used to transfer files over FTPS and then use this value to connect. Note that the MD5 hash of the password uses a self-signed certificate on the Hadoop server that might not be fully trusted.

You can also open a port for ODBC connections (such as to Excel) in this section of the portal. The default port number for the ODBC Server connections is 10000. For more complex port configurations, though, use an RDP connection to your cluster.

Using the ODBC Driver for Hadoop (to Connect to Excel and PowerPivot) You can download an ODBC driver for Hadoop from the portal Downloads page. This driver, which includes an add-in for Excel, can connect from Hadoop to either Excel or PowerPivot. **Figure 4** shows the Hive Pane button that's added to Excel after you install the add-in. The button exposes a Hive Query pane where you can establish a connection to either a locally hosted Hadoop server or a remote instance. After doing so, you can write and execute a Hive query (via HiveQL) against that cluster and then work with the results that are returned to Excel.

You can also connect to Hadoop data using PowerPivot for Excel. To connect to PowerPivot from Hadoop, first create an OLE DB for ODBC connection using the Hive provider. On the Hive Query pane, next connect to the Hadoop cluster using the connection you configured previously, then select the Hive tables (or write a HiveQL query) and return the selected data to PowerPivot.

Be sure to download the correct version of the ODBC driver for your machine hardware and Excel. The driver is available in both 32-bit and 64-bit editions.

Easy and Flexible—but with Some Unknowns

The Hadoop on Windows Azure beta shows several interesting strengths, including:

- Setup is easy using the intuitive Metro-style Web portal.
- You get flexible language choices for running MapReduce jobs and data queries. You can run MapReduce jobs using Java, C#, Pig or JavaScript, and queries can be executed using Hive (HiveQL).
- You can use your existing skills if you're familiar with Hadoop technologies. This implementation is compliant with Apache Hadoop snapshot 0.203+.
- There are a variety of connectivity options, including an ODBC driver (SQL Server/Excel), RDP and other clients, as well as connectivity to other cloud data stores from Microsoft (Windows Azure Blobs, the Windows Azure Data Market) and others (Amazon Web Services S3 buckets).

There are, however, many unknowns in the version of Hadoop on Windows Azure that will be publicly released:

- The current release is a private beta only; there is little information on a roadmap and planned release features.
- Pricing hasn't been announced.
- During the beta, there's a limit to the size of files you can upload, and Microsoft included a disclaimer that "the beta is for testing features, not for testing production-level data loads." So it's unclear what the release-version performance will be like.

To see video demos (screencasts) of the beta functionality of Hadoop on Windows Azure, see my BigData Playlist on YouTube at bit.ly/LyX7Sj. ■

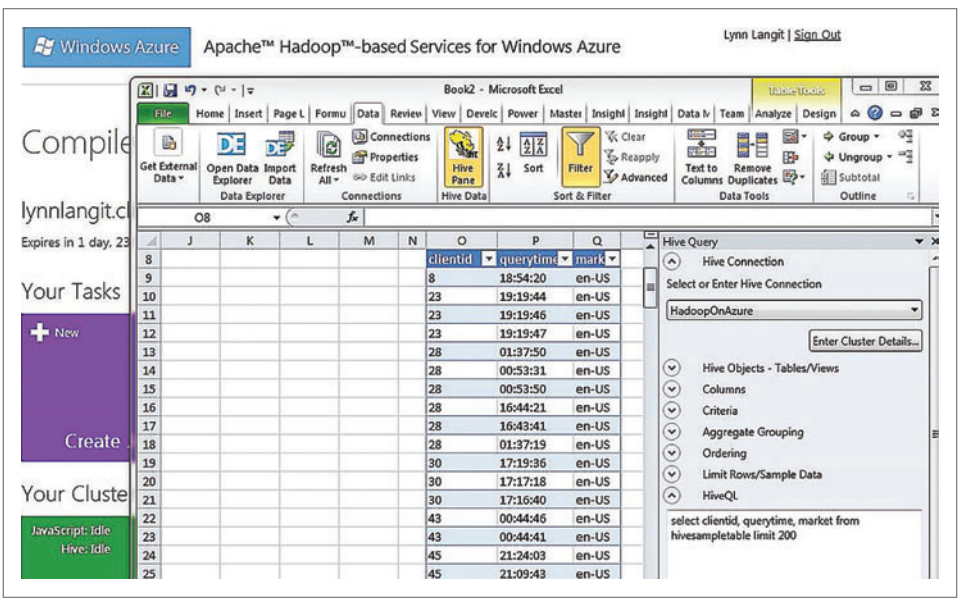


Figure 4 **The Hive Query Pane in Excel**

LYNN LANGIT *(LynnLangit.com) runs her own technical training and consulting company. She designs and builds data solutions that include both RDBMS and NoSQL systems. She recently returned to private practice after working as a developer evangelist for Microsoft for four years. She is the author of three books on SQL Server Business Intelligence, most recently "Smart Business Intelligence Solutions with SQL Server 2008" (Microsoft Press, 2009). She is also the cofounder of the non-profit TKP (TeachingKidsProgramming.org).*

THANKS to the following technical expert for reviewing this article: *Denny Lee*

TECHMENTOR CONFERENCES

AUGUST 20-24
MICROSOFT HQ
REDMOND, WA

Powered by Microsoft | TechNet

Only one event in 2012!

IN-DEPTH TRAINING FOR IT PROS

This year, TechMentor is a **ONE-TIME-ONLY** event at a new special location – Microsoft Headquarters!

Receive cutting-edge and practical education for the IT professional! Learn from IT experts and industry insiders on topics such as:

* Windows 8 Server
* MCITP Training
* Security
* Windows PowerShell
* Virtualization
* Cloud Computing

. . . and much more!

* Visit the Microsoft Campus in Redmond, Washington
* Receive Unbiased, Immediately Usable and In-Depth Tech Training
* Network with Peers, IT Experts and Microsoft Insiders

techmentorevents.com

Register before July 18 and Save $200!

Use promo code JULY1AD

Visit techmentorevents.com or scan the QR code to register and for more event details.

Produced By: 1105 MEDIA Inc

Supported By: Redmond MAGAZINE | Microsoft | TechNet | VIRTUALIZATION REVIEW

How to Handle Relational Data in a Distributed Cache

Iqbal Khan

The Microsoft .NET Framework has become popular for developing an array of high-transaction applications including Web, service-oriented architecture (SOA), high-performance computing or grid computing, and cloud computing. All these application architectures are scalable. But a major bottleneck occurs in the data storage—normally a relational database—which isn't able to scale and handle the increased transaction load that the application tier can.

As a result, distributed caching is becoming quite popular because it allows you to cache data in an in-memory cache that's much faster to access than any database. Also, distributed caching provides linear scalability across multiple cache servers. With linear scalability, you can add more cache servers to the distributed cache cluster if you need to handle a bigger transaction load. The incremental gain in transactional capacity doesn't diminish as you add more servers (it's a straight line in an X-Y graph where X is the number of cache servers and Y is the transaction capacity). **Figure 1** shows how distributed caching fits into a typical ASP.NET or Windows Communication Foundation (WCF) application and how it provides linear scalability.

This article discusses:

- Object relation mapping
- CacheDependency
- Relationship types
- Caching strategies for different types
- Handling collections

Technologies discussed:

ASP.NET, Windows Communication Foundation

In this article, I'll discuss how developers should handle data relationships while caching their data.

Although distributed caching is great, one challenge it presents is how to cache relational data that has various relationships between data elements. This is because a distributed cache provides you a simple, Hashtable-like (key, value) interface where the "key" uniquely identifies a cached item and "value" is your data or object. However, your application probably has a complex domain object model with inheritance, aggregation and other types of relationships.

Additionally, a distributed cache stores individual cached items separately, which is different from the database where you have relationships between different tables. Usually, there are no relationships between the different cached items in a typical distributed cache. That poses a challenge to .NET application developers. In other words, you're facing the challenge of tracking a lot of the relationship information inside your application to make sure you can handle relationships correctly in the cache, and at the same time take advantage of distributed caching.

I'll explain how these relationships can be mapped and provide source code examples. The net effect is that the applications don't have to keep track of these relationships themselves. Rather, the cache can be made aware of them and then handle them independently.

Object Relation Mapping Is Good

First, make sure you transform your relational data into a domain object model. Without this step, you'd have a difficult time handling all of the relationships in a distributed cache.

In any .NET application, you're most likely using DataReader (SqlDataReader, OracleDataReader or OleDbDataReader) or DataTable to retrieve data from the database, which is fine. But many developers then directly access data from these objects (especially DataTable) throughout their application. Some do it out of laziness, because they don't want to create custom domain objects. Others do it because they believe they can use intelligent filtering capabilities in the DataTable.

I strongly recommend you transform your DataReader or DataTable into a domain object model. This will greatly simplify your application design and also allow you to use distributed caching effectively. And a good distributed cache usually provides you with a SQL-like or LINQ querying capability so you won't miss the filtering features of DataTable.

When using a DataTable, you're forced to cache it as one cached item. Note that a large number of rows in a DataTable slows down your application when you cache it.

You can either transform DataReader and DataTable into domain objects manually or use one of the leading Object Relation Mapping (ORM) engines. The Entity Framework from Microsoft is one such engine. Another popular one is NHibernate, which is open source. An ORM tool greatly simplifies your task of transforming relational data into a domain object model.

CacheDependency Helps Manage Relationships

With a domain object model, the first question is how to handle all those relationships in the cache. And the answer is CacheDependency, which is part of ASP.NET Cache and is now also found in some commercial distributed caching solutions.

CacheDependency allows you to inform the cache about different types of relationships between cached items and then let the distributed cache manage data integrity for them. Basically, a CacheDependency allows you to tell the cache that one cached item depends on another cached item. Then the distributed cache can track any changes in the target item and invalidate the source item that depends on the target item.

Let's say that if data item *A* depends on *B*, then if *B* is ever updated or removed from the cache, the distributed cache automatically removes *A* as well. CacheDependency also provides cascading capability, so if *A* depends on *B* and *B* depends on *C* and then *C* is updated or removed, it results in *B* being automatically removed by the distributed cache. And when that happens, *A* is also automatically removed by the distributed cache. This is called the cascading CacheDependency.

I use CacheDependency later in this article to demonstrate how to handle relationships in a distributed cache.

Three Different Types of Relationships

First of all, let's use an example data model for discussion purposes, shown in **Figure 2**.

As you can see, in this data model, Customer has a one-to-many relationship with Order; Product has a one-to-many relationship with Order; and Customer and Product have a many-to-many relationship with each other through the Order table. For our example data model, **Figure 3** shows the equivalent object model that represents the same relationships.

Before I go into the details of different relationship types, I want to explain one thing. Unlike the database, an application domain object model always has a primary object that the application has fetched from the database and that is therefore the starting point for the application during a particular transaction. All other objects are seen as related to this primary object. This concept is valid for all types of relationships and affects how you see relationships in a domain object model. Now, let's proceed.

One-to-One and Many-to-One Relationships

One-to-one and many-to-one relationships are similar. In a one-to-one relationship between table *A* and table *B*, one row in table *A* is related to only one row in table *B*. You keep a foreign key in either table *A* or *B* that is the primary key of the other table.

In the case of a many-to-one relationship between *A* and *B*, you must keep the foreign key in table *A*. This foreign key is the

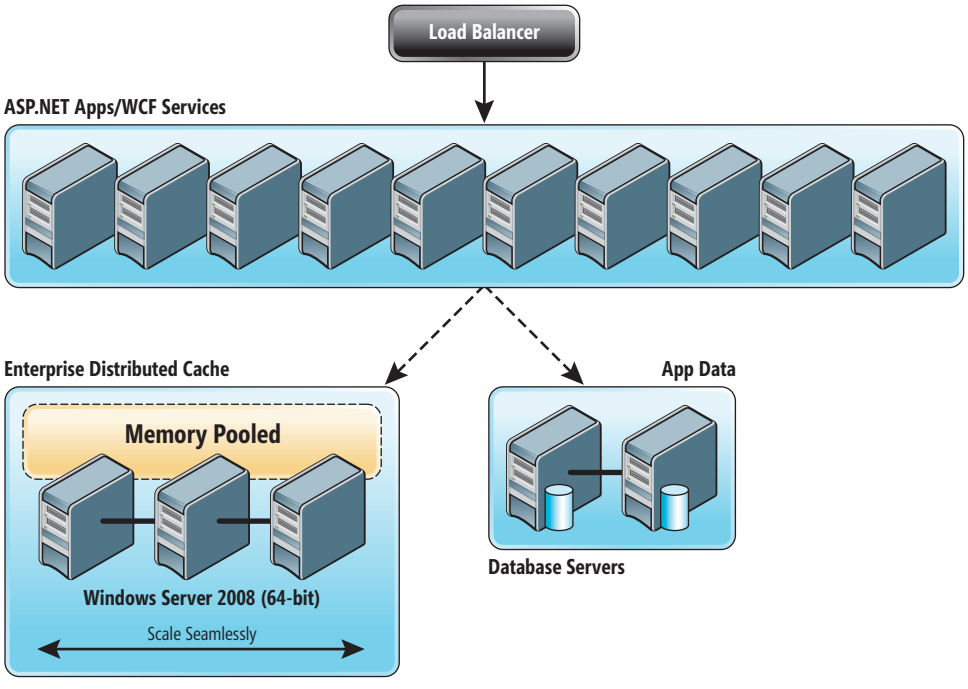


Figure 1 **Distributed Cache Used in ASP.NET or Windows Communication Foundation Applications**

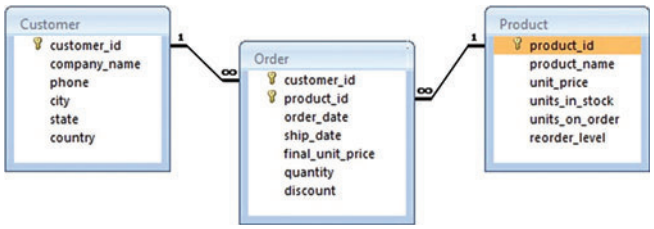


Figure 2 **Example Data Model for Relationships**

primary key of table *B*. In both one-to-one and many-to-one relationships, the foreign key has a unique constraint to make sure there are no duplicates.

The same relationship can easily be transformed into a domain object model. The primary object (explained earlier) keeps a reference to the related object. **Figure 4** shows an example of the primary object keeping relationship information. Note that the Order class contains a reference to the Customer class to indicate a many-to-one relationship. The same would happen even in a one-to-one relationship.

One-to-Many Relationships (Reverse of Many-to-One)

In the case of a one-to-many relationship in the database between tables *A* and *B*, table *B* (meaning the "many side") keeps the foreign key, which is actually the primary key of table *A*, but without a unique constraint on the foreign key.

In the case of a one-to-many domain object model, the primary object is the Customer and the related object is the Order. So the Customer object contains a collection of Order objects. **Figure 4** also shows an example of this relationship between the Customer and Order objects.

Many-to-Many Relationships

In the case of a many-to-many relationship between tables *A* and *B*, there's always an intermediary table *AB*. In our case, Order is the intermediary table and has two foreign keys. One is against the Customer table to represent a many-to-one relationship and the other is against the Order table to again represent a many-to-one relationship.

In case of a many-to-many relationship, the object model is usually one-to-many from the perspective of a primary object (defined earlier) that's either Customer or Product. The object model also now contains a many-to-one relationship between the intermediary object (Order, in our case) and the other object (Product, in our case). **Figure 4** also shows an example of a many-to-many relationship, which in this case is between Customer and Product objects, with the Order object being an intermediary object.

As you can see, the object model is from the perspective of the Customer object that's the primary object, and the application fetched it from the database. All related objects are from the perspective of this primary object. So a Customer object will have a collection of Order objects, and each Order object will contain a reference to the Product object. The Product object probably won't contain a collection of all Order objects belonging to the Product object because that isn't needed here. If Product were the primary object, it would have a collection of Order objects—but then the Customer object wouldn't have a collection of Order objects.

Caching Strategies for Different Relationship Types

So far, I've discussed how to fetch data from the database, transform it into a domain object model and keep the same relationships in the domain object model as in the database—although from the perspective of the primary object. But if your application wants to cache data in a distributed cache, you must understand how to handle all these relationships in the cache. I'll go through each case.

In all cases, you must ensure that relationship information in the cache isn't lost, impacting either data integrity or your ability to fetch related objects from the cache later.

Primary Object

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

namespace Relationships
{
    [Serializable]
    public class Customer
    {
        public long          CustomerId;
        public string        CompanyName;
        public string        Phone;
        public string        City;
        public string        State;
        public string        Country;

        // one-to-many list of Order objects
        public IList<Order> Orders;
    }
}
```

Intermediary Related Object

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

namespace Relationships
{
    [Serializable]
    public class Order
    {
        public long     CustomerId;
        public long     ProductId;
        public DateTime OrderDate;
        public DateTime ShipDate;
        public double   FinalUnitPrice;
        public long     Quantity;
        public double   Discount;

        // reference to the related object
        public Customer OrderingCustomer;

        // reference to the related object
        public Product OrderedProduct;
    }
}
```

Many-to-many Related Object

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

namespace Relationships
{
    [Serializable]
    public class Product
    {
        public long          ProductId;
        public string        ProductName;
        public double        UnitPrice;
        public long          UnitsInStock;
        public long          UnitsOnOrder;
        public long          ReOrderLevel;

        // one-to-many list of Order objects
        public IList<Order> Orders;
    }
}
```

Figure 3 **Example Object Model Against the Data Model**

Figure 4 **Caching a Related Object Separately**

```
public void CacheOrder(Order order)
{
  Cache cache = HttpRuntime.Cache;
  DateTime absolutionExpiration = Cache.NoAbsoluteExpiration;
  TimeSpan slidingExpiration = Cache.NoSlidingExpiration;
  CacheItemPriority priority = CacheItemPriority.Default;

  if (order != null) {
    // This will prevent Customer from being cached with Order
    Customer cust = order.OrderingCustomer;

    // Set orders to null so it doesn't get cached with Customer
    cust.Orders = null;
    string custKey = "Customer:CustomerId:" + cust.CustomerId;
    cache.Add(custKey, cust, null,
              absolutionExpiration,
              slidingExpiration,
              priority, null);

    // Dependency ensures order is removed if Cust updated/removed
    string[] keys = new string[1];
    keys[0] = custKey;
    CacheDependency dep = new CacheDependency(null, keys);
    string orderKey = "Order:CustomerId:" + order.CustomerId
      + ":ProductId:" + order.ProductId;

    // This will only cache Order object
    cache.Add(orderKey, order, dep,
              absolutionExpiration,
              slidingExpiration,
              priority, null);
  }
}
```

Caching One-to-One and Many-to-One Relationships

You have two options here:

1. **Cache related objects with the primary object:** This option assumes that the related object isn't going to be modified by another user while it's in the cache, so it's safe to cache it as part of the primary object as one cached item. If you fear this isn't the case, then don't use this option.
2. **Cache related objects separately:** This option assumes that the related object might be updated by another user while it's in the cache, so it's best to cache the primary and related objects as separate cached items. You should specify unique cache keys for each of the two objects. Additionally, you can use the tag feature of a distributed cache to tag the related object as having a relationship to the primary object. Then you could fetch it later through the tag.

Caching One-to-Many Relationships

In the case of one-to-many relationships, your primary object is always on the "one side" (in our example it's the Customer object). The primary object contains a collection of Order objects. Each collection of related objects represents a one-to-many relationship. Here you have three caching options:

1. **Cache related objects collections with the primary object:** This of course assumes that related objects won't be updated or fetched independently by another user while in the cache, so it's safe to cache them as part of the primary object. Doing so improves your performance because you can fetch everything in one cache call. However, if the collection is large (meaning tens of thousands of objects and reaching into megabytes of data), you won't get the performance gain.
2. **Cache related objects collections separately:** In this situation, you believe other users might want to fetch the same collections from the cache; therefore, it makes sense to cache the collection of related objects separately. You should structure your cache key in such a way that you'll be able to find this collection based on some information about your primary object. I'll discuss the issue of caching collections in much more detail later in this article.
3. **Cache all individual related objects from collections separately:** In this situation, you believe each individual object in the related collection might be updated by other users; therefore, you can't keep an object in the collection and must cache it separately. You can use the tag feature of a distributed cache to identify all objects as related to your primary object so you'll be able to fetch them quickly later on.

Cache Many-to-Many Relationships

Many-to-many relationships in a domain object model don't really exist. Instead, they're represented by one-to-many relationships, with the exception that the intermediary object (Order, in our case) contains a reference to the other side object (Product, in our case). In a pure one-to-many, this reference wouldn't exist.

Handling Collections

The topic of how to handle collections is an interesting one, because you often fetch a collection of objects from the database and you want to be able to cache collections in an efficient way.

Let's say you have a scenario where you request all your New York-based customers and you don't expect any new customers

Figure 5 **Caching a Related Collection Separately**

```
public void CacheCustomer(Customer cust)
{
  Cache cache = HttpRuntime.Cache;
  DateTime absolutionExpiration = Cache.NoAbsoluteExpiration;
  TimeSpan slidingExpiration = Cache.NoSlidingExpiration;
  CacheItemPriority priority = CacheItemPriority.Default;

  if (cust != null)
  {
    string key = "Customer:CustomerId:" + cust.CustomerId;

    // Let's preserve it to cache separately
    IList<Order> orderList = cust.Orders;

    // So it doesn't get cached as part of Customer
    cust.Orders = null;

    // This will only cache Customer object
    cache.Add(key, cust, null,
              absolutionExpiration,
              slidingExpiration,
              priority, null);

    // See that this key is also Customer-based
    key = "Customer:CustomerId:" + cust.CustomerId + ":Orders";
    cache.Add(key, orderList, null,
              absolutionExpiration,
              slidingExpiration,
              priority, null);
  }
}
```

Figure 6 **Caching Each Collection Item Separately**

```
public void CacheOrdersListItems(IList<Order> ordersList)
{
  Cache cache = HttpRuntime.Cache;
  DateTime absolutionExpiration = Cache.NoAbsoluteExpiration;
  TimeSpan slidingExpiration = Cache.NoSlidingExpiration;
  CacheItemPriority priority = CacheItemPriority.Default;

  foreach (Order order in ordersList)
  {
    string key = "Order:CustomerId:" + order.CustomerId
      + ":ProductId" + order.ProductId;

    string[] keys = new string[1];
    keys[0] = key;

    // Dependency ensures Order is removed if Cust updated/removed
    CacheDependency dep = new CacheDependency(null, keys);

    Tag [] tagList = new Tag [1];
    tagList[0] = new Tag ("customerId" + order.CustomerId);

     // Tag allows us to find order with a customerId alone
     cache.Add(key, order, dep,
               absolutionExpiration,
               slidingExpiration,
               priority, null, tagList);
  }
}
```

to be added from New York in the next day or in the next few minutes. Keep in mind you aren't caching data for weeks and months, only for a minute or a few hours in most cases. In some situations you might cache it for many days.

There are different caching strategies for caching collections, as I'll explain.

Cache an Entire Collection as One Item

In this case, you know you won't be adding any more customers from New York and that other users aren't accessing and modifying any New York customer data during the period in which the data will be cached. Therefore, you can cache the entire collection of New York customers as one cached item. Here, you can make the search criteria or the SQL query part of the cache key. Any time you want to fetch customers who are from New York, you just go to the cache and say, "Give me the collection that contains New York customers."

Figure 5 shows how an entire collection of related Order objects is cached.

Cache Each Collection Item Separately

Now let's move on to the second strategy. In this scenario, you or other users want to individually fetch and modify a New York customer. But the previous strategy would require everybody to fetch the entire collection from the cache, modify this one customer, put it back in the collection and cache the collection again. If you do this frequently enough, it will become impractical for performance reasons.

So, in this instance, you don't keep the collection of all the New York customers as one cached item. You break up the collection and store each Customer object separately in the cache. You need to group all these Customer objects so at a later point you can just fetch them all back in one call as either a collection or an IDictionary. The benefit of this approach is being able to fetch and modify individual Customer objects. **Figure 6** shows an example of how to cache each of the related objects in the collection separately.

Keep in mind, however, that this strategy assumes no New York customers have been added to the database in the period in which they've been cached. Otherwise, when you fetch all the New York customers from the cache, you'll get only a partial list. Similarly, if a customer is deleted from the database but not from the cache, you'll get stale list of customers.

Handling Collections Where Objects Are Added or Deleted

The third option for handling collections is where you think new customers may be added from New York or some existing customer may be deleted. In this case, whatever you cache is only partial data or old data. Perhaps the collection had only 100 customers and you added two more today. Those two won't be part of the cache. However, when you fetch all the customers from New York, you want the correct data: You want 102 results, not 100.

The way to ensure this happens is to make a call to the database. Get all the IDs for the customers and do a comparison to see which ones are in the cache and which ones aren't. Individually fetch from the database those that aren't in the cache and add them to the cache. As you can see, this isn't a fast process; you're making multiple database calls and multiple cache calls. With a distributed cache that provides basic features, if you have a collection of 1,000 customers and 50 new are added, you'll end up making 51 database calls and 101 distributed cache calls. In this case, it might be faster just to fetch the collection from the database in one call.

But if the distributed cache provides bulk operations, you'd make one database call to fetch IDs, one distributed cache call to see which IDs exist in the cache, one call to add all those new customers to the cache and one call to fetch the entire collection of customers from the cache. This would be a total of one database call and three distributed cache calls, and that isn't bad at all. And, if no new customers were added (which would be the case 90 percent of the time), this would be reduced to one database call and two distributed cache calls.

Performance and Scalability

I covered various situations that arise when your application fetches relational data from the database, transforms it into a domain object model and then wants to cache the objects. The purpose of my article is to highlight all those areas where object relationships affect how you cache the objects and how you later modify or remove them from the cache.

Distributed caching is a great way to improve your application's performance and scalability. And, because applications deal with relational data most of the time, I hope this article has provided you with some insight into how you should handle relational data in a distributed cache. ■

IQBAL KHAN *is the president and technology evangelist of Alachisoft, which provides NCache and StorageEdge (alachisoft.com). NCache is a distributed cache for .NET and Java and improves application performance and scalability. StorageEdge is an RBS provider for SharePoint and helps optimize storage and performance of SharePoint. Khan received his master's degree in computer science from Indiana University, Bloomington, in 1990. Reach him at iqbal@alachisoft.com.*

THANKS to the following technical expert for reviewing this article: *Damian Edwards*

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

YOUR MAP TO THE .NET DEVELOPMENT PLATFORM

BIRDS OF A FEATHER CODE TOGETHER

Orlando, FL | **December 10-14**

Royal Pacific Resort at Universal Orlando | **vslive.com/orlando**

Don't miss your chance for great .NET education in 2012 at Visual Studio Live! Orlando.

- **Visual Studio / .NET**
- **HTML5**
- **Cloud Computing and Services**
- **Windows 8 / WinRT**
- **Data Management**
- **Silverlight / WPF**
- **Windows Phone**

Register Today and Save $400!

Use Promo Code JULYAD

ROYAL PACIFIC RESORT LIVE! DECEMBER LIVE! ORLANDO 10 - 14

SUPPORTED BY

Microsoft | Visual Studio | msdn | Visual Studio MAGAZINE

PRODUCED BY

1105 MEDIA

MEDIA SPONSOR

THE CODE PROJECT WWW.CODEPROJECT.COM

Visit **vslive.com/orlando** or scan the QR code to register and for more event details.

A Smart Thermostat on the Service Bus

Clemens Vasters

Here's a bold prediction: Connected devices are going to be big business, and understanding these devices will be really important for developers not too far down the road. "Obviously," you say. But I don't mean the devices on which you might read this article. I mean the ones that will keep you cool this summer, that help you wash your clothes and dishes, that brew your morning coffee or put together other devices on a factory floor.

In the June issue of *MSDN Magazine* (msdn.magazine.com/magazine/jj133825), I explained a set of considerations and outlined an architecture for how to manage event and command flows from and to embedded (and mobile) devices using Windows Azure Service Bus. In this article, I'll take things a step further and look at code that creates and secures those event and command flows. And because a real understanding of embedded devices does require looking at one, I'll build one and then wire it up to Windows Azure Service Bus so it can send events related to its current state and be remotely controlled by messages via the Windows Azure cloud.

This article discusses:

- Designing the device in the .NET Gadgeteer
- Implementing local thermostat functionality
- Provisioning the device
- Sending events and receiving commands
- Security issues

Technologies discussed:

Visual Studio, Windows Azure Service Bus, .NET Micro Framework, .NET Gadgeteer

Code download available at:

code.msdn.microsoft.com/mag201207IOT

Until just a few years ago, building a small device with a power supply, a microcontroller, and a set of sensors required quite a bit of skill in electronics hardware design as well as in putting it all together, not to mention good command of the soldering iron. I'll happily admit that I've personally been fairly challenged in the hardware department—so much so that a friend of mine once declared if the world were attacked by alien robots he'd send me to the frontline and my mere presence would cause the assault to collapse in a grand firework of electrical shorts. But due to the rise of prototyping platforms such as Arduino/Netduino or .NET Gadgeteer, even folks who might do harm to man and machine swinging a soldering iron can now put together a fully functional small device, leveraging existing programming skills.

To stick with the scenario established in the last issue, I'll build an "air conditioner" in the form of a thermostat-controlled fan, where the fan is the least interesting part from a wiring perspective. The components for the project are based on the .NET Gadgeteer model, involving a mainboard with a microcontroller, memory and a variety of pluggable modules. The mainboard for the project is a GHI Electronics FEZ Spider board with the following extension modules:

- From GHI Electronics
  - Ethernet J11D Module to provide wired networking (a Wi-Fi module exists)
  - USB Client DP Module as power supply and USB port for deployment
  - Joystick for direct control of the device
- From Seeed Studio
  - Temperature and humidity sensor
  - Relays to switch the fan on or off
  - OLED display to show the current status

Together, these parts cost around $230. That's obviously more than soldering equivalent components onto a board, but did I mention

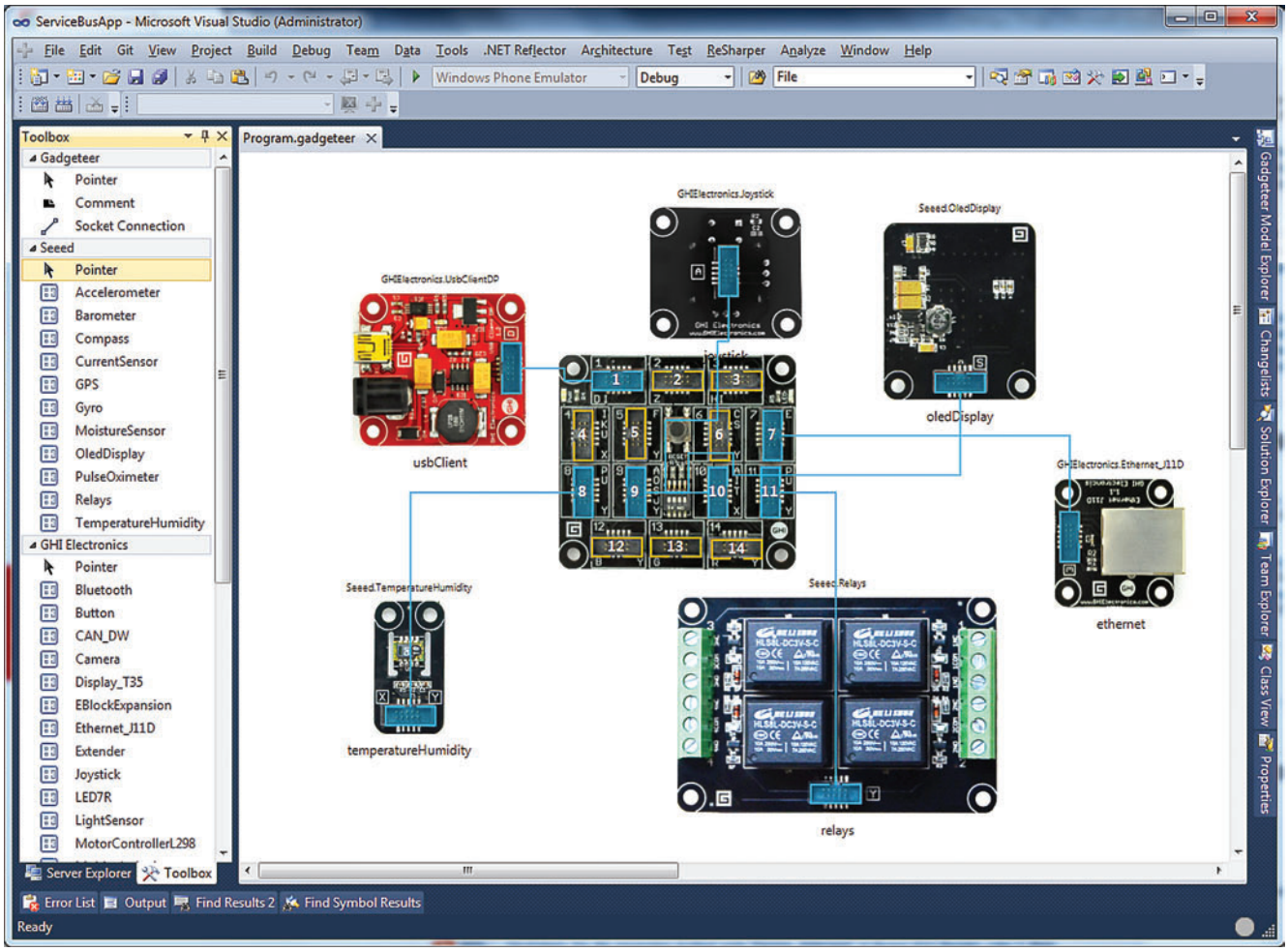


Figure 1 **Designing the Device in the .NET Gadgeteer**

that would require soldering? Also, this is a market that's just starting to get going, so expect prices to go down as the base broadens.

To make the components come alive you'll need Visual C# 2010 Express (at least), the .NET Micro Framework SDK, and the Gadgeteer SDK from GHI Electronics or Seeed. Once you have these installed, the development experience is—if you'll allow the superlative—fairly spectacular and as visual as things can conceivably get in Visual Studio, as you can see in **Figure 1**.

Figure 1 shows the design view of the .NET Gadgeteer program in Visual Studio. I thought of including a photo of the actual device with this article, but all the photo would do is confirm the diagram. This is exactly how it looks.

The file with the .gadgeteer extension contains an XML model that's visualized in the editor. From that XML file, the Gadgeteer tooling auto-generates a partial Program class with wrappers for each of the modules plugged into the mainboard. Your code sits in Program.cs holding another part of the Program class, just like the codebehind model you're familiar with from other .NET APIs.

You use the .NET Micro Framework with these devices. It's an entirely open source version of the Microsoft .NET Framework that has been specifically created for small devices with limited compute power and not much memory. The .NET Micro Framework contains many of the familiar .NET Framework classes, but most have gone through a feature diet to reduce the overall code footprint. Because the Framework is a layer over the native hardware of the device and the device isn't a general-purpose computer with an OS that handles all hardware abstraction (there's really no OS here), the framework version you can use with a device depends on the board manufacturer supporting the prerequisites, which is obviously very different from the experience with a regular PC, where particularities of hardware are far removed from things as high-level as the .NET Framework.

There are several other differences compared with the regular .NET Framework, and the PC platform in general, that are—coming from a PC background—initially surprising. For example, the device here doesn't have an onboard battery. No battery means no buffered clock, so the device has no idea of the correct wall-clock time when it wakes up. Lacking an OS, and with a third-party extension display, the device also doesn't have onboard fonts you can use to draw strings for display. If you want to display a string, you'll have to add a font to do so.

Likewise, the device doesn't have a prepopulated, Windows Update-maintained certificate store. If you want to validate SSL/TLS certificates, you'll have to deploy at least the root CA certificates into the device—and of course you'll also have to have the current time to check the certificate's validity. As you've might

Figure 2 **Reading Temperature and Humidity**

```
void WireEvents()
{
  this.InitializeTemperatureSensor();
  this.InitializeJoystick();
}

void InitializeTemperatureSensor()
{
  this.temperatureCheckTimer = new Timer(5000);
  this.temperatureCheckTimer.Tick += (t) =>
    this.temperatureHumidity.RequestMeasurement();
  this.temperatureCheckTimer.Start();
    this.temperatureHumidity.MeasurementComplete += this.
TemperatureHumidityMeasurementComplete;
}

void InitializeJoystick()
{
  this.joystick.JoystickPressed += this.JoystickPressed;
}

void JoystickPressed(Joystick sender, Joystick.JoystickState state)
{
  this.temperatureCheckTimer.Stop();
  var jStick = this.joystick.GetJoystickPostion();
  if (jStick.Y < .3 || jStick.X < .3)
  {
    settings.TargetTemperature -= .5;
    StoreSettings(settings);
  }
  else if (jStick.Y > .7 || jStick.X > .7)
  {
    settings.TargetTemperature += .5;
    StoreSettings(settings);
  }
  this.RedrawDisplay();
  this.temperatureHumidity.RequestMeasurement();
  this.temperatureCheckTimer.Start();
}


void TemperatureHumidityMeasurementComplete(TemperatureHumidity sender,
double temperature, double relativeHumidity)
{
  var targetTemp = settings.TargetTemperature;
  this.lastTemperatureReading = temperature;
  this.lastHumidityReading = relativeHumidity;
  this.relays.Relay1 = (lastTemperatureReading > targetTemp);
  this.RedrawDisplay();
}
```

already guessed, the handling of certificates represents a bit of a hurdle for these devices, and the cryptography requirements for SSL/TLS are so significant in terms of computation effort, memory consumption and code footprint that not all devices can support them. However, because security is clearly becoming increasingly important, even in this space as devices need to communicate across the Internet, version 4.2 of the .NET Micro Framework brings significant improvements for SSL/TLS support for devices having sufficient resources to handle it. I'll discuss this subject in more depth a bit later.

Thermostat Functionality

Implementing local thermostat functionality for this sample is pretty straightforward. I'll check temperature and humidity on a schedule using the sensor and switch the fan connected via one of the relay ports off or on when the temperature drops below or rises above a certain threshold. The current status is displayed on the OLED screen and the joystick allows adjusting the target temperature manually.

As I start the device, I'll wire up events to a timer to trigger the temperature readings and to read events from the joystick. When the joystick is pressed, I'll suspend the timer, check the target temperature according to the joystick position, immediately request a new temperature reading from the sensor and resume the timer. When a temperature reading finishes, the TemperatureHumidityMeasurementComplete event gets raised by the sensor. I'll then store the current readings and adjust the state of the relay to switch the fan if necessary. That's the extent of the thermostat logic, which is shown in part in **Figure 2**.

Figure 3 **Updating Stored Data**

```
static ApplicationSettings GetSettings()
{
  var data = ExtendedWeakReference.RecoverOrCreate(
    typeof(ApplicationSettings),
    0,
    ExtendedWeakReference.c_SurviveBoot | ExtendedWeakReference.c_SurvivePowerdown);
  var settings = data.Target as ApplicationSettings;
  if (settings == null)
  {
    data.Target = settings = ApplicationSettings.Defaults;
  }
  return settings;
}

static void StoreSettings(ApplicationSettings settings)
{
  var data = ExtendedWeakReference.RecoverOrCreate(
    typeof(ApplicationSettings),
    0,
    ExtendedWeakReference.c_SurviveBoot | ExtendedWeakReference.c_SurvivePowerdown);
  data.Target = settings;
  data.PushBackIntoRecoverList();
}
```

Whenever I adjust the target temperature in the JoystickPressed method, I store the new value in the Program class' settings field and call StoreSettings. The settings field is of type ApplicationSettings, a serializable class in the device code that holds everything the device needs to remember across resets and power cycles. To persistently store the data, the .NET Micro Framework reserves some storage pages in the device's non-volatile memory and provides access to this storage via the ExtendedWeakReference class. That's probably not intuitive until you recognize that it's primarily a mechanism to swap data out of main memory under pressure, and it conveniently doubles as a storage feature. The class holds weak references to objects, just as the regular WeakReference does in the .NET Framework, but will swap the data to non-volatile storage instead of discarding it once the garbage collector comes around. Because the data gets swapped out of main memory, it needs to be serialized for storage, which explains why the ApplicationSettings class (which you'll see being used later when we discuss provisioning) needs to be serializable.

Recovering an object from its storage location or creating a new storage slot with the RecoverOrCreate method requires specifying a unique identifier. I only have one object to store, so I'll use a fixed identifier (zero). After the object has been stored and recovered once, any updates need to be forced back into storage using the PushBackIntoRecoveryList method on the ExtendedWeakReference instance, so that's what I do in StoreSettings to flush changes out, as shown in **Figure 3**.

Provisioning

In the beginning, the device is in "factory new" state—the device code has been deployed but the device hasn't yet been initialized

and therefore doesn't have any current settings. You can see this state reflected in the GetSettings method when the settings object is still null and is therefore initialized with default settings.

Because I want to let the device communicate with and through an Internet infrastructure—Windows Azure Service Bus—I need to equip the device with a set of credentials to talk to that infrastructure

Figure 4 **The Provisioning Service**

```
namespace BackendWebRole
{
  using System;
  using System.Configuration;
  using System.Linq;
  using System.Net;
  using System.ServiceModel;
  using System.ServiceModel.Web;
  using Microsoft.ServiceBus;
  using Microsoft.ServiceBus.AccessControlExtensions;
  using Microsoft.ServiceBus.Messaging;

  [ServiceContract(Namespace = "")]
  public class ProvisioningService
  {
    const string DevicesTopicPath = "devices";
    const string EventsTopicPath = "events";
    static readonly AccessControlSettings AccessControlSettings;
    static readonly string ManagementKey;
    static readonly string NamespaceName;
    static Random rnd = new Random();

      static ProvisioningService()
      {
        NamespaceName = ConfigurationManager.AppSettings["serviceBusNamespace"];
        ManagementKey = ConfigurationManager.AppSettings["managementKey"];
        AccessControlSettings = new AccessControlSettings(
          NamespaceName, ManagementKey);
      }

      [OperationContract, WebInvoke(Method = "POST", UriTemplate = "/setup")]
      public void SetupDevice()
      {
        var rcx = WebOperationContext.Current.OutgoingResponse;
        var qcx = WebOperationContext.Current.IncomingRequest;
        var id = qcx.Headers["P-DeviceId"];
        if (this.CheckAllowList(id))
        {
          try
          {
            var deviceConfig = new DeviceConfig();
            CreateServiceIdentity(ref deviceConfig);
            CreateAndSecureEntities(ref deviceConfig);

            rcx.Headers["P-DeviceAccount"] = deviceConfig.DeviceAccount;
            rcx.Headers["P-DeviceKey"] = deviceConfig.DeviceKey;
            rcx.Headers["P-DeviceSubscriptionUri"] =
              deviceConfig.DeviceSubscriptionUri;
            rcx.Headers["P-EventSubmissionUri"] = deviceConfig.EventSubmissionUri;
            rcx.StatusCode = HttpStatusCode.OK;
            rcx.SuppressEntityBody = true;
          }
          catch (Exception)
          {
            rcx.StatusCode = HttpStatusCode.InternalServerError;
            rcx.SuppressEntityBody = true;
          }
        }
        else
        {
          rcx.StatusCode = HttpStatusCode.Forbidden;
          rcx.SuppressEntityBody = true;
        }
      }

      static void CreateAndSecureEntities(ref DeviceConfig deviceConfig)
      {
        var namespaceUri = ServiceBusEnvironment.CreateServiceUri(
          Uri.UriSchemeHttps, NamespaceName, string.Empty);
        var nsMgr = new NamespaceManager(namespaceUri,
          TokenProvider.CreateSharedSecretTokenProvider("owner", ManagementKey));
        var ruleDescription = new SqlFilter(
          string.Format("DeviceId='{0}' OR Broadcast=true",
            deviceConfig.DeviceAccount));
        var subscription = nsMgr.CreateSubscription(
          DevicesTopicPath, deviceConfig.DeviceAccount, ruleDescription);

        deviceConfig.EventSubmissionUri = new Uri(
          namespaceUri, EventsTopicPath).AbsoluteUri;
        deviceConfig.DeviceSubscriptionUri =
          new Uri(namespaceUri,
            SubscriptionClient.FormatSubscriptionPath(
              subscription.TopicPath,
              subscription.Name)).AbsoluteUri;
        GrantSendOnEventTopic(deviceConfig);
        GrantListenOnDeviceSubscription(deviceConfig);
      }

      static void GrantSendOnEventTopic(DeviceConfig deviceConfig)
      {
        var settings = new AccessControlSettings(NamespaceName, ManagementKey);
        var topicUri = ServiceBusEnvironment.CreateServiceUri(
          Uri.UriSchemeHttp, NamespaceName, EventsTopicPath);
        var list = NamespaceAccessControl.GetAccessControlList(topicUri, settings);
        var identityReference =
          IdentityReference.CreateServiceIdentityReference(
            deviceConfig.DeviceAccount);
        var existing = list.FirstOrDefault((r) =>
          r.Condition.Equals(identityReference) &&
          r.Right.Equals(ServiceBusRight.Send));
        if (existing == null)
        {
          list.AddRule(identityReference, ServiceBusRight.Send);
          list.SaveChanges();
        }
      }

      static void GrantListenOnDeviceSubscription(DeviceConfig deviceConfig)
      {
        var settings = new AccessControlSettings(NamespaceName, ManagementKey);
        var subscriptionUri = ServiceBusEnvironment.CreateServiceUri(
          Uri.UriSchemeHttp,
          NamespaceName,
          SubscriptionClient.FormatSubscriptionPath(
            DevicesTopicPath, deviceConfig.DeviceAccount));
        var list = NamespaceAccessControl.GetAccessControlList(
          subscriptionUri, settings);
        var identityReference = IdentityReference.CreateServiceIdentityReference(
          deviceConfig.DeviceAccount);
        var existing = list.FirstOrDefault((r) =>
          r.Condition.Equals(identityReference) &&
          r.Right.Equals(ServiceBusRight.Listen));
        if (existing == null)
        {
          list.AddRule(identityReference, ServiceBusRight.Listen);
          list.SaveChanges();
        }
      }

      static void CreateServiceIdentity(ref DeviceConfig deviceConfig)
      {
        var name = Guid.NewGuid().ToString("N");
        var identity =
          AccessControlServiceIdentity.Create(AccessControlSettings, name);
        identity.Save();

        deviceConfig.DeviceAccount = identity.Name;
        deviceConfig.DeviceKey = identity.GetKeyAsBase64();
      }
        bool CheckAllowList(string id)
      {
        return true;
      }
    }
  }
```

Figure 5 **Configuring the Device**

```
bool PerformProvisioning()
{
  [ ... display status ... ]

  try
  {
    var wr = WebRequest.Create(
      "http://cvdevices.cloudapp.net/Provisioning.svc/setup");
    wr.Method = "POST";
    wr.ContentLength = 0;
    wr.Headers.Add("P-DeviceId", this.deviceId);

    using (var wq = (HttpWebResponse)wr.GetResponse())
    {
      if (wq.StatusCode == HttpStatusCode.OK)
      {
        settings.DeviceAccount = wq.Headers["P-DeviceAccount"];
        settings.DeviceKey = wq.Headers["P-DeviceKey"];
        settings.DeviceSubscriptionUri = new Uri(
          wq.Headers["P-DeviceSubscriptionUri"]);
        settings.EventSubmissionUri = new Uri(
          wq.Headers["P-EventSubmissionUri"]);
        settings.NetworkProvisioningCompleted = true;
        StoreSettings(settings);
        return true;
      }
    }
  }
  catch (Exception e)
  {
    return false;
  }
  return false;
}

void NetworkAvailable(Module.NetworkModule sender,
  Module.NetworkModule.NetworkState state)
{
  ConvertBase64.ToBase64String(ethernet.NetworkSettings.PhysicalAddress);
  if (state == Module.NetworkModule.NetworkState.Up)
  {
    try
    {
      Utility.SetLocalTime(NtpClient.GetNetworkTime());
    }
    catch
    {
      // Swallow any timer exceptions
    }

    if (!settings.NetworkProvisioningCompleted)
    {
      if (!this.PerformProvisioning())
      {
        return;
      }
    }
    if (settings.NetworkProvisioningCompleted)
    {
      this.tokenProvider = new TokenProvider(
        settings.DeviceAccount, settings.DeviceKey);
      this.messagingClient = new MessagingClient(
        settings.EventSubmissionUri, tokenProvider);
    }
  }
}
```

and also tell it which resources to talk to. That first step of setting up a factory new device with the required network configuration and setting up the matching resources on the server side is known as *provisioning*; I discussed the basic architectural model for it in the previous article.

In the device code I'll be fairly strict about getting the device provisioned properly and will initiate the provisioning steps every time the device connects to the network and doesn't have a valid configuration. For that, I keep a Boolean flag in the settings to tell me whether I've had previous success. If the flag isn't set, I issue the call to the provisioning service hosted in Windows Azure.

The provisioning service is responsible for verifying the identity of the device using its unique device identifier, which was registered on an allow-list maintained by the service when it was produced. Once the device is activated, it gets removed from the allow-list. To keep things reasonably simple for this article, however, I'll skip the implementation of the allow-list management.

Once the device is considered legitimate, the provisioning service, following the model established in the previous article, allocates the device to a particular scale-unit and to a particular fan-out Topic inside of that scale-unit. For this example, I'm going to keep it simple and create a subscription for a single fixed Topic named *devices* that serves as the command channel from the cloud into the device, and for a Topic named *events* to collect information from the devices. In addition to creating the subscription and associating the device with the Topic, I'll also create a service-identity for the device in the Access Control Service (a feature of Windows Azure Active Directory) and grant that identity the necessary rights to send messages into the events Topic and to receive messages from the newly created subscription to the devices Topic. The device can perform exactly those two operations on Windows Azure Service Bus—nothing more.

Figure 4 shows the core of the provisioning service. The service depends on the Windows Azure Service Bus management API (the NamespaceManager) found in the core Microsoft.ServiceBus.dll assembly that ships as part of the Windows Azure SDK or via NuGet. It also relies on a helper library for managing access control accounts and permissions available as part of the Authorization sample for Service Bus and, of course, also included in the downloadable code for this article.

The service consists of a single HTTP resource, named */setup*, implemented using the Windows Communication Foundation (WCF) Web operation SetupDevice, which accepts POST requests. You'll notice that the method is parameterless and also doesn't return an entity payload. That's no accident. Instead of using an HTTP entity-body in XML, JSON or form-encoding to carry request and response information, I'm making it very simple for the device and putting the payloads in custom HTTP headers. This eliminates the need for a specific parser to analyze the payload, and it keeps the code footprint small. The HTTP client already knows how to parse headers, and for what I want to do here, that's plenty.

The matching device code calling the HTTP resource is shown in **Figure 5**, and it's difficult to imagine making that call any simpler than this. The device-identifier is sent in a header and the post-provisioning configuration settings are likewise returned via headers. No juggling of streams, no parsing, just simple key/value pairs the HTTP client readily understands.

If the settings indicate that provisioning is necessary, the PerformProvisioning method is invoked from the NetworkAvailable function, which gets triggered when the network is up and the device is assigned an IP address via DHCP. Once provisioning is complete, the settings are used to configure the token provider and messaging client to talk to Windows Azure Service Bus. You'll also notice an

NTP client call. NTP stands for "network time protocol" and I've borrowed a simple, BSD-licensed NTP client written by Michael Schwarz to enable the sample to get the current time, which you need if you want to check SSL certificate expiration.

As you can see back in **Figure 4**, SetupDevices calls the mock-check on the allow-list CheckAllowList and, if that's successful, then calls CreateServiceIdentity and CreateAndSecureEntities. The CreateServiceIdentity method creates a new service identity along with a secret key in the Access Control namespace associated with the Windows Azure Service Bus namespace configured for the application. The CreateAndSecureEntities method creates a new subscription for the entity on the devices Topic, configuring the subscription with a SQL rule that allows sending messages into the Topic to target either the specific subscription by including a DeviceId property set to the device-account name, or all subscriptions by including a Broadcast property with a Boolean value of true. After the subscription has been created, the method calls the GrantSendOnEventTopic and GrantListenOnDeviceSubscription methods that grant the required permissions on the entities to the new service identity using the Access Control library.

Once all that has been successfully completed, the results of the provisioning operations are mapped to headers in the HTTP response for the request and returned with an OK status code, and the device stores the results in non-volatile memory and sets the flag for NetworkProvisioningCompleted.

Sending Events and Receiving Commands

With provisioning completed, the device is now ready to send events to the Windows Azure Service Bus events Topic and to receive commands from its subscription to the devices Topic. But before I go there, I have to discuss a sensitive issue: security.

As I mentioned earlier, SSL/TLS is an expensive protocol suite for small devices. That is to say, some devices won't ever be able to support SSL/TLS, or they might support it only in a limited fashion because of compute capacity or memory constraints. As a matter of fact, though at the time of this writing the GHI Electronics FEZ Spider mainboard based on the .NET Micro Framework 4.1 I'm using here can nominally speak SSL/TLS and therefore HTTPS, its SSL/TLS firmware apparently can't deal with the certificate chain presented to it by Windows Azure Service Bus or the Access Control service. As the firmware for these devices gets updated to the new 4.2 version of the .NET Micro Framework, these limitations will go away for this particular device, but the issue that some devices are simply too constrained to deal with SSL/TLS remains true in principle, and there's active discussion in the embedded device community on appropriate protocol choices that aren't quite as heavyweight.

Thus, even though the device now has a proper account, it can't get a token from the Access Control service because using HTTPS is a prerequisite for doing so. The same is true for sending a message into Windows Azure Service Bus, which mandates HTTPS for all requests that require passing an access token, including all interactions with Queues and Topics. Moreover, if this sample were production code, I would, of course, have to expose the provisioning endpoint via HTTPS to protect the secret key as it's returned to the device.

What now? Well, "what now" is ultimately about making the right trade-offs, and this definitely includes money—in manufacturing, price differences of a few cents add up when millions of a particular kind of device are being made. If the device isn't capable of handling a required security protocol, the questions are how much harm could be caused by not having that protocol and how to close the gap between the device's lack of required features and the infrastructure's demands.

What should be clear is that any data stream that isn't encrypted and signed is susceptible to eavesdropping and manipulation. When a device reports only sensor data, it's worth considering whether a man-in-the-middle manipulation on such a network path is conceivably valuable for anyone or could be analytically detected. The result might sometimes be that sending the data in the clear is OK. Command and control paths are a different matter; as soon as the behavior of a device can be remotely controlled over a network, I can't think of a case where I wouldn't want to have that communication path protected at least with a signature for integrity. The value of privacy for command and control depends on the use case. If there's mitigation against manipulation in place, setting the thermostat target temperature doesn't seem to be worthy of lot of encryption effort.

The sample code that goes with this article includes two variations of the communication flow from the device to the cloud. The first is a much-simplified Windows Azure Service Bus API that requires HTTPS and does the regular handshake of acquiring an Access Control token and talking to Windows Azure Service Bus directly.

The second path uses the same general shape of Windows Azure Service Bus HTTP protocol to send and receive messages, but it creates an HMACSHA256 signature over the message using the secret key it holds. This won't protect the message from eavesdropping, but it does protect the message from manipulation and allows detecting replay attacks when including a unique message-id. The replies will be signed with the same key. Because Service Bus doesn't yet support this authentication model—even though this article is a good indicator that Microsoft is actively thinking about this problem—the path uses a custom gateway service hosted alongside the provisioning service. The custom gateway checks and strips the signature and passes the remaining message on to Windows Azure Service Bus. Using a custom gateway for protocol translation is also generally the right model if you need the cloud system to speak one of the myriad proprietary device protocols. But one thing to keep in mind about the custom gateway approach is that it needs to scale up to the number of devices that concurrently send messages, so making the gateway layer very thin and stateless is a good idea.

The distinction between the two paths is ultimately made by the provisioning service that either dispenses HTTP or HTTPS URIs. In the device code, the difference is handled by the TokenProvider. In the HTTPS case, the devices talk straight to Windows Azure Service Bus, whereas in the HTTP case, the provisioning service talks to the custom gateway. The assumption here is that for the HTTP case, the devices get preprovisioned without exposing the secret key on an unprotected Internet communication path. In other words, the provisioning service runs in the factory, not in Windows Azure.

The device code has two interactions with Windows Azure Service Bus: sending events and receiving commands. I'm going

Figure 6 **Sending Events and Receiving Commands**

```
void TemperatureHumidityMeasurementComplete(TemperatureHumidity sender,
  double temperature, double relativeHumidity)
{
  [...] (see Figure 2)

  if (settings.NetworkProvisioningCompleted &&
    DateTime.UtcNow - settings.LastServerUpdate >
      TimeSpan.FromTicks(TimeSpan.TicksPerMinute))
  {
    settings.LastServerUpdate = DateTime.UtcNow;
    SendEvent(this.lastTemperatureReading, this.lastHumidityReading);
    ProcessCommands();
  }
}

void SendEvent(double d, double lastHumidityReading1)
{
  try
  {
    messagingClient.Send(new SimpleMessage()
      {
        Properties = {
          {"Temperature",d},
          {"Humidity", lastHumidityReading1},
          {"DeviceId", settings.DeviceAccount}
        }
      });
  }
  catch (Exception e)
  {
    Debug.Print(ethernet.ToString());
  }
}
```

```
void ProcessCommands()
{
  SimpleMessage cmd = null;
  try
  {
    do
    {
      cmd = messagingClient.Receive(TimeSpan.Zero, ReceiveMode.ReceiveAndDelete);
      if (cmd != null && cmd.Properties.Contains("Command"))
      {
        var commandType = (string)cmd.Properties["Command"];
        switch (commandType)
        {
          case "SetTemperature":
            if (cmd.Properties.Contains("Parameter"))
            {
              this.settings.TargetTemperature =
                double.Parse((string)cmd.Properties["Parameter"]);
              this.RedrawDisplay();
              this.temperatureHumidity.RequestMeasurement();
              StoreSettings(this.settings);
            }
            break;
        }
      }
    }
    while (cmd != null);
  }
  catch (Exception e)
  {
    Debug.Print(e.ToString());
  }
}
```

to send out an event once every minute after a new temperature reading has been made, and I'll also use that opportunity to grab any pending commands and execute them. To do that I'll amend the TemperatureHumidityMeasurementComplete method shown in **Figure 2**, and add calls to SendEvent and ProcessCommands to be processed once every minute, as shown in **Figure 6**.

The SendEvent method uses the messaging client, which gets initialized once network connectivity is available. The messaging client is a tiny version of the Windows Azure Service Bus API that's capable of sending and receiving messages to and from Service Bus Queues, Topics and Subscriptions. The ProcessCommands method uses the same client to fetch commands from the device's subscription and processes them. For now, the device only understands the SetTemperature command with a Parameter indicating the absolute temperature to set as a numeric value (in Celsius, by the way). Mind that ProcessCommands specifies a TimeSpan.Zero timeout for receiving messages from the Windows Azure Service Bus subscription, indicating that it's not willing to wait for messages to arrive. I want to grab a message only if there's one available and otherwise immediately back off. That reduces the traffic for the custom gateway (should I use HTTP and have one in place) and doesn't require me to keep a receive loop open on the device. The trade-off is latency. In the worst case, commands have a latency of one minute. If that's a problem, you can use a longer timeout that causes long polling (which the library supports) and, with that, drive down command latency to a few milliseconds.

The matching server side, for receiving events and sending commands to all registered devices by dropping messages into the Topic, simply follows the regular rules of the Windows Azure Service Bus API and is part of the sample code you can download, so I'll omit that code here.

Wrapping Up

The goal of this series on the Internet of Things is to provide some insight into the kinds of technologies we're working on here at Microsoft to enable connected-device prototyping and development. We also want to show how cloud technologies such as Windows Azure Service Bus and analytics technologies such as StreamInsight can help you manage the data flow from and to connected devices; how you can create large-scale cloud architectures for handling a great many devices; and how to aggregate and digest information from them.

On the way, I built an embedded device that you can place on any home network and control remotely from any other network, which is pretty cool if you ask me.

I believe we're in the very early stages of this journey. In talking to Microsoft customers from all over, I've seen that a huge wave of connected and custom-built devices is on the way, and this is a great opportunity for .NET developers and innovative companies looking to build cloud offerings to connect to these devices and to combine them with other connected assets in inventive ways. Let's see what you can do. ■

CLEMENS VASTERS *is the principal technical lead on the Windows Azure Service Bus team. Vasters has been on the team from the earliest incubation stages and works on the technical feature roadmap for Windows Azure Service Bus, which includes push notifications and high-scale signaling for Web and devices. He's also a frequent conference speaker and architecture courseware author. Follow him on Twitter at twitter.com/clemensv.*

THANKS to the following technical experts for reviewing this article:
Elio Damaggio, Todd Holmquist-Sutherland, Abhishek Lal, Zach Libby, Colin Miller and Lorenzo Tessiore

JAMES MCCAFFREY

Classification and Prediction Using Neural Networks

In this month's column, I explain how to use neural networks to solve classification and prediction problems. The goal of classification is best explained by example. Suppose you have some historical data about iris flowers that resembles:

```
5.1 3.5 1.4 0.2 Setosa
7.0 3.2 4.7 1.4 Versicolor
6.3 3.3 6.0 2.5 Virginica
6.4 3.2 4.5 1.5 Versicolor
...
```

Each line of space-delimited data has five fields. The first four numeric fields are sepal (the green bud covering) length, sepal width, petal (the colored part of the flower) length and petal width. The fifth field is the species: Setosa, Versicolor or Virginica. The goal of classification is to determine an equation or set of rules that predicts which species or class an iris belongs to. Then the rule set can be used to predict the class of a new iris based on its values for sepal and petal length and width. This iris plant data is a classic example first used by R. A. Fisher in 1936. It may not excite you, but classification is extremely important. Examples include classifying an applicant's credit rating based on variables such as income and monthly expenses (or equivalently, predicting their credit worthiness), and classifying whether a hospital patient has cancer based on the values from a blood test.

> One way to think about neural networks is that they are virtual input-output devices.

There are many approaches for classifying data, including the use of neural networks. One way to think about neural networks is that they are virtual input-output devices that accept any number of numeric inputs and produce any number of numeric outputs. The best way for you to see where I'm headed is to examine the screenshot in **Figure 1** and the image in **Figure 2**. **Figure 1** shows neural network classification in action. To keep the concepts of classification using neural networks clear, I didn't use real-life data. Instead I used artificial data where the input x-values are four arbitrary numeric values with no particular meaning. The output y-variable to classify is color, and it can take one of three categorical values: red, green or blue. The program shown in **Figure 1** begins by generating a text file with 100 lines of artificial data (for example, "8.0 5.0 9.0 5.0 green"), and then displays the first four lines of that data. Next, the program reads the raw data file into memory as a training matrix with 80 rows of data and a testing matrix with 20 rows. Notice that two transformations are applied to the raw data. The raw numeric input data is normalized so that all values are between -1.00 and +1.00, and the raw output data (such as "red") is encoded into a vector with three values ("1.0 0.0 0.0").

After creating the training and testing matrices, the demo program creates a fully connected feed-forward neural network with three input neurons, five hidden neurons for computation and three output neurons. It turns out that a 4-5-3 fully connected neural network requires 43 weights and biases. Next, the classification program analyzes the training data to find the best 43 weights and biases (those that minimize the total classification error). The program uses particle swarm optimization along with cross-entropy error to estimate the best values of the weights and biases.

The classification program then loads the best weights and biases into the neural network and evaluates the predictive accuracy of the model on the 20 rows of data in the test matrix. Notice the output of the neural network has been designed so that the three output values sum to 1.0. In this example, the model correctly predicts 17 of the 20 test vectors. The image in **Figure 2** illustrates the neural network accepting input of (-1.00, 1.00, 0.25, -0.50) and generating a predicted output of (0.9, 0.1, 0.0), which corresponds to red.

The example program points out that there are five main decisions to make when using neural networks for classification where the input data is numeric and the output data is categorical:

1. How to normalize numeric input data
2. How to encode categorical output data
3. How to generate neural output in the range [0.0, 1.0]
4. How to measure error when training
5. How to measure accuracy when testing

In the sections that follow I'll explain that the choices made in the example program are:

1. Perform a linear transformation on numeric input data
2. Use 1-of-N encoding for categorical output data
3. Use a Softmax activation function for the output layer
4. Use cross-entropy error to determine the best weights
5. Use a winner-takes-all approach to determine accuracy

Code download available at code.msdn.microsoft.com/mag201207TestRun.

The program code that generated the screenshot in **Figure 1** is a bit too long to present in this article, so I focus on the algorithms used instead. The complete program source is available from the MSDN code download site at code.msdn.microsoft.com/mag201207TestRun. This article assumes you have advanced programming skills and a basic understanding of neural networks. I explain neural network basics in the May 2012 issue of *MSDN Magazine* (msdn.microsoft.com/magazine/hh975375).

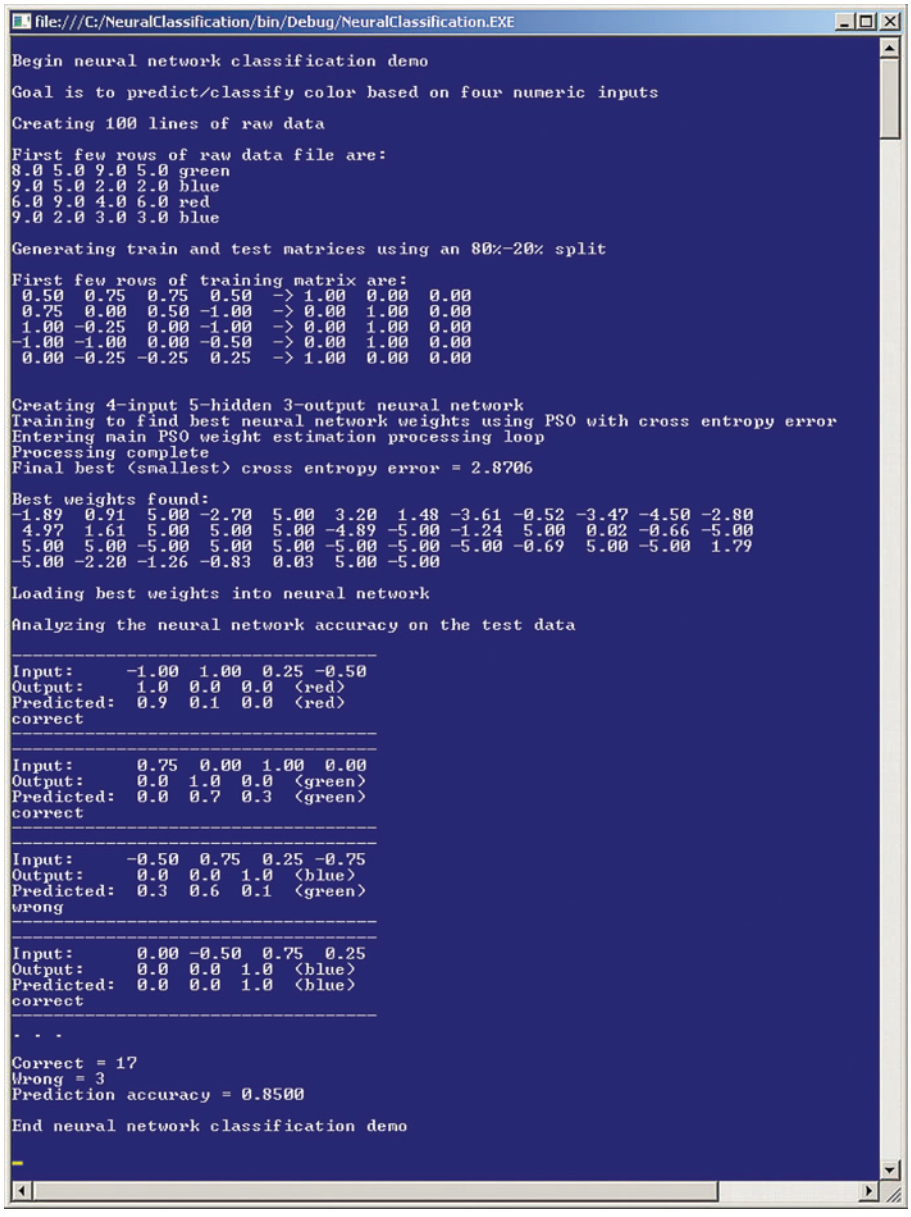


Figure 1 **Neural Network Classification in Action**

Overall Program Structure

Figure 3 lists the program structure of the example shown running in **Figure 1**. I used Visual Studio 2010 to create a single C# console application named NeuralClassification. In the Solution Explorer window, I renamed file Program.cs to the more descriptive NeuralClassificationProgram.cs, which automatically renamed the class containing Main. I removed unneeded using statements that were generated by the Visual Studio template and added a reference to the System.IO namespace.

In addition to the class that contains the Main method, the program has three other classes. Class NeuralNetwork encapsulates a fully connected feed-forward neural network. All of the core program logic is contained in this class. Class Helpers contains six utility routines. Class Particle defines a particle object that's used by the particle swarm optimization algorithm in the Train method of the NeuralNetwork class. One characteristic of classification programs is that there are many possible program structures; the organization presented here is just one possibility.

Generating the Raw Data File

In most classification scenarios you will already have a set of raw data, but for this article I created dummy raw data using method MakeData. Here's the process in pseudo-code:

```
create 43 arbitrary weights between -2.0 and +2.0
create a 4-5-3 neural network
load weights into the neural network
open a result file for writing
loop 100 times
  generate four random inputs x0, x1, x2, x3 between 1.0 and 9.0
  compute the y0, y1, y2 neural outputs for the input values
  determine largest of y0, y1, y2
  if y0 is largest write x0, x1, x2, x3, red
  else if y1 is largest write x0, x1, x2, x3, green
  else if y2 is largest write x0, x1, x2, x3, blue
end loop
close result file
```

The goal here is to get some data that is definitely classifiable with 100 percent accuracy, rather than random data where it's not clear how effective any classification approach would be. In other words, I use a neural network to create raw data and then I start over and use a neural network to attempt to classify that data.

Creating the Training and Testing Matrices

When performing classification analysis with a set of existing data, one common approach, called holdout validation, is to split the data into a larger data set (often 80 percent) for training the neural network and a smaller data set (20 percent) for testing the model. Training means finding the neural network weights and biases that minimize some error value. Testing means evaluating the neural network with the best weights found during training, using some measure of accuracy. Here, method MakeTrainAndTest creates

training and testing matrices and also normalizes the numeric input data and encodes the categorical output data. In pseudo-code, the method works like this:

```
determine how many rows to split data into
create a matrix to hold all data
loop
  read a line of data
  parse each field
  foreach numeric input
    normalize input to between -1.0 and +1.0
  end foreach
  encode categorical output using 1-of-N
  place normalized and encoded data in matrix
end loop
shuffle matrix
create train and test matrices
transfer first 80% rows of matrix to train, remaining rows to test
```

The method signature is:

```
static void MakeTrainAndTest(string file, out double[][] trainMatrix,
  out double[][] testMatrix)
```

The parameter called file is the name of the raw data file to create. Parameters trainMatrix and testMatrix are output parameters where the results are placed. The method begins with:

```
int numLines = 0;
FileStream ifs = new FileStream(file, FileMode.Open);
StreamReader sr = new StreamReader(ifs);
while (sr.ReadLine() != null)
  ++numLines;
sr.Close(); ifs.Close();
int numTrain = (int)(0.80 * numLines);
int numTest = numLines - numTrain;
```

This code counts the number of lines in the raw data file and then computes how many lines constitute 80 percent and 20 percent of the data. Here, the percentages are hard-coded; you may want to parameterize them. Next, a matrix that will hold all data is allocated:

```
double[][] allData = new double[numLines][];
  for (int i = 0; i < allData.Length; ++i)
    allData[i] = new double[7];
```

> An important design alternative to the train-test approach is to split raw data into three sets: train, validate and test.

There are seven columns: one column for each of the four numeric inputs and three columns for the 1-of-N encoded value of the categorical color variable. Recall that for this example, the goal is to predict color, which can be one of three categorical values: red, green or blue. Encoding using the 1-of-N technique in this situation means encoding red as (1.0, 0.0, 0.0), green as (0.0, 1.0, 0.0) and blue as (0.0, 0.0, 1.0). Categorical data must be encoded numerically because neural networks deal directly only with numeric values. It turns out that encoding color using a simple approach such as 1 for red, 2 for green and 3 for blue is a bad idea. The explanation of why this is bad is a bit lengthy and outside the scope of this article.

An exception to the 1-of-N encoding guideline for categorical output data is that when there are only two possible values, such as "male" or "female," you can use 1-of-(N-1) encoding where you have a single numeric output value so that, for example, 0.0 means male and 1.0 means female.

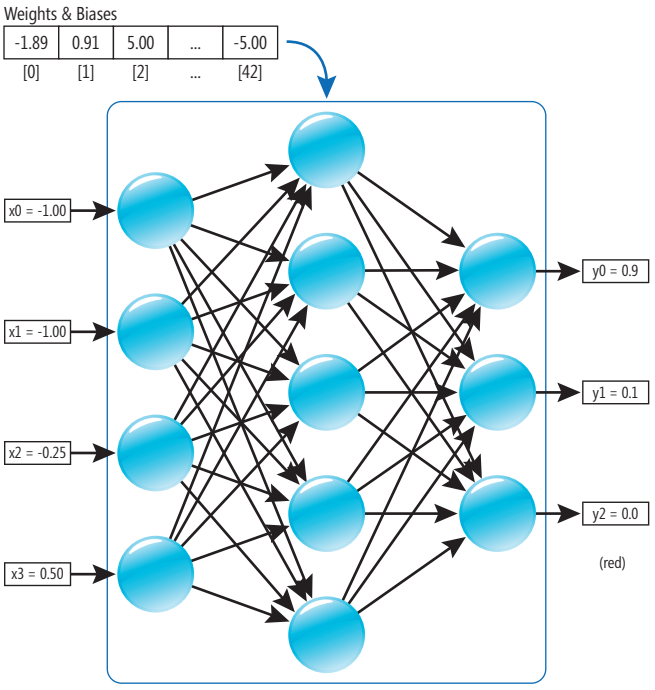


Figure 2 **Neural Network Structure**

The encoding is performed by this code:

```
tokens = line.Split(' ');
allData[row][0] = double.Parse(tokens[0]);
allData[row][1] = double.Parse(tokens[1]);
allData[row][2] = double.Parse(tokens[2]);
allData[row][3] = double.Parse(tokens[3]);

for (int i = 0; i < 4; ++i)
  allData[row][i] = 0.25 * allData[row][i] - 1.25;

if (tokens[4] == "red") {
  allData[row][4] = 1.0; allData[row][5] = 0.0; allData[row][6] = 0.0; }
else if (tokens[4] == "green") {
  allData[row][4] = 0.0; allData[row][5] = 1.0; allData[row][6] = 0.0; }
else if (tokens[4] == "blue") {
  allData[row][4] = 0.0; allData[row][5] = 0.0; allData[row][6] = 1.0; }
```

Recall that a line of raw data looks like this:

```
8.0 5.0 9.0 5.0 green
```

The five fields are parsed using String.Split. Experience has shown that in most situations, better results are obtained when numeric input is scaled to values between -1.0 and +1.0. Each of the first four numeric inputs is converted by multiplying by 0.25 and subtracting 1.25. Recall that the numeric inputs in the dummy data file are all between 1.0 and 9.0. In a real classification problem, you would have to scan the raw data and determine the min and max values. We want -1.0 to correspond to 1.0 and +1.0 to correspond to 9.0. Doing a linear transformation means finding the slope (0.25 here) and intercept (-1.25). These values can be computed as:

```
slope = 2 / (max value - min value) = 2 / (9.0 - 1.0) = 0.25
intercept = 1.0 - (slope * max value) = 1 - (0.25 * 9.0) = -1.25
```

There are many alternatives to performing a linear transform on numeric input values, but the approach presented here is simple and a good starting point in most situations.

After the four numeric input values have been transformed, the color value from the raw data file is encoded using 1-of-N encoding.

Figure 3 **Neural Classification Program Structure**

```
using System;
using System.IO;

namespace NeuralClassification
{
  class NeuralClassificationProgram
  {
    static Random rnd = null;
    static void Main(string[] args)
    {
      try
      {
        Console.WriteLine("\nBegin Neural network classification\n");
        rnd = new Random(159); // 159 makes a nice example

        string dataFile = "..\\..\\colors.txt";
        MakeData(dataFile, 100);

        double[][] trainMatrix = null;
        double[][] testMatrix = null;
        MakeTrainAndTest(dataFile, out trainMatrix, out testMatrix);

        NeuralNetwork nn = new NeuralNetwork(4, 5, 3);
        double[] bestWeights = nn.Train(trainMatrix);

        nn.SetWeights(bestWeights);
        double accuracy = nn.Test(testMatrix);

        Console.WriteLine("\nEnd neural network classification\n");
      }
      catch (Exception ex)
      {
        Console.WriteLine("Fatal: " + ex.Message);
      }
    } // Main()

    static void MakeData(string dataFile, int numLines) { ... }
    static void MakeTrainAndTest(string file, out double[][] trainMatrix,
      out double[][] testMatrix) { ... }
  }

  class NeuralNetwork
  {
    // Class member fields here

    public NeuralNetwork(int numInput, int numHidden,
      int numOutput) { ... }
    public void SetWeights(double[] weights) { ... }
    public double[] ComputeOutputs(double[] currInputs) { ... }
    private static double SigmoidFunction(double x) { ... }
    private static double[] Softmax(double[] hoSums) { ... }
    public double[] Train(double[][] trainMatrix) { ... }
    private double CrossEntropy(double[][] trainData,
      double[] weights) { ... }
    public double Test(double[][] testMatrix) { ... }
  }

  public class Helpers
  {
    static Random rnd = new Random(0);
    public static double[][] MakeMatrix(int rows, int cols) { ... }
    public static void ShuffleRows(double[][] matrix) { ... }
    public static int IndexOfLargest(double[] vector) { ... }
    public static void ShowVector(double[] vector, int decimals,
      bool newLine) { ... }
    public static void ShowMatrix(double[][] matrix, int numRows) { ... }
    public static void ShowTextFile(string textFile, int numLines) { ... }
  }

  public class Particle
  {
    // Class member fields here

    public Particle(double[] position, double fitness,
      double[] velocity, double[] bestPosition,
     double bestFitness) { ... }
    public override string ToString() { ... }
  }

} // ns
```

When all values in the raw data file have been computed and placed in the allData matrix, that matrix has its rows rearranged randomly using the ShuffleRows utility method in the Helpers class. After the order of the rows in allData has been shuffled, space for matrices trainMatrix and testMatrix is allocated, then the first numTrain rows of allData are copied into trainMatrix and the remaining numTest rows of allData are copied into testMatrix.

An important design alternative to the train-test approach is to split raw data into three sets: train, validate and test. The idea is to use the train data to determine the best set of neural network weights in conjunction with the validate data, which is used to know when to stop training. There are other approaches as well, collectively called cross-validation techniques.

The Softmax Activation Function

When performing classification using a neural network where the output variable is categorical, there's a rather tricky relationship among the neural network output activation function, computing error during training and computing the predictive accuracy of the neural network. When categorical output data (such as color with values red, green or blue) is encoded using 1-of-N encoding—for example, (1.0 0.0 0.0) for red—you want the neural network to emit three numeric values so you can determine an error when training the network. You don't, however, want three arbitrary numeric values to be emitted, because then it's not entirely clear how to compute an error term. But suppose the neural network emits three numeric values that are all between 0.0 and 1.0 and that sum to 1.0. Then the emitted values can be interpreted as probabilities, which, as it turns out, makes it easy to compute an error term when training and to compute accuracy when testing. The Softmax activation function emits output values in this form. The Softmax function accepts neural hidden-to-output sums and returns the final neural output values; it can be implemented like this:

```
private static double[] Softmax(double[] hoSums)
{
  double max = hoSums[0];
  for (int i = 0; i < hoSums.Length; ++i)
    if (hoSums[i] > max) max = hoSums[i];
  double scale = 0.0;
    for (int i = 0; i < hoSums.Length; ++i)
      scale += Math.Exp(hoSums[i] - max);
  double[] result = new double[hoSums.Length];
    for (int i = 0; i < hoSums.Length; ++i)
      result[i] = Math.Exp(hoSums[i] - max) / scale;
  return result;
}
```

In principle, the Softmax function computes a scaling factor by taking Exp of each hidden-to-output sum, summing them and then dividing the Exp of each value by the scaling factor. For example, suppose three hidden-to-output sums are (2.0, -1.0, 4.0). The scaling factor would be Exp(2.0) + Exp(-1.0) + Exp(4.0) = 7.39 + 0.37 + 54.60 = 62.36. Then the Softmax output values would be Exp(2.0)/62.36, Exp(-1.0)/62.36, Exp(4.0)/62.36) = (0.12, 0.01, 0.87). Notice that the final outputs are all between 0.0 and 1.0 and do in fact sum to 1.0, and further that the largest hidden-to-output sum (4.0) has the largest output/probability (0.87), and the relationship is similar for the second- and third-largest values.

Unfortunately, a naive implementation of Softmax can often fail because the Exp function becomes very large very quickly and can

produce arithmetic overflow. The preceding implementation uses the fact that Exp(a - b) = Exp(a) / Exp(b) to compute output in a way that avoids overflow. If you trace the execution of the implementation using (2.0, -1.0, 4.0) as inputs, you'll get the same (0.12, 0.01, 0.87) outputs as explained in the preceding section.

Cross-Entropy Error

The essence of training a neural network is to find the set of weights that produces the smallest error for the data in the training set. For example, suppose a row of normalized, encoded training data is (0.75 -0.25 0.25 -0.50 0.00 1.00 0.00). Recall that the first four values are the normalized inputs and the last three values represent green in 1-of-N encoding. Now suppose the inputs are fed through the neural network, which has been loaded with some set of weights, and the Softmax output is (0.20 0.70 0.10). A traditional approach to computing error for this test vector is to use sum of squared differences, which in this case would be (0.00 - 0.20)^2 + (1.00 - 0.70)^2 + (0.00 - 0.10)^2 = 0.14. But suppose the Softmax output was (0.30 0.70 0.00). Both this vector and the previous vector predict that the output is green because the probability of green is 0.70. However, the sum of squared deviations for this second vector is 0.18, which is different from the first error term. Although sum of squared deviations can be used to compute training error, some research results suggest that using an alternative measure called cross-entropy error is preferable.

In principle, the cross-entropy error for a given neural network output vector v and a test output vector t is computed by determining the negative sum of the product of each v vector component and the corresponding t vector component. As usual, this is best understood by example. If a test vector is (0.75 -0.25 0.25 -0.50 0.00 1.00 0.00) and the corresponding Softmax neural network output is (0.20 0.70 0.10), then the cross-entropy error is -1 * (0.00 * Log(0.20)) + (1.00 * Log(0.70)) + (0.00 * Log(0.10)) = -1 * (0 + -0.15 + 0) = 0.15. Notice that all but one of the terms in the sum will always be zero when 1-of-N encoding is used. The cross-entropy error for the entire training set can be computed as the sum of the cross-entropy of all test vectors, or the average cross-entropy of each test vector. An implementation of cross-entropy error is listed in **Figure 4**.

Figure 4 **Cross-Entropy Error**

```
private double CrossEntropy(double[][] trainData, double[] weights)
{
  this.SetWeights(weights);
  double sce = 0.0; // sum of cross entropies
  for (int i = 0; i < trainData.Length; ++i)
  {
    double[] currInputs = new double[4];
    currInputs[0] = trainData[i][0];
    currInputs[1] = trainData[i][1];
    currInputs[2] = trainData[i][2];
    currInputs[3] = trainData[i][3];
    double[] currExpected = new double[3];
    currExpected[0] = trainData[i][4];
    currExpected[1] = trainData[i][5];
    currExpected[2] = trainData[i][6];
    double[] currOutputs = this.ComputeOutputs(currInputs);
    double currSum = 0.0;
    for (int j = 0; j < currOutputs.Length; ++j)
    {
      if (currExpected[j] != 0.0)
        currSum += currExpected[j] * Math.Log(currOutputs[j]);
    }
    sce += currSum; // accumulate
  }
  return -sce;
}
```

Training the Neural Network

There are many ways to train a neural network classifier to find the set of weights that best matches the training data (or, equivalently, yields the smallest cross-entropy error). At a high level of abstraction, training a neural network looks like this:

```
create an empty neural network
loop
  generate a candidate set of weights
  load weights into neural network
  foreach training vector
    compute the neural output
    compute cross-entropy error
    accumulate total error
  end foreach
  if current weights are best found so far
    save current weights
  end if
until some stopping condition
return best weights found
```

> The essence of training a neural network is to find the set of weights that produces the smallest error for the data in the training set.

By far, the most common technique used to train neural networks is called back-propagation. This technique is the subject of a large number of research articles—so many, in fact, that if you're new to the field of neural network classification, you could easily be led to believe that back-propagation is the only technique used for training. Estimating the best set of weights for a neural network is a numeric minimization problem. Two common alternatives to using back-propagation are using a real-valued genetic algorithm (also called an evolutionary optimization algorithm), and using particle swarm optimization. Each estimation technique has strengths and weaknesses. The program shown in **Figure 1** uses particle swarm optimization. I describe evolutionary optimization algorithms in the June 2012 issue of *MSDN Magazine* (msdn.microsoft.com/magazine/jj133825) and particle swarm optimization in the August 2011 issue (msdn.microsoft.com/magazine/hh335067).

There are many techniques to determine when to stop training a neural network. Although you could simply let a training algorithm run until the cross-entropy error is very close to zero, indicating a near-perfect fit, the danger is that the resulting weights will over-fit the training data and the weights will create a neural network that classifies poorly for any data that isn't in the training set. Also, training until there is no change in the cross-entropy error can easily lead to model over-fitting. A simple approach, and one used by the example program, is to limit training to a fixed number of

iterations. In most cases, a much better tactic to avoid over-fitting is to split the source data set into train-validate-test sets. Typically these three data sets use 60 percent, 20 percent and 20 percent of the source data, respectively. The technique computes error on the training set as previously described, but after each iteration in the main loop the technique computes the cross-entropy error on the validation data set. When the cross-entropy error on the validation set starts to show a consistent increase in error, there's a good chance the training algorithm has begun to over-fit the data and the training should halt. There are many other stopping techniques possible.

Classification with neural networks is an important, fascinating and complex topic.

Evaluating the Neural Network Classifier

After the neural network classifier has been trained and has produced a set of best weights and biases, the next step is to determine how accurate the resulting model (where model means the neural network with the set of best weights) is on the test data. Although measures such as sum of squared deviations or cross-entropy error can be used, a reasonable measure of accuracy is simply the percentage of correct predictions made by the model. Once again, there are several approaches for measuring accuracy, but a simple technique is to use what's called the winner-takes-all approach. And, as usual, the technique is best explained by example. Suppose a test vector is (-1.00 1.00 0.25 -0.50 1.00 0.00 0.00), as shown in the first set of prediction data in **Figure 1**. With the set of best weights, the neural network generates a predicted Softmax output of (0.9 0.1 0.0). If each output is interpreted as a probability, then the highest probability is 0.9 and the predicted output can be thought of as (1 0 0), so the model makes the correct prediction. Put another way, the winner-takes-all technique determines the neural network output component with the largest value, makes that component a 1 and all other components 0, and compares that result with the actual data in the training vector. In the third set of accuracy analysis data in **Figure 1**, the test vector is (-0.50 0.75 0.25 -0.75 0.0 0.0 1.0). The actual output data is (0.0 0.0 1.0), which corresponds to blue. The predicted output is (0.3 0.6 0.1). The largest component is the 0.6, so the model prediction is (0 1 0), which corresponds to green. The model made an incorrect prediction.

Wrapping Up

Classification with neural networks is an important, fascinating and complex topic. The example presented here should give you a solid basis for experimenting with neural network classification. However, this article describes only one very specific neural network classification scenario—numeric input variables with a categorical output variable—and is just a starting point. Other scenarios require slightly different techniques. For example, if the input data contains a categorical variable, you might expect that it would be encoded using 1-of-N encoding just like a categorical output variable. In this case, however, the input data should be encoded using the 1-of-(N-1) technique. If you want to learn more about classification using neural networks, I recommend the set of seven FAQs on neural networks available at faqs.org. The links to these FAQs tend to move around but you should be able to find them easily with an Internet search. ■

Dr. James McCaffrey *works for Volt Information Sciences Inc., where he manages technical training for software engineers working at the Microsoft Redmond, Wash., campus. He has worked on several Microsoft products including Internet Explorer and MSN Search. McCaffrey is the author of ".NET Test Automation Recipes" (Apress, 2006). He can be reached at jmccaffrey@volt.com or jammc@microsoft.com.*

Thanks to the following Microsoft technical expert for reviewing this article: *Matthew Richardson*

TED NEWARD AND
JOE HUMMEL, PH.D.

The Science of Computers

For many years, I've heard colleagues describe themselves as "computer scientists." I've also heard it said that, "Any field of study that has to put 'science' in the name isn't really a science." Certainly a fairly large number of people in my field have had no formal training in computer science, and another fairly large number look with disdain at the work coming out of universities and collegiate professorial offices—and with some justification. When's the last time a slew of Greek symbols purporting to describe a type system helped you get that line-of-business app up and running?

I know a number of ridiculously smart people, and one of them is Joe Hummel, someone who can legitimately claim the title computer scientist, owing to the Ph.D. behind his name and all. I asked him to join me in writing this column to explore the science of computing. I thought it would be interesting to start with some of the simpler parts of computer science and work through how understanding the theory behind some of those parts can, in fact, make your life as a working programmer just a little bit easier.

For this installment, we tackle the classic "Big O" discussion, which seeks to give us a consistent way to describe a simple concept: How many resources (time, memory, power and so on) will the computer need to execute a given program? This determination is referred to as algorithm analysis, with the goal of placing your chosen algorithm into a particular mathematical category. Why? So you can select the best one for a situation. There are many categories, with each successive category denoting a jump in resource consumption. We're going to focus on execution time, and just the first three categories: *O(1)*, *O(lgN)* and *O(N)*. As is typical, *N* denotes the number of data items processed by the algorithm. For an executive summary of where we're going with all this, see **Figure 1**, which highlights the execution time you can expect for each category as *N* grows.

An *order N* algorithm is written *O(N)*, pronounced "Big O of N" and also known as "linear." Given *N* data items, a *O(N)* algorithm takes on the order of *N* time steps. An *order log N* algorithm is written *O(lgN)*, pronounced "Big O of log N" and referred to as "logarithmic." Given *N* data items, a *O(lgN)* algorithm takes on the order of *log2(N)* time steps (many fewer). Finally, an *order 1* algorithm is written *O(1)*, pronounced "Big O of 1" and referred to as "constant." For *N* data items, a *O(1)* algorithm takes a constant number of time steps (fewer still). Take a moment to revisit **Figure 1**. What else might this graph convey? For example, notice that when *N* doubles, a linear algorithm takes twice as much time.

Code download available at bit.ly/srchlogfiles.

Let's look at an example of how this applies in real life, and where knowing just a little about algorithm analysis can make a huge impact. Suppose you're given a log file to search for matches against a list of suspicious IP addresses. Simple enough:

```
List<string> suspicious = BuildList(); /* step 0 */
foreach (string line in logfile)
{
  string ipaddr = parse(line);          /* step 1 */
  if (suspicious.Contains(ipaddr))      /* step 2 */
    matches++;                          /* step 3 */
}
```

How do we categorize the time complexity of this approach? Let's assume there are *M* lines in the log file and *N* suspicious IP addresses. Building an unsorted list of *N* elements (step 0) will incur a one-time cost of *O(N)*. The loop will perform *M* iterations, so the cost of the loop is *O(M * cost of one iteration)*. Each iteration performs three basic steps:

1. Parses the line
2. Searches the list
3. Increments matches if a match is found

Getting Analytical with It

The intellectual aspect of algorithm analysis is learning what to count and what not to count. For example, while parsing a line—perhaps applying a regular expression or splitting the line into tokens—might be expensive, it takes a constant amount of time *relative to the data*. In other words, the time required to parse a line remains entirely unchanged, whether there are *M* or *2M* lines in

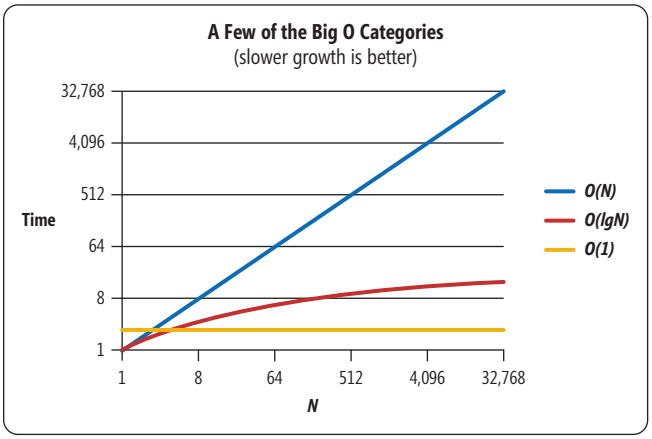


Figure 1 **Execution Times as the Number of Items Processed (*N*) Grows**

Does your Team do more than just track bugs?

Free Trial and Single User FreePack™ available at www.alexcorp.com

Alexsys Team® does! Alexsys Team 2 is a multi-user Team management system that provides a powerful yet easy way to manage all the members of your team and their tasks - including defect tracking. Use Team right out of the box or tailor it to your needs.

Track all your project tasks in one database so you can work together to get projects done.

- Quality Control / Compliance Tracking
- Project Management
- End User Accessible Service Desk Portal
- Bugs and Features
- Action Items
- Sales and Marketing
- Help Desk

Native Smart Card Login Support including Government and DOD

New in Team 2.11

- Full Windows 7 Support
- Windows Single Sign-on
- System Audit Log
- Trend Analysis
- Alternate Display Fields for Data Normalization
- Lookup Table Filters
- XML Export
- Network Optimized for Enterprise Deployment

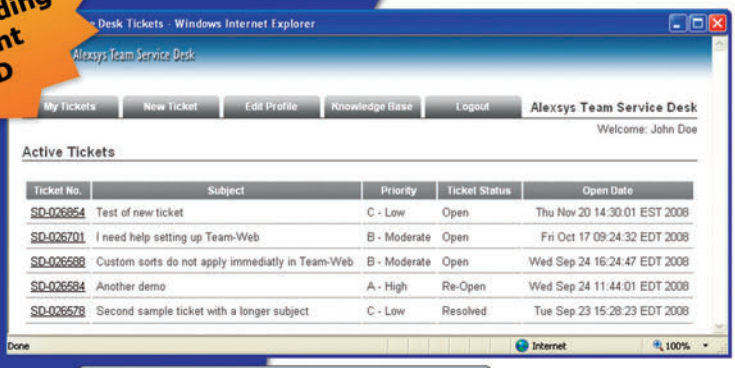
Service Desk Features

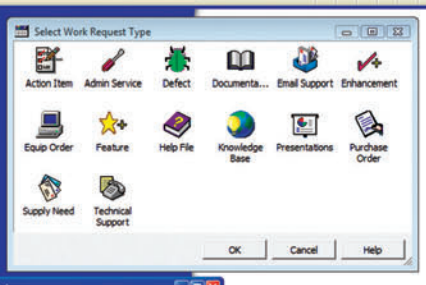
- Fully Secure
- Unlimited Users Self Registered or Active Directory
- Integrated into Your Web Site
- Fast/AJAX Dynamic Content
- Unlimited Service Desks
- Visual Service Desk Builder

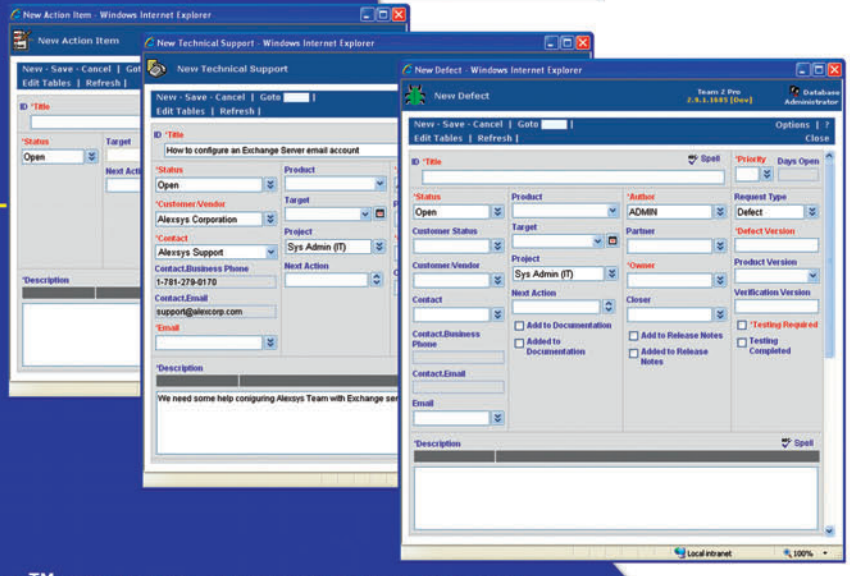
Team 2 Features

- Windows and Web Clients
- Multiple Work Request Forms
- Customizable Database
- Point and Click Workflows
- Role Based Security
- Clear Text Database
- Project Trees
- Time Recording
- Notifications and Escalations
- Outlook Integration

ZDNet Editors' Pick







Free Trial and Single User FreePack™ available at www.alexcorp.com.
FreePack™ includes a free single user Team Pro and Team-Web license.
Need more help? Give us a call at 1-888-880-ALEX (2539).

Team 2 works with its own standard database, while Team Pro works with Microsoft SQL, MySQL, and Oracle Servers.
Team 2 works with Windows 7/2008/2003/Vista/XP.
Team-Web works with Internet Explorer, Firefox, Netscape, Safari, and Chrome.

Figure 2 **Algorithmic Complexities Assuming *M* >> *N***

	Time (t)
Version 1: Using Linear Search	O(M * N)
Version 2: Using Binary Search	O(M * lgN)
Version 3: Using Hashing	O(M)

the file. Nor does it change whether there are *N* or *2N* suspicious IP addresses. Thus, we say steps 1 and 3 take *c* (constant) time. However, step 2, which searches the list, does change relative to the number of suspicious IP addresses:

```
if (suspicious.Contains(ipaddr))
```

The *Contains* method performs a linear search, starting from the first element and searching until either a match is found or the end of the list is reached. (How could we know this? One of three ways: we tested it, we looked at the source code or the MSDN Library documentation tells us.)

With random data, on average we'll search half the list, requiring *N/2* steps. This would hold for most cases, but in this particular case, a stronger argument is that most IP addresses in the log file are trustworthy, because our Web server isn't a bank or a presidential candidate's Web site. This means that rather than there being an even chance that a given client is untrustworthy, the vast majority of searches will exhaust the list without finding a match, requiring *N* steps.

Because constants are dropped when computing the order of things, the search is *O(N)* in either case, yielding a complexity of *O(M * N)* for the loop. In turn, this yields an overall algorithmic complexity of:

= O(build + loop)
= O(N + M * N)
= O((1 + M) * N)
= O(M * N)

Thus, version 1 of our solution requires *O(MN)* time.

'Data, Data, Data! I Cannot Make Bricks Without Clay!'

The first question to ask yourself is: "Do we need a better algorithm?" *M* is typically large, in the millions (possibly billions) of lines. There's not much you can do about that, because you have to touch each line to search it. If *N*, the number of suspicious IP addresses, is small (say, *N < 100*), then your work is done. With today's fast machines, there's little measurable difference between *M* time steps and *100M* time steps. However, as *N* grows larger (for example, *N >> 100*, or *significantly* larger than 100), it's worthwhile to consider other search algorithms.

Unless you have a really big hardware budget, of course. (Sometimes, even then.)

Figure 3 **Predicted Change in Execution Time When *N* Is Doubled**

	Time for 2N
Version 1	O(M * 2 * N) => 2 * O(M * N) => 2t
Version 2	O(M * lg(2N)) => O(M * (lg2 + lgN)) => O(M * (1 + lgN) => O(M + M * lgN) => M + O(M * lgN) => M + t
Version 3	O(M) => t

A more efficient algorithm is *binary search*, which jumps to the middle and then searches left or right depending on whether the element you're looking for is less than the middle element or greater. By cutting the search space in half each time, the algorithm exhibits *O(lgN)* behavior. What does this mean in practice? Given *N*=1,000 elements, at worst binary search takes on the order of 10 time steps ($2^{10} \approx 1{,}000$), while linear search would take 1,000. For *N* = 1 million, the difference is even more dramatic—on the order of 20 time steps versus 1 million. The key trade-off is that binary search requires the data to be in sorted order. Here's the log file search program rewritten to use binary search (changes in bold):

```
List<string> suspicious = BuildList();      /* step 0 */
suspicious.Sort();                          /* step 0.5 */

foreach (string line in logfile)
{
  string ipaddr = parse(line);               /* step 1 */
  if (suspicious.BinarySearch(ipaddr) >= 0)  /* step 2 */
    matches++;                               /* step 3 */
}
```

Sorting the *suspicious* list (step 0.5) represents a one-time cost of *O(NlgN)*, while the cost per iteration is reduced from *O(N)* to *O(lgN)* given the more efficient binary search (step 2). This yields an overall algorithmic complexity of:

= O(build + sort + loop)
= O(N + N * lgN + M * lgN)
= O(N * (1 + lgN) + M * lgN)
= O(N * lgN + M * lgN)
= O((N + M) * lgN)

In our scenario, where *M* >> *N*, the *N* acts more like a small constant when added to *M*, and so version 2 of our solution requires approximately *O(MlgN)* time. So? For *N* = 1,000 suspicious IP addresses, we just reduced the execution time from an order of *1,000M* time steps to *10M*, which is 100x faster. The additional sort time of roughly 10,000 time steps (*O(NlgN)*) is more than offset by the repeated savings in search time.

Let's Go Hashing

When you know something about the data you're searching, you can (and should) build a hash table. Hashing reduces the average search time to *O(1)*—and you can't do any better than that. It does this by creating a mapping of a single value to the elements stored in the table, where that value is produced by a hash function. The key requirement for hashing is the existence of a good hash function that maps the search data across a range of integers, with few *collisions* (duplicate mappings). Our revised log file program (version 3) takes advantage of built-in support for effective string hashing in the Microsoft .NET Framework (changes in bold):

```
Dictionary<string, int> suspicious = BuildHashTable();  /* step 0 */
foreach (string line in logfile)
{
  string ipaddr = parse(line);                          /* step 1 */
  if (suspicious.ContainsKey(ipaddr))                   /* step 2 */
    matches++;                                          /* step 3 */
}
```

Building the hash table (step 0) incurs a one-time, up-front cost of *O(N)*. This yields an overall algorithmic complexity of:

= O(build + loop)
= O(N + M * 1)
= O(N + M)

Assuming our scenario of $M >> N$, version 3 is our fastest solution, at roughly $O(M)$ time. For $N = 1{,}000$ suspicious IP addresses, we just reduced the execution time by another factor of 10, which is 1,000x faster than our original version. And what's remarkable about this approach is that for even larger values of N—say, 10,000, or even 100,000—the execution time is the same, on the order of M time steps.

So what's the catch? With version 2 and binary search, it's simple: The list of IP addresses must first be sorted. In this version with hashing, there are three issues to consider: creating an effective hash function, the up-front construction cost and a larger memory footprint. The key is the hash function. An effective hash function spreads the data across the table with a minimum of collisions, eliminating the need for additional searching after the initial hash into the table. Such a function offers an up-front construction cost of $O(1)$ per datum, or $O(N)$, the same as linear search. However, such effectiveness typically involves a load factor in the neighborhood of 10 percent, which means the data occupies only 10 percent of the table. Hashing thus requires a memory footprint 10x larger than linear and binary search, though formally this is still $O(N)$ like the others.

Why Are We Doing This, Again?

After all that pseudo-math, it might strike the non-computer science graduate that we did a lot of theoretical manipulation, and while we think we found an efficient solution, we sure dropped a lot of constants along the way. And this leads us to the million-dollar question: "Does our theoretical analysis really pay off in practice?" A fair question, especially when you consider the ultimate goal of this article—to encourage you to use algorithm analysis (if you aren't already) to evaluate competing approaches before committing to code.

Many times I've sat around a table watching engineers argue for hours—even days, sometimes—over whether approach A is better than approach B. Rarely do folks pull out a napkin and perform a quick algorithmic analysis.

So, now that we've done the analysis, let's run some tests and see what happens in practice. To do that, we generated random log files with M entries and then searched these files against randomly generated lists of N IP addresses. (Complete source and input files are available at bit.ly/srchlogfiles.)

To recap, assuming $M >> N$, we derived the algorithmic complexities shown in **Figure 2**.

Figure 4 **Actual Execution Times (Seconds)**

M	N	Version 1	Version 2	Version 3
1,000,000	128	3.81	2.88	2.15
	256	5.65	2.98	2.15
	512	9.39	3.11	2.17
	1,024	16.78	3.23	2.19
	2,048	31.52	3.36	2.19
	4,096	61.18	3.49	2.28
	8,192	120.63	3.68	2.39
	16,384	238.86	3.95	2.54
	32,768	473.91	4.34	2.89

For our tests, we set M at 1 million, and double N for each test: 128, 256, 512 and so on. The doubling of N highlights the performance characteristics of the search algorithms. If our analysis is accurate, each time N is doubled, the execution time should change, as shown in **Figure 3**.

In other words, version 1 should double in time ($2t$), version 2 should take another M time steps ($t + M$) and version 3 should take no additional time (t). **Figure 4** shows the actual recorded times (in seconds).

As you can see in **Figure 4**, version 1—based on linear search—does indeed double in time as N doubles. That's probably a bad thing for most situations.

Version 2—based on binary search—runs orders of magnitude faster than version 1 and grows much more slowly. The growth rate is harder to quantify because it depends on the time T_M to execute M operations, which appears to be anywhere from 0.1 to 0.3 seconds. According to our analysis, version 2 should be growing at a constant rate of T_M as N doubles, but this doesn't appear to be the case; this could be the result of increased cache pressure (and thus cache misses) as N grows.

Finally, **Figure 4** confirms that version 3 is indeed the fastest algorithm, with essentially the same execution time regardless of N (though again not quite constant).

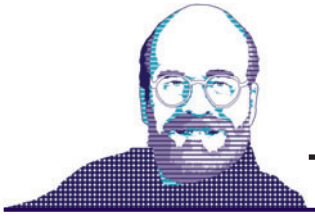
Seek Balance

From a pragmatist's perspective, it's not clear that the enhancement from version 2 to version 3 is worth anything more than a trivial amount of effort—despite the savings of more than a second. The potential savings isn't really going to make a big difference unless we start to see really big values of M or N (keeping in mind that hashing might require 10x more memory to store the IP addresses). And this raises the last point: knowing when to stop seeking to optimize. It might be exciting and compelling for a programmer to find the absolute fastest algorithm to perform some kind of operation, but unless this operation is front-and-center in the user's face, the time spent trying to optimize that last second away often could be much better spent working on something else. Clearly the jump from version 1 to version 2 is worth the time (that's a savings of about 10,000 percent), but as with all things, the programmer must seek balance. Sometimes theory must give way to practice, but at other times, the theory helps dictate where we should put our efforts into practice.

Happy coding! ■

Ted Neward *is an architectural consultant with Neudesic LLC. He has written more than 100 articles and authored or coauthored a dozen books, including "Professional F# 2.0" (Wrox, 2010). He's a C# MVP and speaks at conferences around the world. He consults and mentors regularly—reach him at ted@tedneward.com or Ted.Neward@neudesic.com if you're interested in having him come work with your team, or read his blog at blogs.tedneward.com.*

Joe Hummel *is a private consultant, a member of the technical staff at Pluralsight LLC, a trainer for MVP-Training.net and a visiting researcher at the Department of Computer Science at the University of California, Irvine. He earned a Ph.D. at UC Irvine in the field of high-performance computing and is interested in all things parallel. He resides in the Chicago area, and when he isn't sailing can be reached at joe@joehummel.net.*



CHARLES PETZOLD

Windows Phone Motion and 3D Views

The new Motion class introduced with Windows Phone 7.1 is a wonderful tool for programmers who need to know how the phone is oriented in three-dimensional space. The Motion class combines information from the sensors (accelerometer, magnetometer and gyroscope), and also smooths the data and makes it available in convenient forms.

When a Windows Phone program knows the orientation of the phone in 3D space, the phone can provide a portal into a 3D world. This facility has applications in mapping, virtual reality, augmented reality and, of course, games, but probably also some applications that have not yet been conceived. This is definitely an area where you'll want to exercise your imagination as much as your coding skills.

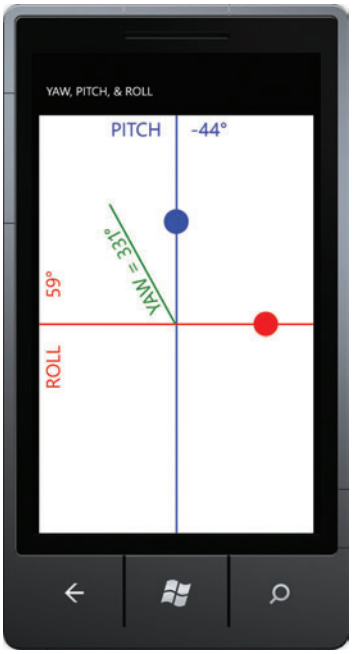


Figure 1 **The YawPitchRoll Display**

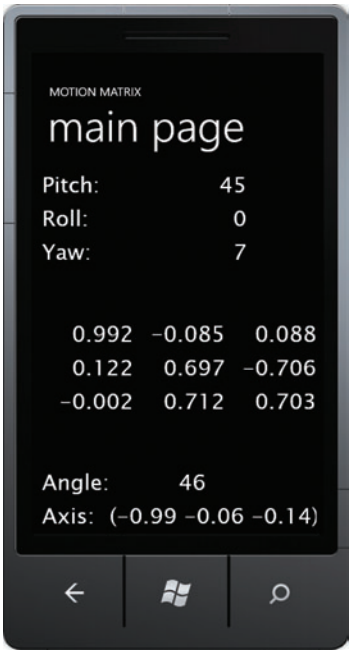


Figure 2 **The MotionMatrix Display**

Motion Data

I want to focus solely on obtaining the orientation of the phone rather than velocity or acceleration, which are also available from the Motion class. The sensor provides a MotionReading structure, and the orientation information is available in the Attitude property. The word "attitude" comes from flight dynamics, where it indicates the orientation of a plane in 3D space—distinct from altitude, which is the height above the Earth. The word is also sometimes used in vector geometry.

This Attitude property is of type AttitudeReading, another structure that defines five properties describing three-dimensional orientation:

- Yaw of type float, an angle in radians
- Pitch of type float, an angle in radians
- Roll of type float, an angle in radians
- RotationMatrix of type Matrix, a 4 × 4 matrix type defined in XNA
- Quaternion of type Quaternion, a type defined in XNA

Yaw, Pitch and Roll are also terms used in flight dynamics, and they're often referred to as Euler angles. Yaw indicates the compass direction that the nose of the plane is facing, so as the plane turns right or left, the yaw changes. Pitch changes as the nose goes up for a climb or down for a dive. Roll changes as the plane banks left and right. To visualize these relative to the phone, it helps to imagine "flying" on your phone like a magic carpet by sitting on the phone's screen with the top of the phone to your front and the three standard buttons to your rear.

Code download available at code.msdn.microsoft.com/mag201207TouchAndGo.

A little program called YawPitchRoll included with the downloadable code for this article might also help visualize these angles. (Like all programs in this article, it requires references to the Microsoft.Devices.Sensors and Microsoft.Xna.Framework assemblies.) As shown in **Figure 1**, the program displays the values of these three angles converted to degrees, and it also symbolizes their values graphically.

Yaw is displayed with a simple line pointing north like a compass, whereas Pitch and Roll are displayed as solid balls that seem to roll toward the Earth. When the phone is sitting on a level table with the screen up and the top of the phone pointing north, all three angles have zero values.

As you tilt the top of the phone up and down, you can change Pitch from 90° when the phone is upright to -90° when the top of the phone points down. Similarly, you can change Roll from 90° to -90° by tipping the phone right and left.

When the phone's display faces down, the Yaw angle points south. Pitch takes on values ranging from 90° to 180°, and from -90° to -180°. In the YawPitchRoll program, these values are symbolized by a hollow, red-outlined ball. The values of Roll continue to take on values ranging from 90° to -90°.

Figure 3 **The View3DPhonePlain Program**

```
public class Game1 : Microsoft.Xna.Framework.Game
{
    GraphicsDeviceManager graphics;
    Model model;
    Matrix scaleMatrix, worldMatrix, lookatMatrix,
           viewMatrix, projectionMatrix;
    ...
    protected override void LoadContent()
    {
        model = this.Content.Load<Model>("Phone Retro Caesar N170910");

        foreach (ModelMesh mesh in model.Meshes)
            foreach (BasicEffect effect in mesh.Effects)
                effect.EnableDefaultLighting();

        // World matrix
        scaleMatrix = Matrix.CreateScale(0.0001f);
        worldMatrix = scaleMatrix;

        // View matrix
        lookatMatrix = Matrix.CreateLookAt(new Vector3(0, 0, 10f),
                                           Vector3.Zero,
                                           Vector3.Up);
        viewMatrix = lookatMatrix;

        // Projection matrix
        float aspectRatio = graphics.GraphicsDevice.Viewport.AspectRatio;
        projectionMatrix =
            Matrix.CreatePerspectiveFieldOfView(MathHelper.PiOver4,
                                                aspectRatio,
                                                1f, 100.0f);
    }
    ...
    protected override void Draw(GameTime gameTime)
    {
        GraphicsDevice.Clear(Color.CornflowerBlue);
        model.Draw(worldMatrix, viewMatrix, projectionMatrix);
        base.Draw(gameTime);
    }
}
```

Rotational Perspectives

Although we spend our entire lives in a three-dimensional universe, many of us have a very poor intuitive grasp of three-dimensional rotation. Consequently, visualizing and working with rotations in 3D space can be challenging. Yaw, Pitch and Roll seem to adequately describe rotation in 3D space, but they turn out to be rather awkward in practical programming.

Programmers much prefer working with alternative ways to describe 3D rotation:

- A single angle describing rotation around a 3D vector (axis/angle rotation)
- A rotation matrix (a subset of a full 3D matrix transform)
- The quaternion, a 3D analogue of 2D rotation in the complex plane

These forms can all be converted between each other, as I demonstrate in Chapters 7 and 8 of my book "3D Programming for Windows" (Microsoft Press, 2008). The quaternion is particularly useful for smooth movement from one orientation to another because it lends itself to linear interpolation.

The Motion class provides 3D rotation with a rotation matrix and quaternion, but I'll focus exclusively on the RotationMatrix property of the AttitudeReading structure. This is an XNA Matrix value, which is a standard 4 × 4 transform matrix, but the matrix represents only rotation. It has no scaling and no translation. The M14, M24, M34, M41, M42 and M43 fields are all 0, and the M44 field is 1.

When working with this rotation matrix, a change in perspective will be useful. In the previous installment of this column, I described how the three-dimensional vector available from the Accelerometer and Compass sensors is relative to a 3D coordinate system imposed on the geography of the phone. However, when working with the rotation matrix, you really need to visualize two different 3D coordinate systems, one for the phone and the other for the Earth:

- In the phone's 3D coordinate system, positive Y points to the top of the phone (in portrait mode), positive X points right and positive Z comes out of the screen.
- In the Earth's 3D coordinate system, positive Y points north, positive X points east and positive Z comes out of the ground.

Although we spend our entire lives in a three-dimensional universe, many of us have a very poor intuitive grasp of three-dimensional rotation.

When the phone sits on a level surface with the screen up and the top pointing north, the RotationMatrix value is the identity matrix. Otherwise, it describes how the Earth is rotated relative to the phone, which is opposite of the rotation described by the Euler angles.

To illustrate this, I wrote a little program called MotionMatrix. The display (shown in **Figure 2**) consists only of numeric values: the Yaw, Pitch and Roll values, the 3 × 3 rotation matrix subset of the full matrix, and rotation expressed in the axis/angle form.

When you first start playing with this program, your immediate instinct is probably to orient the phone so that all the Euler angles are zero. However, the axis vector down at the bottom goes crazy because there's almost no rotation, so the axis value can be almost anything. Simple rotations calm the display. The screenshot in **Figure 2** shows the top of the phone pointing north but elevated about 45°. That's what the Pitch value reports, and the axis/angle rotation shows a very close value of 46°.

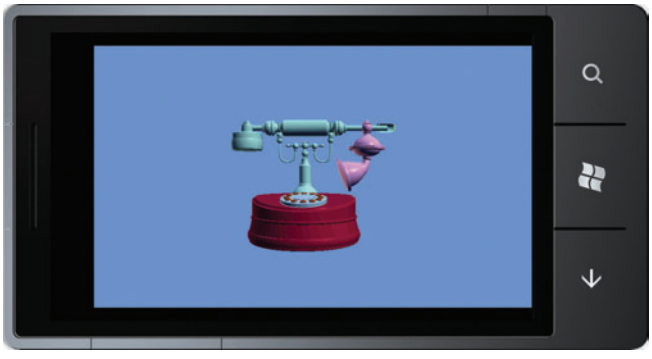


Figure 4 **The View3DPhonePlain Display**

But notice the axis of rotation: It's approximately (–1, 0, 0), which is the coordinate axis that points to the left of the phone. XNA axis/angle rotations abide by the right-hand rule: Point the thumb of your right hand in the direction of the axis (that is, to the left of the phone in this case). The curve of your fingers then shows the direction of positive rotation. This means the rotation is opposite the Pitch angle. It tells us that the phone must be rotated -45° to be aligned with the Earth.

The MotionMatrix and XNA

It seems reasonable to use the rotation matrix from the Motion class for viewing 3D objects on the phone. But rather than the normal approach—where a 3D object is rotated relative to the display, perhaps with a mouse or finger—here the display would be rotated relative to the 3D object (conceptually, anyway).

To experiment with this, I downloaded a 3D model of an antique telephone from the archive3d.net Web site (specifically bit.ly/GSNTi3) and then converted the 3DS format to the XNA-friendly FBX format using a converter tool from Autodesk. I created four Visual Studio projects that show progressively better ways of displaying this object. (Actually, I floundered around quite a bit and only later created these four projects to disguise that fact!) To reduce your source code downloading time, all four projects reference the same 3D model file.

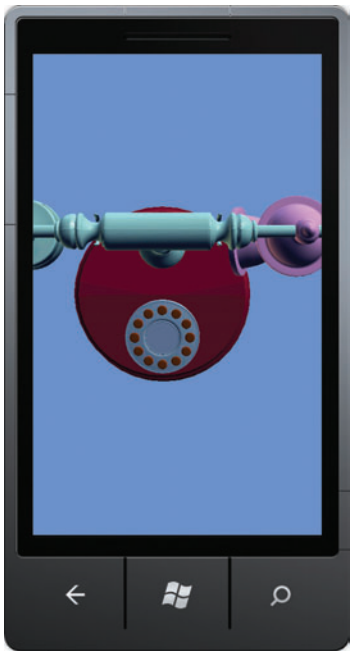


Figure 5 **The View3DPhone-Reoriented Display**

Conceptually, the 3D object should seem to remain fixed in space.

The first of the four projects is called View3DPhonePlain, which is just a static view of the object; **Figure 3** shows the relevant code. This program loads in the model, defines a world transform that scales the model down quite a lot, and defines a simple camera. (The apparent superfluity of field variables will make sense shortly.)

In a 3D graphical system like XNA, models are subjected to a series of matrix transforms on their way to the display. The world transform is applied first, and this situates the model in 3D space. The camera is a combination of two transforms applied in succession: The view transform accounts for the location and orientation of the camera, while the projection transform handles perspective effects and also normalizes all the coordinates for clipping.

Figure 4 shows how the model looks in XNA's default landscape mode, and it's pretty much as we might expect.

The next step is the project named View3DPhoneReoriented. I wanted to switch to portrait mode so that the 3D coordinates used within XNA were the same as the 3D coordinate system used by the sensors. This is a simple matter of adding a few statements to the constructor of the Game derivative:

```
graphics.IsFullScreen = true;
graphics.PreferredBackBufferWidth = 480;
graphics.PreferredBackBufferHeight = 800;
```

I also wanted to reorient the phone itself. As I mentioned, the 3D rotation matrix supplied by the Motion class is the identity matrix when the phone is sitting on a table with the screen up and the top pointing north. How did I want the 3D telephone to appear when the phone was in that position? I wanted to be looking at the top of the phone as if viewed from above, so I simply added a rotation around the X axis to the world transform:

```
worldMatrix *= Matrix.CreateRotationX(MathHelper.PiOver2);
```

The result is shown in **Figure 5**.

The camera looks a little too close to the object at this point, but this didn't bother me because I knew that when I incorporated the rotation matrix, I'd be able to rotate the phone a bit to get it all in view. (A sharp-eyed viewer of 3D light and shading might notice a conceptual flaw in the approach I'm using, but let me naively traipse down this path first and then I'll fix things up.)

Now let's incorporate the rotation matrix from the Motion class.

Conceptually, the 3D object should seem to remain fixed in space, and moving the phone should allow you to view the object from different directions. Of course, the experience is not like "real life" because you can't ever move the phone to look away from the object. The object is always there on the display, but the orientation of the display in 3D space gives you different views of it.

To achieve this effect, we need to apply a matrix to the object that achieves rotation from the phone's coordinate system to the Earth's coordinate system, where this object conceptually exists.

We should be able to achieve this effect simply by including the

Figure 6 **The Update Override in View3DPhoneWorldMotion**

```
protected override void Update(GameTime gameTime)
{
    if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
        this.Exit();

    worldMatrix = scaleMatrix * Matrix.CreateRotationX(MathHelper.PiOver2);

    if (motion != null && motion.IsDataValid)
        worldMatrix *= motion.CurrentValue.Attitude.RotationMatrix;

    base.Update(gameTime);
}
```

Figure 7 **The Update Override in View3DPhoneCameraMotion**

```
protected override void Update(GameTime gameTime)
{
    if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
        this.Exit();

    if (motion != null && motion.IsDataValid)
        viewMatrix = Matrix.CreateRotationX(MathHelper.PiOver2) *
                     motion.CurrentValue.Attitude.RotationMatrix *
                     lookatMatrix;
    else
        viewMatrix = Matrix.CreateRotationX(MathHelper.PiOver2) *
                     lookatMatrix;

    base.Update(gameTime);
}
```

Figure 8 **One View in View3DPhoneCameraMotion**

RotationMatrix object from the Motion class in the calculation of the world transform. The View3DPhoneWorldMotion project does this. It includes code to create a Motion sensor in its constructor, start it in the OnActivated override and stop it in the OnDeactivated override. I then moved the calculation of the world transform to the Update override, as shown in **Figure 6**.

In this program, as you move the phone's orientation in 3D space, you view the 3D object from all different directions. The effect is very smooth and—I must admit—very cool.

Shifting the Rotation

And yet, the more I played around with this version of the program, the more I felt that something was wrong.

By applying the rotation matrix to the world transform, the 3D object is effectively rotated not only relative to the camera, but also relative to the light source. In some scenarios, this would be correct. For example, if you implement a touch interface so you can turn the 3D image around with your fingers, applying the rotation matrix to the world transform would be appropriate.

But for this program, the paradigm is quite different. I wanted the phone's display to be like a mobile camera that moves relative to the 3D object, and that means this object should remain fixed relative to the light source. The rotation matrix from the Motion sensor needs to be applied to the view transform rather than the world transform.

For the final version of the program—called View3DPhoneCameraMotion—I restored the calculation of the world transform to the original version:

```
worldMatrix = scaleMatrix;
```

Even applying the initial rotation was a mistake because it implied that the light was coming from behind the phone rather than from above. The new Update method is shown in **Figure 7**.

You really need to try out this program on a phone to get the full effect, but the image in **Figure 8** shows one possible view. ■

CHARLES PETZOLD *is a longtime contributor to MSDN Magazine and is currently updating his classic book "Programming Windows" (Microsoft Press, 1998) for Windows 8. His Web site is charlespetzold.com.*

THANKS to the following technical expert for reviewing this article: *Donn Morse*

DAVID S. PLATT

The Patient Knows What's Wrong with Him

I hear the same complaint in every UI class I teach: "The users don't know what they want. How can we build it for them if they won't tell us?"

This isn't a new problem. At my first job, my boss handed around a poem entitled "T'was the Nite Before Implementation" (bit.ly/189U65). It ended with the lines:

And the user exclaimed with a snarl and a taunt,
"It's just what I asked for, but not what I want!"

That was 30 years ago, and I doubt the poem was new then.

Users have valuable information for us. Their satisfaction is our ultimate quality metric. But users don't know how to design UIs, in the same way that medical patients don't know how to diagnose their own diseases. They can tell you if something feels good or bad once you ask them. But pulling the correct diagnosis out of the void isn't their problem. It's ours, trained professionals that we claim to be.

We need to learn to interview our target users and observe them at work. UX designers who can do that well are good designers.

Think about it. When a patient goes to the doctor, he doesn't usually say, "I think I have type C fulminating leprosy, Kaminski variation." No. He says something like, "Ow, my elbow hurts." It's the doctor's job to figure out whether the patient's elbow hurts because cancer is eating away at his bones, or his wife's cheating on him and he's displacing anxiety, or maybe he just banged it on something. It's a difficult skill to learn, and doctors who can do it well are good doctors.

My father, a retired physician, comes from the old school of medicine, which subscribes to this belief: The information you need to make the diagnosis is in the patient's head. But the patient doesn't know which pieces are important, or how to relate one piece to another. It's the doctor's job to ask the right questions: "When did your elbow start hurting? Does your other elbow hurt too? Does it hurt when I bend it this way? That way?" Lab tests and imaging might confirm it, but all diagnosis stems from interviewing and examining the patient.

Likewise, it's our job to ask the right questions, extract the relevant data from our users' heads and use it to construct a good UX. We need to learn to interview our target users and observe them at work. UX designers who can do that well are good designers.

It's also important to distinguish between symptom and cause. My father tells the following story about one of the first patients he ever treated. The ink wasn't even dry on his M.D. diploma as he started his first job at the hospital clinic. (Brand-new doctors were called interns back then, before Bill and Monica gave that term a different meaning.)

A patient came in complaining, "I haven't had a bowel movement in a week." "OK," thought my dad. "Bowel movement. No problem. Four years of medical school; I can handle this." He prescribed a mild laxative; nothing happened. Tried a stronger one; still nothing. Getting a little frustrated, he tried an enema. Nothing again. An ultrasound was negative, an MRI scan didn't show anything, and blood work was unremarkable.

Getting desperate, he took the patient up to the operating room and did an exploratory laparotomy; sliced open his abdomen to look around. There was nothing to see—the patient had a completely normal belly. Finally, he did what he should have done in the first place: took a thorough history, during the course of which he asked the patient what he did for a living. The patient said, "I'm a musician." Finally, the light bulb flashed on. "Ah! Of course!" exclaimed my dad. "Look, here's 10 bucks. Go buy yourself something to eat."

So it is with our user population. They have some notion of what hurts: "I worked for three hours, then up came this stupid box saying, 'Microsoft Word has encountered an error and needs to close,' and all my work was gone." It's not their job to say, "Jeez, I wish some animated character with bushy eyebrows would pop up and say, 'It looks like you're about to crash. Don't you think you ought to save your work?'" It's our job to figure out what's really hurting them, so we can make it stop. ■

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.*

Free
E-book

HTTP
Succinctly

by Scott Allen

The newest addition to
Syncfusion's *Succinctly* Series

Download *HTTP Succinctly* today to learn about:

HTTP from the software developer's perspective
HTTP connections and optimization
Web architecture, proxies, and caching

Get it here: syncfusion.com/httpebook

Syncfusion | Technology Resource Portal