DevExpress

# Taking Touch to the Next Level.

Touch-enabled app development requires developers to re-think the future of their user experiences. Whether you're a WinForms developer building client applications or a Web developer building for mobile platforms, DevExpress 12.1 tools help you take on these new challenges using your existing skills & 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

Copyright © 1998-2012 Developer Express Inc. ALL RIGHTS RESERVED. All trademarks or registered trademarks are property of their respective owners.

THE MICROSOFT JOURNAL FOR DEVELOPERS

JUNE 2012 VOL 27 NO 6

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!**

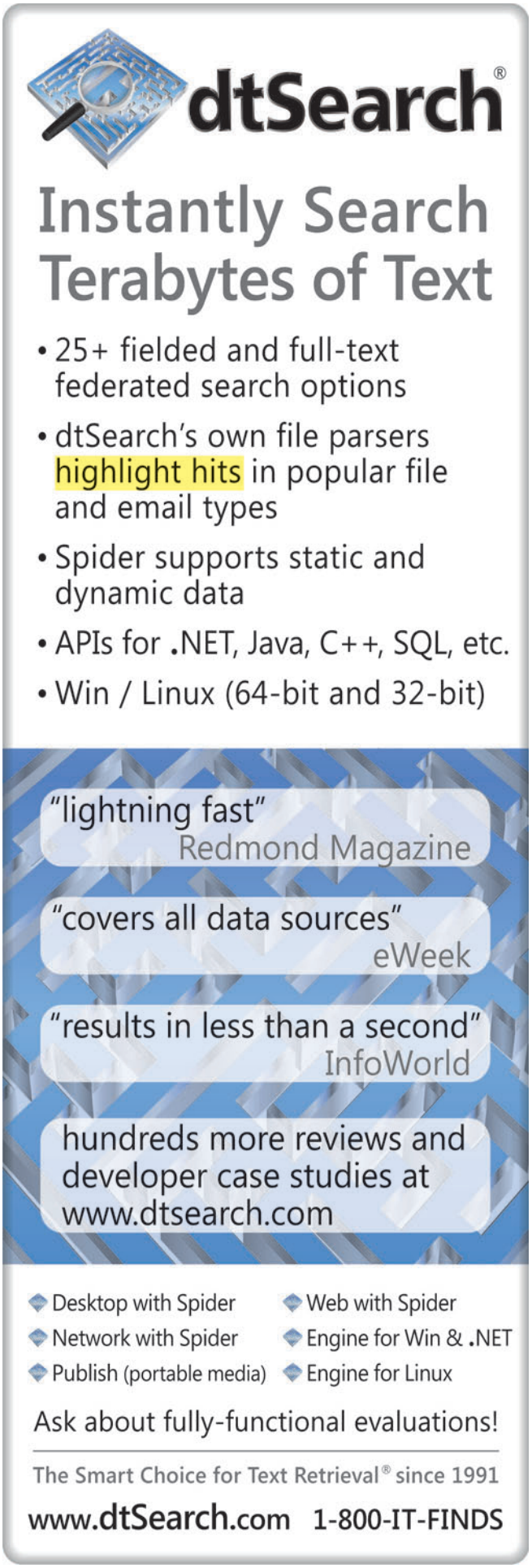
www.infragistics.com/NET

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

Copyright 1996-2012 Infragistics, Inc. All rights reserved. Infragistics and NetAdvantage are registered trademarks of Infragistics, Inc. The Infragistics logo is a trademark of Infragistics, Inc. All other trademarks or registered trademarks are the respective property of their owners.


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

JUNE 2012 VOLUME 27 NUMBER 6

**MITCH RATCLIFFE** Director
**LUCINDA ROWLEY** Editorial Director/mmeditor@microsoft.com
**PATRICK O'NEILL** Site Manager

**MICHAEL DESMOND** Editor in Chief/mmeditor@microsoft.com
**DAVID RAMEL** Technical Editor
**SHARON TERDEMAN** Features Editor
**WENDY HERNANDEZ** Group Managing Editor
**KATRINA CARRASCO** Associate Managing Editor

**SCOTT SHULTZ** Creative Director
**JOSHUA GOULD** Art Director

**CONTRIBUTING EDITORS** Dino Esposito, Joseph Fultz, Kenny Kerr, Julie Lerman, Dr. James McCaffrey, Ted Neward, John Papa, Charles Petzold, David S. Platt

Redmond Media Group

**Henry Allain** President, Redmond Media Group
**Doug Barney** Vice President, New Content Initiatives
**Michele Imgrund** Sr. Director of Marketing & Audience Engagement
**Tracy Cook** Director of Online Marketing

ADVERTISING SALES: 508-532-1418/mmorollo@1105media.com

**Matt Morollo** VP/Group Publisher
**Chris Kourtoglou** Regional Sales Manager
**William Smith** National Accounts Director
**Danna Vedder** Microsoft Account Manager
**Jenny Hernandez-Asandas** Director, Print Production
**Serena Barnes** Production Coordinator/msdnadproduction@1105media.com

1105 MEDIA, INC

**Neal Vitale** President & Chief Executive Officer
**Richard Vitale** Senior Vice President & Chief Financial Officer
**Michael J. Valenti** Executive Vice President

**Christopher M. Coates** Vice President, Finance & Administration
**Erik A. Lindgren** Vice President, Information Technology & Application Development
**David F. Myers** Vice President, Event Operations

**Jeffrey S. Klein** Chairman of the Board

*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.

Printed in the U.S.A. Reproductions in whole or part prohibited except by written permission. Mail requests to "Permissions Editor," c/o *MSDN Magazine*, 4 Venture, Suite 150, Irvine, CA 92618.

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

**Corporate Address:** 1105 Media, 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®

## THE WORLD LEADER IN IMAGING SDKs

LEADTOOLS provides developers easy access to decades of expertise in developing color, grayscale, document, medical, vector and multimedia imaging technology. Develop with LEADTOOLS to eliminate months of research and development and add essential functionality to your application without sacrificing quality or performance.

| | | | | |
|---|---|---|---|---|
| OCR | BARCODE | PDF & PDF/A | FORMS RECOGNITION & PROCESSING | 150 + FILE FORMATS |
| DOCUMENT PREPROCESSING | | ANNOTATIONS | SCANNING | |
| DICOM & PACS | MEDICAL WORKSTATION | | IMAGE PROCESSING | VIRTUAL PRINTER |
| MPEG-2 TRANSPORT | | MULTIMEDIA PLAYBACK & CAPTURE | DVD & DVR | CODECS |

ZERO FOOTPRINT AND RICH CLIENT CONTROLS FOR DOCUMENT & MEDICAL

## C++ .NET METRO HTML5
## WEB SERVICES, WPF, ASP.NET & CLOUD

DOWNLOAD OUR 60 DAY EVALUATION
WWW.LEADTOOLS.COM

LEAD TECHNOLOGIES INCORPORATED

SALES@LEADTOOLS.COM
800.637.1840

# EDITOR'S NOTE

MICHAEL DESMOND

# Old Soldiers Never Die

General Douglas MacArthur famously said during his 1951 farewell address to the U.S. Congress: "Old soldiers never die, they just fade away." A half-century later, MacArthur's famous statement could just as easily apply to programming languages. Specifically, Visual Basic.

As Don't Get Me Started columnist David Platt recounts in this month's issue of *MSDN Magazine*, Microsoft has formally extended "It Just Works" support for its Visual Basic 6 programming language through the full lifetime of the Windows Vista, Windows Server 2008, Windows 7 and Windows 8 OSes. That means the core Visual Basic 6 runtime in each OS will enjoy five years of mainstream support, followed by another five years of extended support, from the point that a given OS shipped.

It's no surprise that developers are reluctant to port perfectly good code, and often opt to maintain existing code and extend functionality using contemporary tools and languages.

While Microsoft has extended support for Visual Basic 6 within its Windows OSes, the announcement doesn't address support of the actual development tools. Microsoft retired mainstream support for Visual Basic 6 Enterprise and Standard Editions on March 31, 2005. Extended support for the IDE ended on April 8, 2008.

As Platt notes in his column, Visual Basic 6 continues to thrive more than 10 years after Microsoft launched Visual Basic .NET to replace it. At the core of the language's enduring appeal is its focus on simplicity. "My rule of thumb for Visual Basic 6 was: if I couldn't do it within 10 minutes, I couldn't do it at all," Platt writes in his column.

Karl Peterson agrees. A longtime Visual Basic 6 developer and former columnist for *Visual Studio Magazine* and *Visual Basic Programming Journal*, Peterson has forgotten more about Visual Basic 6 than most of us will ever know. He says developers remain loyal to the language because it continues to do what they need it to do. No more. No less.

"There has yet to be a good reason to migrate Classic VB code. There's nothing wrong with picking up new languages and starting new projects with them. But rewriting functional code, non-recreationally, just doesn't pencil out," says Peterson, who calls Visual Basic 6 "the COBOL of the 2020s."

It's no surprise that developers are reluctant to port perfectly good code, and often opt to maintain existing code and extend functionality using contemporary tools and languages. It's an imperfect solution, but one that balances cost against stability.

"Businesses run on Classic VB. Governments run on Classic VB," Peterson continues, before adding that he doesn't think Microsoft would expect corporations or governments to "move to an operating system that doesn't support Classic VB code."

Visual Basic 6 is hardly alone as a legacy language that continues to appeal to an active community of developers. Platt says he sees "pockets" of support on the Internet for Borland Delphi, Microsoft FoxPro and Sybase PowerBuilder. There are even devs out there still working with DEC software, he says. Visual Basic, however, is unique.

"None of them is anywhere near as large as VB6, though. That huge quantity has a quality all its own," Platt says.

What's apparent is that legacy tools and software can thrive long after the vendor has moved on.

"It's not like hardware, where the lack of spares forces you off the platform," Platt says. "The Internet makes it much easier to connect with other people who use the same legacy as you.

"I'm thinking of starting a consulting company that does only legacy work, leveraging all these 50-year-old guys with institutional knowledge that can't be found anywhere else," Platt continues, and I'm not sure if he's joking or not. "I'll call it Graybeard Consulting, or Old Fart Consulting, or something like that. Because of all the books and articles I've written on it over the years, I still get calls to do COM now and again, and I do it if the project is interesting. And, of course, if the price is right."

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

© 2012 Microsoft Corporation. All rights reserved.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, you are not permitted to reproduce, store, or introduce into a retrieval system *MSDN Magazine* or any part of *MSDN Magazine*. If you have purchased or have otherwise properly acquired a copy of *MSDN Magazine* in paper format, you are permitted to physically transfer this paper copy in unmodified form. Otherwise, you are not permitted to transmit copies of *MSDN Magazine* (or any part of *MSDN Magazine*) in any form or by any means without the express written permission of Microsoft Corporation.

A listing of Microsoft Corporation trademarks can be found at microsoft.com/library/toolbar/3.0/trademarks/en-us.mspx. Other trademarks or trade names mentioned herein are the property of their respective owners.

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

# 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: Markup

As more and more devices are used to browse the Web, having a mobile-optimized Web site is becoming a critical factor for companies. Let's face it: Even with most modern smartphones, browsing a desktop site is quite problematic. You need pan and zoom to make sense of content and follow links. Doable, sure, but not a fantastic user experience.

I use my smartphone to browse desktop sites, and I often prefer to see the full version rather than the mobile version when available. I do this for one key reason: Mobile sites often aren't as well designed as desktop sites. As a frequent user of a site, you develop certain expectations. You expect to find all of them met by the site regardless of the medium you use to reach the site. If you reach the site with a mobile device, you ideally expect the same level of service, but targeted to your current status—that of a mobile user, possibly on the go, possibly hurried and possibly using a poor connection.

This is precisely the challenge that architects and developers of a mobile site face. What's most problematic with a mobile site isn't necessarily the implementation of functionalities. A mobile site generally has fewer functions and actions per page than a desktop site, and fewer lines of code means fewer problems. However, to offer a great user experience, commonly requested functionalities are required even more urgently than for a desktop site. Ensuring a great user experience on mobile devices can be challenging, and often a team has to resort to extremely clever coding and design solutions.

In addition, mobile sites might need to serve a wide range of different devices. If dealing with different desktop browsers scared you until a few years ago, think about the mobile space, where the different device profiles you might be dealing with are in the order of thousands. (This doesn't mean, however, that you must arrange for thousands of different versions of a page—you just reduce these thousands of device profiles to a few classes of devices that you handle differently.)

This article is the first of a series in which I'll try to approach mobile site development using a perspective that isn't primarily focused on technology. I find that too often mobile site development is associated with specific frameworks and their solutions, without spending much time thinking over use cases and restructuring of the content. In this article I'll start from the basics—the mobile markup.

## Browsers and the Viewport

By default, most mobile browsers today render pages in viewports far larger than the actual device screen sizes. The actual viewport width varies with the browser, but it's always close to 1,000 pixels. For example, it's known to be 980 pixels in Safari for iPhone and Internet Explorer 9, and 800 pixels on Android WebKit. As a mobile developer, your first task should be understanding the role and origin of the viewport. **Figure 1** shows how mobile browsers use the viewport.

The viewport is a fixed-width area where the browser renders any content. The viewport size is not necessarily related to the physical screen size. This isn't a big deal in a desktop browser, but it becomes an issue on the significantly smaller screens of mobile devices. Browsers tend to render pages in whatever their internal viewport size is. When it comes to display, however, they zoom out the viewport so that the whole content fits in the real screen size.

The net effect is that, as a user, either you need to scroll horizontally and vertically to reach content or you need to zoom in to make content readable, and, more importantly, allow yourself to follow links. An excellent resource to learn more on internal implementation of viewports in browsers is at bit.ly/bZYlKb.

## Viewport and Markup

In the beginning of mobile site development, the markup used wasn't the same as in desktop sites. Over the years, we moved from Wireless Markup Language (WML) to compact HTML (cHTML) to extensible HTML (XHTML) for mobile. In all of these cases, a distinct Multipurpose Internet Mail Extensions (MIME) type was served to the browser. Based on that, the browser could figure out what type of content it was going to receive and render it accordingly. With the advent of the iPhone it was clear that the capabilities of mobile devices could be compared to laptops, at least for rendering Web pages. But serving the same markup to mobile and desktop browsers raised the issue of different screen sizes and forced mobile users to zoom content in and out as appropriate.

If the same MIME type (text/html) is used for both desktop and mobile requests, the browser can no longer use the MIME type to figure out whether you intend the content to be laid out on a desktop- or mobile-sized viewport. That's why Apple Inc. introduced the viewport meta tag a few years ago. Now the viewport meta tag is a de facto standard, and any page intended to be a mobile page should have it set. Here's how you typically use the viewport tag:

```
<meta name="viewport" content="width=device-width" />
```

The content attribute of the meta tag is set to an expression where a few properties can be set to ad hoc values. The most important of these properties is width. Browsers use the width property to dynamically set the size of their viewports. You can set the width property to a specific number of pixels or to a special expression, as in the preceding example. In particular, the value device-width indicates the width of the screen in pixels. Note that in this context a pixel is intended as a device-independent pixel or a

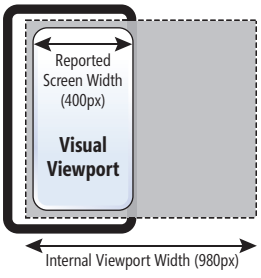


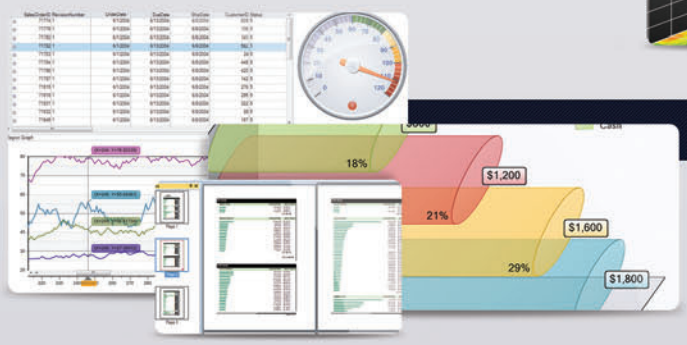

Figure 1 **How Mobile Browsers Use the Viewport**

# <xaml>
# <windows>
# <web>

windows forms · wpf · silverlight · wp7 · webforms · mvc · html5 · iphone · compact framework · entity framework · activex · lightswitch

## <xaml>

Develop for the desktop and web at once, and deliver line-of-business apps that take advantage of the latest trends.


## <windows>

Embrace the powerful desktop with next-generation controls complete with built-in features, smart designers, and hundreds of samples.

## <web>

The best is standard in our tools for ASP.NET and HTML5 apps; this includes application-wide theming, full cross-browser support, unmatched performance, and built-in Web Standards.


## *FREE TRIAL @: http://c1.ms/ultimate*

*Build everything... everywhere* with the ultimate collection of tools from ComponentOne. Create any type of application, reach every platform, and impress your end users with stunning UIS and boosted performance.

ComponentOne

© 2012 ComponentOne LLC. All rights reserved. All other product and brand names are trademarks and/or registered trademarks of their respective holders.

Figure 2 **Properties in a Viewport Expression**

| Property | Description |
|---|---|
| width | Indicates the desired viewport width in device-independent pixels. It can be an explicit number (for example, 240) or, better yet, a relative value such as device-width. |
| height | Indicates the desired viewport height in device-independent pixels. It can be an explicit number (for example, 320) or, better yet, a relative value such as device-height. |
| initial-scale | Indicates the desired zoom level when the page is first loaded. A value of 1 indicates the page should be rendered with its natural size—no zoom at all. A value of 2 will double the size of the page, and so on. You can use decimal values as well, such as "1.0." |
| minimum-scale | Indicates the minimum level of zoom allowed for the page. A value of 1 indicates the user can't zoom out to shrink the page more than its natural size. |
| maximum-scale | Indicates the maximum level of zoom allowed for the page. Maximum value is 10.0. |
| user-scalable | A yes/no property that indicates whether the user is allowed to zoom on the page. |

CSS pixel. On devices with a high density of pixels the browser is forced to rescale, and it might be that a width of 100 pixels covers some 150 or even 200 physical pixels. **Figure 2** lists the properties supported in a viewport expression.

Some browsers might have extra properties. In particular, Android browsers also understand the target-densitydpi property through which developers communicate the desired screen resolution intended for the page. This might help Android browsers to optimize scaling of pixel-based resources such as images. The target-densitydpi property can be set to a specific number or to a special value such as device-dpi.

Here's a more specific way to set the viewport meta tag:

```
<meta name="viewport"
  content="width=device-width, user-scalable=no, initial-scale=1"/>
```

The height property isn't used often. You typically use it if you have elements that are to be placed at the bottom of the screen or in a position dependent on the height of the viewport. Finally, note that setting user-scalable to "no" will disable pinch-and-zoom on that page.

The viewport meta tag is pretty common today, but in the past other meta tags such as HandheldFriendly and MobileOptimized were used by some browsers for the same purpose:

```
<meta name="HandheldFriendly" content="true" />
<meta name="MobileOptimized" content="320" />
```

The World Wide Web Consortium (W3C) is elaborating a paper to standardize the viewport element. You can read the current draft at bit.ly/AavTG5.

## HTML and XHTML Mobile Profile

The ultimate goal of the viewport and other meta tags is to indicate the size of the viewport on which the developer wants the content rendered. You use meta tags if you serve plain HTML markup to mobile browsers. If, for some reason, it's OK for you to serve mobile markup, you don't need meta tags, but just a document with an XHTML Mobile Profile (MP) DOCTYPE:

```
<!DOCTYPE html PUBLIC
  "-//WAPFORUM//DTD XHTML Mobile 1.2//EN"
  "http://www.openmobilealliance.org/tech/DTD/xhtmlmobile12.dtd">
```

The MIME type for XHTML MP is "application/vnd.wap.xhtml+xml." Compared to the expressivity of plain HTML, XHTML MP is a thing of the past because it has limitations in Document Object Model (DOM) manipulation and important areas such as support for CSS and JavaScript.

The current standard that works for most of today's devices is using HTML with the viewport tag. However, it might be that your site should be open to a really large set of devices, including old devices that don't support HTML or the viewport tag. How can you address this situation?

As blunt as it might sound, you need some help from special databases that store information about the capabilities of mobile browsers. These databases are known as Device Description Repositories (DDRs). Wireless Universal Resource File (WURFL)—the DDR used by Facebook—has a property called preferred_markup that you can query on the server (for example, from within an ASP.NET application) to determine the most appropriate markup to serve to a requesting device. You can read more about WURFL at wurfl.sourceforge.net. I plan to cover WURFL and other DDRs such as 51Degrees in future columns.

## HTML5 and Mobile Browsers

HTML5 is a markup language that modern mobile browsers support quite well—at least for the current state of the standard. This applies to browsers embedded in smartphones (that is, Safari on iPhone and iPad, Android WebKit and Internet Explorer 9 on Windows Phone) as well as external browsers such as Fennec (the mobile version of Firefox) and Opera Mobile.

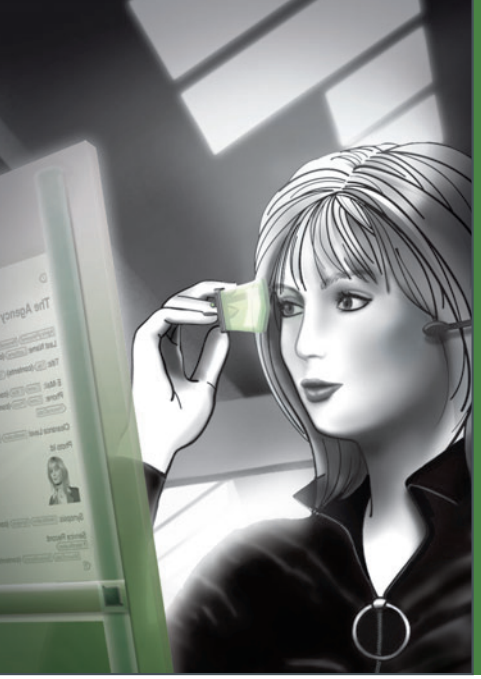
In particular, you can check out mobilehtml5.org for a thorough comparison of HTML5 support on a variety of mobile browsers. You'll see that Canvas and SVG support is nearly ubiquitous, whereas input elements—critical on mobile pages—are well supported on iOS 5 but not on current versions of Android WebKit. Local storage, geolocation and multimedia are also nearly everywhere.

## Not That Smart

Too often a mobile site is simply perceived as an offspring of the existing desktop site. So it happens that most of the design is done from the perspective of the desktop site while assuming that the client is as "smart"—or rich and powerful—as a desktop browser. But smartphones aren't as "smart" as laptops. This large fragmentation is a critical aspect of mobile site development. For this reason, when you start a mobile site project, the first aspects to consider are use cases to cover and devices to target. Any technology you might want to use comes later. In future columns I'll discuss techniques to identify ad hoc mobile use cases, patterns of mobile development and device-sensitive rendering. ■

**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.*

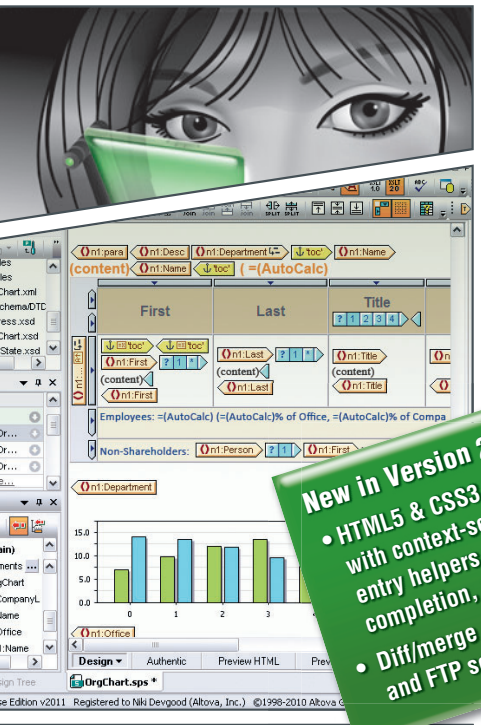
**THANKS** to the following technical experts for reviewing this article: *Shane Church and Steve Sanderson*



# Uncover Web development simplicity with the complete set of tools from Altova®

ALTOVA® missionkit®

**Experience how the Altova MissionKit® suite of integrated XML and database tools can simplify even the most advanced Web 2.0 development projects.**



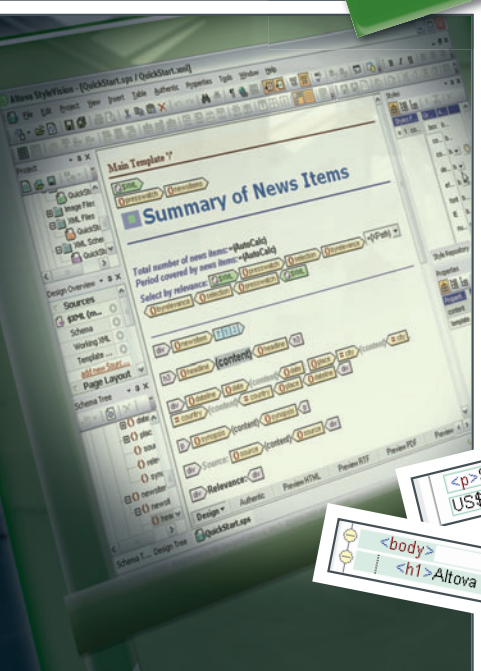
## The Altova MissionKit includes powerful Web development tools:

**XMLSpy®** – advanced XML editor
- Intelligent HTML5 & CSS3 editing
- XSLT 1.0/2.0 editing help, plus debugging & profiling

**StyleVision®** – graphical stylesheet & report designer
- Drag-and-drop XSLT 1.0/2.0 stylesheet and Web page design
- Powerful XML, XBRL, & database report builder
- Advanced CSS and JavaScript functionality

**DiffDog®** – XML-aware diff / merge utility
- File, folder, directory, DB comparison and merging
- One-click directory syncing; Web and FTP server support

**New in Version 2012:**
- HTML5 & CSS3 editing, with context-sensitive entry helpers, code completion, & more
- Diff/merge on HTTP and FTP servers




Download a 30 day free trial!

Try before you buy with a free, fully functional trial from **www.altova.com**.


Scan to learn more about Web development with the MissionKit.



JULIE LERMAN

# Data Bind OData in Web Apps with Knockout.js

As a data geek, I spend quite a lot of time writing back-end code and miss out on a lot of the fun stuff on the client side. John Papa, who used to pen this column, is now writing a column on client technologies for this magazine, and he's been doing a lot of work on a hot new client-side technology called Knockout.js. Thanks to the passion that Papa and others have been expressing about Knockout.js, I jumped at an offer for a presentation on Knockout at my local user group, VTdotNET, by Jon Hoguet from MyWebGrocer.com. The meeting drew a larger crowd than we typically get, and included devs from outside the .NET community. As Hoguet worked his way through the presentation, it became obvious to me why so many Web developers are enticed by Knockout: It simplifies client-side data binding in Web applications by leveraging the Model-View-ViewModel (MVVM) pattern. Data binding … Now that's something I can do with data! The next day I was already learning how to weave Knockout.js with data access—and now I can share my discoveries with you.

I should provide fair warning that my JavaScript skills are terribly lacking, so this took me a bit longer to work out than it ought to have. However, I'm guessing that many of you who read this column are in the same boat, so you just might appreciate my baby steps. You can get a much deeper understanding of Knockout from Papa's columns, as well as his excellent course on Pluralsight.com.

> My goal was to see how I could use Knockout.js to bind to and then update data retrieved from a WCF Data Service.

My goal was to see how I could use Knockout.js to bind to and then update data retrieved from a WCF Data Service. What I'll do in this article is show you the critical working parts. Then you can see how they all fit together in the accompanying download sample.

I began with an existing WCF Data Service. In fact, it's the same service I used in my December 2011 Data Points column, "Handling Entity Framework Validations in WCF Data Services" (msdn.microsoft.com/magazine/hh580732), which I've now updated to the recently released WCF Data Services 5.

**Code download available at code.msdn.microsoft.com/mag201206DataPoints.**

Figure 1 **Executing a Query and Handling the Response**

```
OData.read({
  requestUri: peopleFeed + "?$top=1"
  },
  function (data, response) {
    mapResultsToViewModel(data.results);
  },
  function (err) {
    alert("Error occurred " + err.message);
  });

  function mapResultsToViewModel(results) {
    person = results[0];
    vm = personToViewModel(person)
    ko.applyBindings(vm);
}
```

As a reminder, my demo-ware model comprised a single class:

```
public class Person
{
  public int PersonId { get; set; }
  [MaxLength(10)]
  public string IdentityCardNumber { get; set; }
  public string FirstName { get; set; }
  [Required]
  public string LastName { get; set; }
}
```

A DbContext class exposed the Person class in a DbSet:

```
public class PersonModelContext : DbContext
{
  public DbSet<Person> People { get; set; }
}
```

The Data Service then exposed that DbSet for reading and writing:

```
public class DataService : DataService<PersonModelContext>
{
  public static void InitializeService(DataServiceConfiguration config)
  {
    config.SetEntitySetAccessRule("People", EntitySetRights.All);
  }
}
```

With Knockout.js, a client-side script can respond to changes in the property values of data-bound objects. For example, if you call the Knockout applyBindings method in your script, passing in an object, Knockout will alert any data-bound element of updates to that object's properties. You can achieve similar behavior for collections that Knockout is watching as they acquire or eject items. All of this happens in the client with no need to write any event-handling script or return to the server for help.

There are a number of tasks I had to perform to achieve my goal:

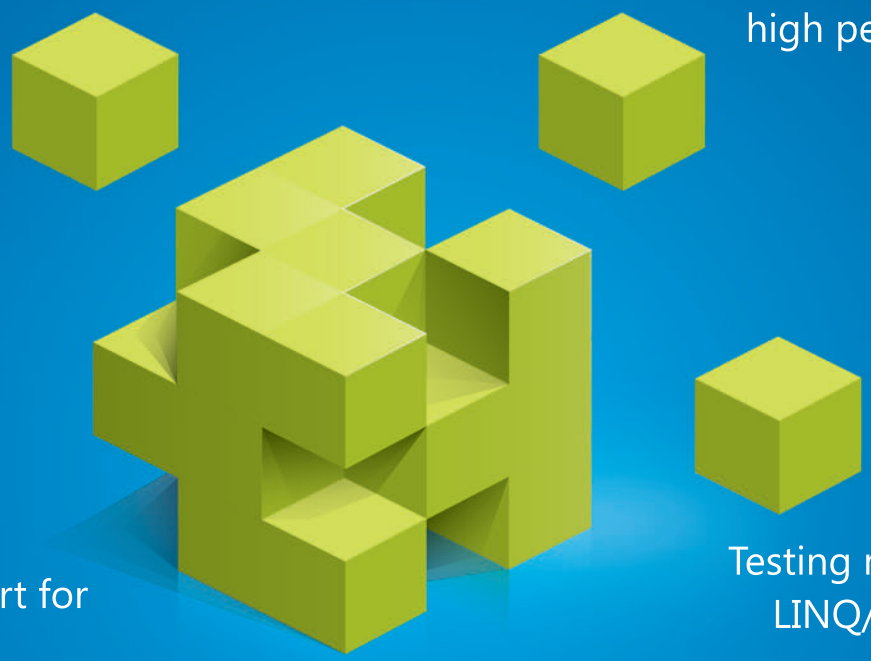
- Create Knockout-aware view models.
- Get OData in JSON format.
- Move the OData results into my view model object.

# First-Class ORM Model Designer

## for NHibernate, Entity Framework, and LINQ to SQL

Rich design experience – visual editors, intuitive editing – everything for your convenience

Advanced support for large models – multiple diagrams, high performance



Deep support for each of the ORM solutions

Testing model with LINQ/ESQL/HQL queries

Configurable T4-based code generation with support for validation engines

Synchronization of model and database in any direction on any stage of development

**Entity Developer** is a powerful modeling and code generation tool for NHibernate, Entity Framework, and LINQ to SQL with support for model-first, database-first, and mixed design approaches, and flexible template-based code generation. Entity Developer empowers model design process with numerous benefits over standard tools.

devart

www.devart.com

- Bind the data with Knockout.js.
- For updates, move the view model values back into the OData result object.

## Creating Knockout-Aware View Models

In order for this to work, Knockout needs to be able to "observe" the properties of that object. You can enable this using Knockout when you define the properties in your object. But wait! I'm not suggesting that you "dirty up" your domain objects with UI-specific logic so that Knockout can observe value changes. This is where the MVVM pattern comes in. MVVM lets you create a UI-specific version (or view) of your model—that's the VM (ViewModel) of MVVM. This means you can pull your data into your app however you like (querying against WCF Data Services, hitting a service or even via the new Web API), then reshape the results to align to your view. For example, my data service returns Person types with a FirstName, LastName and IdentityCard. But in my view, I'm only interested in FirstName and LastName. You can even apply view-specific logic in the view model version of your object. This gives you the best of both worlds: you get an object that's specifically targeted to your view, regardless of what the data source provides.

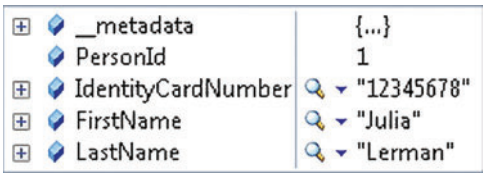


Figure 2 **Person Data from the OData Service**

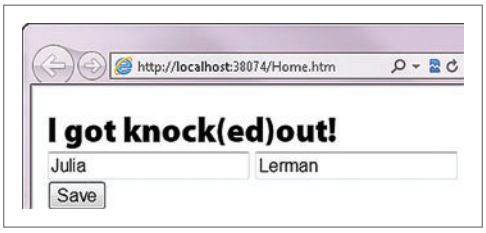


Figure 3 **The PersonViewModel Object Bound to Input Controls**

> By default, OData results are returned as ATOM.

Here's the client-side PersonViewModel that I've defined in a JavaScript object:

```
function PersonViewModel(model) {
  model = model || {};
  var self = this;

  self.FirstName = ko.observable(model.FirstName || ' ');
  self.LastName = ko.observable(model.LastName || ' ');

}
```

No matter what's returned from the service, I only want to use the first and last name in my view, so these are the only properties it contains. Notice that the names aren't defined as strings but as Knockout observable objects. This is important to keep in mind when setting values, as you'll see further on.

## Getting OData in JSON Format

The OData query I'll use will simply return the first Person from the data service. Currently, that's coming from my development server:

```
http://localhost:43447/DataService.svc/People?$top=1
```

By default, OData results are returned as ATOM (which is expressed using XML). Knockout.js works with JSON data, though, which OData can also provide. Therefore, because I'm working directly in JavaScript, it's a lot simpler to deal with JSON results than with XML. In a JavaScript request, you can append a parameter to the OData query to specify that the results are returned as JSON: "$format=json." But this requires that your particular data service knows how to process the format query option. Mine doesn't. If I want to go this route—for example, if I'm using AJAX to make my OData calls—I have to use an extension in my service to support the JSON output (see bit.ly/mtzpN4 for more information).

However, because I'm using the datajs toolkit for OData (datajs.codeplex.com), I don't need to be concerned with this. The toolkit's default behavior is to automatically add header information to requests so that they'll return JSON results. So I won't need to add the JSONP extension to my data service. The OData object from the datajs toolkit has a read method that lets you execute a query whose results will be in JSON format:

```
OData.read({
  requestUri: http://localhost:43447/DataService.svc/People?$top=1"
  })
```

## Pushing OData into PersonViewModel

Once the results have been returned—in my case a single Person type as defined by my domain model—I then want to create a PersonViewModel instance from the result. My JavaScript method, personToViewModel, takes a Person object, creates a new PersonViewModel from its values and then returns the PersonViewModel:

```
function personToViewModel(person) {
  var vm=new PersonViewModel;
  vm.FirstName(person.FirstName);
  vm.LastName(person.LastName);
  return vm;
}
```

> With Knockout.js, a client-side script can respond to changes in the property values of data-bound objects.

Notice that I'm setting the values by passing in the new values as though the properties are methods. Originally, I set the values using vm.FirstName=person.FirstName. But that turned FirstName into a string, rather than an observable. I struggled for a while, trying to see why Knockout wasn't noticing subsequent changes to the value, and finally had to give in and ask for help. The properties are functions, not strings, so you need to set them using method syntax.

I want to run personToViewModel in response to the query. This is possible because OData.read lets you tell it what callback method to use when the query has executed successfully. In this case, I'll pass the results to a method called mapResultsToViewModel,

CSSLP®

Certified Secure Software Lifecycle Professional

# Is your software open to attacks?

# FREE CSSLP® Webcast Series: Securing each phase of the SDLC

**www.isc2.org/csslppreview.aspx**

Slam the door on would-be attackers by learning best practices for securing each phase of the software lifecycle. Watch these 10-15 minute webcasts, which will show you what security measures need to take place at the beginning in the **requirements phase**, how security must be built in the **design phase**, and how to test if the application is resilient enough to withstand attacks in the **testing phase**.

Also, this series will feature a webcast on the **value of the CSSLP** and **how to study** for the exam.

*Connect with us:*
*www.isc2intersec.com*
*www.twitter.com/isc2*
*www.facebook.com/csslp*

(ISC)²

which, in turn, calls personToViewModel (see **Figure 1**). Elsewhere in the solution, I've predefined the peopleFeed variable as "http://localhost:43447/DataService.svc/People."

## Binding to HTML Controls

Notice the code in the mapResultsToViewModel method: ko.applyBindings(vm). This is another key aspect of how Knockout works. But what am I applying bindings to? That's what I'll define in my markup. In my markup code, I use the Knockout data-bind attribute to bind the values from my PersonViewModel to some input elements:

```
<body>
  <input data-bind="value: FirstName"></input>
  <input data-bind="value: LastName"></input>
  <input id="save" type="button" value="Save" onclick="save();"></input>
</body>
```

If I only wanted to display the data, I could use label elements and instead of data binding to the value, I could bind to text. For example:

```
<label data-bind="text: FirstName"></label>
```

But I want to edit, so I'm not only using an input element. My Knockout data bind also specifies that I'm binding to the value of these properties.

The key ingredients that Knockout is providing to my solution are the observable properties in my view model, the data-bind attribute for my markup elements, and the applyBindings method that adds the runtime logic necessary for Knockout to notify those elements when a property value changes.

> Thanks to Knockout, when it's time to save I don't have to extract the values from the input elements.

If I run what I have so far in the application, I can see the person returned by the query in debug mode, as shown in **Figure 2**.

**Figure 3** shows the PersonViewModel property values displayed on the page.

## Saving Back to the Database

Thanks to Knockout, when it's time to save I don't have to extract the values from the input elements. Knockout has already updated the PersonViewModel object that was bound to the form. In my save method I'll push the PersonViewModel values back to the person object (that came from the service) and then save those changes back up to the database by way of my service. You'll see in the code download that I kept the person instance that was originally returned from the OData query and I'm using that same object here. Once I update person with the viewModeltoPerson method, I can then pass it into the OData.request as part of a request object, as shown in **Figure 4**. The request object is the first parameter and consists of the URI, the method and the data. Take a look at the datajs documentation at bit.ly/FPTkZ5 to learn more about the request method.

Figure 4 **Saving Changes Back to the Database**

```
function save() {
  viewModeltoPerson(vm, person);

  OData.request(
    {
      requestUri: person.__metadata.uri,
      method: "PUT",
      data: person
    },
    success,
    saveError
    );
  }

  function success(data, response) {
    alert("Saved");
  }

  function saveError(error) {
    alert("Error occurred " + error.message);
  }
}

function viewModeltoPerson(vm,person) {
  person.FirstName = vm.FirstName();
  person.LastName = vm.LastName();
}
```

Notice I'm leveraging the fact that the person instance stored the URI it's bound to in the __metadata.uri property. With that property, I won't have to hardcode the URI, which is "http://localhost:43447/DataService.svc/People(1)."

Now when I modify the data—by changing Julia to Julie, for example—and hit the Save button, not only do I get a "Saved" alert to indicate that no errors occurred, but I can see the database update in my profiler:

```
exec sp_executesql N'update [dbo].[People]
set [FirstName] = @0
where ([PersonId] = @1)
',N'@0 nvarchar(max) ,@1 int',@0=N'Julie',@1=1
```

## Knockout.js and WCF Data Services for the Masses

Exploring Knockout.js pushed me to learn some new tools that can be used by developers of any stripe, not just in the .NET platform. And while it did force me to exercise some rusty JavaScript skills, the code I wrote focused on the familiar task of object manipulation and not on the drudgery of interacting with the controls. It also led me down a path of architectural goodness, using the MVVM approach to differentiate my model objects from those I want to present in my UI. There's certainly a lot more to what you can do with Knockout.js, especially for building responsive Web UIs. You can also use great tools like the WCF Web API (bit.ly/kthOgY) to create your data source. I look forward to learning more from the pros and finding other excuses to work on the client side. ■

**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 experts for reviewing this article: *John Papa and Alejandro Trigo*

# MATCH YOUR SERVER TO YOUR BUSINESS.

## ONLY PAY FOR WHAT YOU NEED!


- Change hardware configurations in real time to meet your business needs.
- Control costs with pay-per-configuration and hourly billing
- Up to 6 Cores, 24 GB RAM, 800 GB storage
- 2000 GB of traffic included free
- Parallels® Plesk Panel 10 for unlimited domains, reseller ready.

**1&1 DYNAMIC CLOUD SERVER**

Starting at

**$49.99**

PER MONTH*

- NEW: Monitor and manage your cloud server through 1&1 mobile apps for Android™ and iPhone®.

MEMBER OF united internet

**www.1and1.com**

1&1®

*Set-up fee of $49 applies. Base configuration includes 1 core, 1 GB Ram, 100 GB Storage. Other terms and conditions may apply. Visit www.1and1.com for full promotional offer details. Program and pricing specifications and availability subject to change without notice. 1&1 and the 1&1 logo are trademarks of 1&1 Internet, all other trademarks are the property of their respective owners. © 2012 1&1 Internet. All rights reserved.

# What's New in the .NET 4.5 Base Class Library

Immo Landwerth

The Microsoft .NET Framework base class library (BCL) is all about fundamentals. Although some of the basic constructs are stable and don't change very much (for example, System.Int32 and System.String), Microsoft is still investing a great deal in this space. This article covers the big (and some of the small) improvements made to the BCL in the .NET Framework 4.5.

When reading this article, please keep in mind that it's based on the .NET Framework 4.5 beta version, not on the final product and APIs, thus features are subject to change.

If you'd like an overview of other areas in the .NET Framework, such as Windows Communication Foundation (WCF) or Windows Presentation Foundation (WPF), take a look at the MSDN Library page, "What's New in the .NET Framework 4.5 Beta" (bit.ly/p6We9u).

## Simplified Asynchronous Programming

Using asynchronous I/O has several advantages. It helps to avoid blocking the UI and it can reduce the number of threads the OS has to use. And yet, chances are you're not taking advantage of it because asynchronous programming used to be quite complex. The biggest problem was that the previous Asynchronous Programming Model (APM) was designed around Begin/End method pairs. To illustrate how this pattern works, consider the following straightforward synchronous method that copies a stream:

```
public void CopyTo(Stream source, Stream destination)
{
  byte[] buffer = new byte[0x1000];
  int numRead;
  while ((numRead = source.Read(buffer, 0, buffer.Length)) != 0)
  {
    destination.Write(buffer, 0, numRead);
  }
}
```

In order to make this method asynchronous using the earlier APM, you'd have to write the code shown in **Figure 1**.

Clearly, the asynchronous version is not nearly as simple to understand as the synchronous counterpart. The intention is hard to distill from the boilerplate code that's required to make basic programming language constructs, such as loops, work when delegates are involved. If you aren't scared yet, try to add exception handling and cancellation.

Fortunately, this release of the BCL brings a new asynchronous programming model that's based around Task and Task<T>. C# and Visual Basic added first-class language support by adding the async and await keywords (by the way, F# already had language support for it via asynchronous workflows and, in fact, served as an inspiration for this feature). As a result, the compilers will now take care of most, if not all, of the boilerplate code you used to have to write. The new language support, as well as certain API additions to the .NET Framework, work together to make writing asynchronous methods virtually as simple as writing synchronous code. See for yourself—to make the CopyTo method asynchronous now requires only the following highlighted changes:

This article is based on the Microsoft .NET Framework 4.5 beta. All information is subject to change.

**This article discusses:**

- Simplified asynchronous programming
- Read-only collection interfaces
- Support for .zip archives
- Miscellaneous improvements to the base class library

**Technologies discussed:**

Visual Studio 11; Microsoft .NET Framework Base Class Library

Telerik RadControls for

# ASP.NET AJAX

All Telerik Developer tools support Visual Studio 2011

## The industry's leading AJAX components


We created RadControls for ASP.NET AJAX with your productivity in mind. We offer a suite of feature-rich controls for every need that will truly save you time, while helping you deliver awesome-looking apps with the performance you desire. You be the judge.

**TRY IT FOR YOURSELF** at www.telerik.com/ajax

Visual Studio Magazine 2011 WINNER

SD Times 2011 100

Members Choice Winner The Code Project 2011

Microsoft Partner Network

2011 PARTNER OF THE YEAR
Mobility
Finalist

telerik
deliver more than expected

```
public async Task CopyToAsync(Stream source, Stream destination)
{
  byte[] buffer = new byte[0x1000];
  int numRead;
  while ((numRead = await source.ReadAsync(buffer, 0, buffer.Length)) != 0)
  {
    await destination.WriteAsync(buffer, 0, numRead);
  }
}
```

There are lots of interesting things to say about asynchronous programming using the new language features, but I'd like to focus on the impact on the BCL and you, the BCL consumer. If you're super curious, though, go on and read the MSDN Library page, "Asynchronous Programming with Async and Await (C# and Visual Basic)" (bit.ly/nXerAc).

To create your own asynchronous operations, you need low-level building blocks. Based on those, you can build more complex methods like the CopyToAsync method shown earlier. That method requires only the ReadAsync and WriteAsync methods of the Stream class as building blocks. **Figure 2** lists some of the most important asynchronous APIs added to the BCL.

Please note that we did not add asynchronous versions of APIs that had a very small granularity, such as TextReader.Peek. The reason is that asynchronous APIs also add some overhead and we wanted to prevent developers from accidentally heading in the wrong direction. This also means we specifically decided against providing asynchronous versions for methods on BinaryReader or BinaryWriter. To use those APIs, we recommend using Task.Run to start a new asynchronous operation and then using synchronous APIs from within that operation—but don't do it per method call. The general guidance is: Try to make your asynchronous operation as chunky as possible. For example, if you want to read 1,000 Int32s from a stream using BinaryReader, it's better to run and await one Task to synchronously read all 1,000 than to individually run and await 1,000 Tasks that each reads only a single Int32.

If you want to learn more about writing asynchronous code, I recommend reading the Parallel Programming with .NET blog (blogs.msdn.com/b/pfxteam).

Figure 1 **Copying a Stream the Old Asynchronous Way**

```
public void CopyToAsyncTheHardWay(Stream source, Stream destination)
{
  byte[] buffer = new byte[0x1000];
  Action<IAsyncResult> readWriteLoop = null;
  readWriteLoop = iar =>
  {

    for (bool isRead = (iar == null); ; isRead = !isRead)
    {
      switch (isRead)
      {
        case true:
          iar = source.BeginRead(buffer, 0, buffer.Length, readResult =>
          {
            if (readResult.CompletedSynchronously) return;
            readWriteLoop(readResult);
          }, null);
          if (!iar.CompletedSynchronously) return;
          break;

        case false:
          int numRead = source.EndRead(iar);
          if (numRead == 0)
          {
            return;
          }
          iar = destination.BeginWrite(buffer, 0, numRead, writeResult =>
          {
            if (writeResult.CompletedSynchronously) return;
            destination.EndWrite(writeResult);
            readWriteLoop(null);
          }, null);
          if (!iar.CompletedSynchronously) return;
          destination.EndWrite(iar);
          break;
        }
      }

  };
  readWriteLoop(null);
}
```

## Read-Only Collection Interfaces

One of the long-standing feature requests for the BCL was read-only collection interfaces (not to be confused with immutable collections; see "Different Notions of Read-Only Collections" on p. 22 for details). Our position has been that this particular aspect (being read-only) is best modeled using the optional feature pattern. In this pattern, an API is provided that allows consumers to test whether a given feature is unsupported and to throw a NotSupportedException.

The benefit of this pattern is that it requires fewer types, because you don't have to model feature combinations. For example, the Stream class provides several features, and all of them are expressed via Boolean get accessors (CanSeek, CanRead, CanWrite and CanTimeout). This allows the BCL to have a single type for streams and yet support every combination of the streaming features.

Over the years, we came to the conclusion that adding a read-only collection interface is worth the added complexity. First, I'd like to show you the interfaces and then discuss the features they offer. **Figure 3** shows a Visual Studio class diagram of the existing (mutable) collection interfaces; **Figure 4** shows the corresponding read-only interfaces.

Note that IReadOnlyCollection<T> did not make it into the .NET Framework 4.5 beta, so don't be surprised if you can't find it. In the beta, IReadOnlyList<T> and IReadOnlyDictionary<TKey, TValue> derive directly from IEnumerable<T> and each interface defines its own Count property.

IEnumerable<T> is covariant. That means if a method accepts an IEnumerable<Shape> you can call it with an IEnumerable<Circle> (assuming Circle is derived from Shape). This is useful in scenarios with type hierarchies and algorithms that can operate on specific types, such as an application that can draw various shapes. IEnumerable<T> is sufficient for most scenarios that deal with collections of types, but sometimes you need more power than it provides:

1. **Materialization.** IEnumerable<T> does not allow you to express whether the collection is already available ("materialized") or whether it's computed every time you iterate over it (for example, if it represents a LINQ query). When an algorithm requires multiple iterations over the collection, this can result in performance degradation if computing the sequence is expensive; it can also cause subtle bugs because of identity mismatches when objects are being generated again on subsequent passes.
2. **Count.** IEnumerable<T> does not provide a count. In fact, it might not even have one, as it could be an infinite sequence. In many cases, the static extension method Enumerable.Count is good enough, though. First, it special-cases known collection types such as ICollection<T> to

# Files *driving* you crazy?

## Get on the right track with Aspose

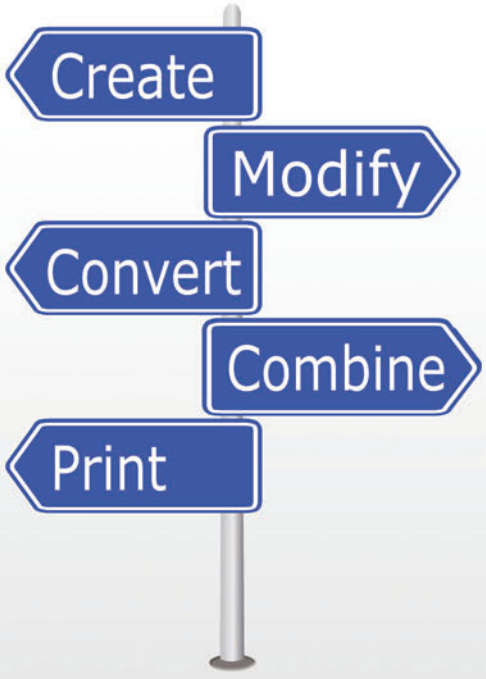
### Aspose.Words
DOC, DOCX, RTF, HTML, PDF,
XPS & other document formats.

### Aspose.Cells
XLS, XLSX, XLSM, XLTX, CSV,
SpreadsheetML & image formats.

### Aspose.Pdf
PDF, XML, XLS-FO, HTML, BMP,
JPG, PNG & other image formats.

### Aspose.Email
MSG, EML, PST, EMLX &
other formats.

*and many more!*




Aspose.BarCode Aspose.Tasks
Aspose.Slides Aspose.Diagram
Aspose.OCR Aspose.Imaging

Scan our QR Code
for an exclusive
20% coupon code.

Follow us on
Facebook & Twitter

ASPOSE
Your File Format Experts

Get your FREE evaluation copy at **http://www.aspose.com**

US Sales: 1.888.277.6734
sales@aspose.com

EU Sales: +44 (0)800 098 8425
sales.europe@aspose.com

avoid iterating the entire sequence; second, in many cases, computing the result is not expensive. However, depending on the size of the collection, this can make a difference.

3. **Indexing.** IEnumerable<T> doesn't allow random access to the items. Some algorithms, such as quick sort, depend on being able to get to an item by its index. Again, there's a static extension method (Enumerable.ElementAt) that uses an optimized code path if the given enumerable is backed by an IList<T>. However, if indexing is used in a loop, a linear scan can have disastrous performance implications, because it will turn your nice O(n) algorithm into an $O(n^2)$ algorithm. So if you need random access, well, you really need random access.

So why not simply use ICollection<T>/IList<T> instead of IEnumerable<T>? Because you lose covariance and because you can no longer differentiate between methods that only read the collections and methods that also modify the collections—something that becomes increasingly important when you use asynchronous programming or multithreading in general. In other words, you want to have your cake and eat it, too.

Enter IReadOnlyCollection<T> and IReadOnlyList<T>. IReadOnlyCollection<T> is basically the same as IEnumerable<T> but it adds a Count property. This allows creating algorithms that can express the need for materialized collections, or collections with a known, finite size. IReadOnlyList<T> expands on that by adding an indexer. Both of these interfaces are covariant, which means if a method accepts an IReadOnlyList<Shape> you can call it with a List<Circle>:

```
class Shape { /*...*/ }
class Circle : Shape { /*...*/ }

void LayoutShapes(IReadOnlyList<Shape> shapes) { /*...*/ }

void LayoutCircles()
{
  List<Circle> circles = GetCircles();
  LayoutShapes(circles);
}
```

Unfortunately, our type system doesn't allow making types of T covariant unless it has no methods that take T as an input. Therefore, we can't add an IndexOf method to IReadOnlyList<T>. We believe this is a small sacrifice compared to not having support for covariance.

All our built-in collection implementations, such as arrays, List<T>, Collection<T> and ReadOnlyCollection<T>, also implement the read-only collection interfaces. Because any collection can now be treated as a read-only collection, algorithms are able to declare their intent more precisely without limiting their level of reuse—they can be used across all collection types. In the preceding example, the consumer of LayoutShapes could pass in a List<Circle>, but LayoutShapes would also accept an array of Circle or a Collection<Circle>.

Another benefit of these collection types is the great experience they provide for working with the Windows Runtime (WinRT). WinRT supplies its own collection types, such as Windows.Foundation.Collections.IIterable<T> and Windows.Foundation.Collections.IVector<T>. The CLR metadata layer projects these directly to the corresponding BCL data types. For example, when consuming WinRT from the .NET Framework, IIterable<T> becomes IEnumerable<T> and IVector<T> becomes IList<T>. In fact, a developer using Visual Studio and IntelliSense couldn't even tell that WinRT has different collection types. Because WinRT also provides read-only versions (such as IVectorView<T>), the new read-only interfaces complete the picture so all collection types can easily be shared between the .NET Framework and WinRT.

Figure 2 **Asynchronous Methods in the BCL**

| Type | Methods |
|---|---|
| System.IO.Stream | ReadAsync<br>WriteAsync<br>FlushAsync<br>CopyToAsync |
| System.IO.TextReader | ReadAsync<br>ReadBlockAsync<br>ReadLineAsync<br>ReadToEndAsync |
| System.IO.TextWriter | WriteAsync<br>WriteLineAsync<br>FlushAsync |

## Support for .zip Archives

Another common request over the years was support for reading and writing regular .zip archives. The .NET Framework 3.0 and later supports reading and writing archives in the Open Packaging Convention (bit.ly/ddsfZ7). However, System.IO.Packaging was tailored to support that particular specification and generally can't be used to process ordinary .zip archives.

This release adds first-class zipping support via System.IO.Compression.ZipArchive. In addition, we've fixed some long-standing issues with performance and compression quality in our DeflateStream implementation. Starting with the .NET Framework 4.5, the DeflateStream class uses the popular zlib library. As a result, it provides a better implementation of the deflate algorithm and, in most cases, a smaller compressed file than in earlier versions of the .NET Framework.

Extracting an entire archive on disk takes only a single line:

```
ZipFile.ExtractToDirectory(@"P:\files.zip", @"P:\files");
```

We've also made sure that typical operations don't require reading the entire archive into memory. For example, extracting a single file from a large archive can be done like this:

```
using (ZipArchive zipArchive = ZipFile.Open(@"P:\files.zip", ZipArchiveMode.Read))
{
  foreach (ZipArchiveEntry entry in zipArchive.Entries)
  {
    if (entry.Name == "file.txt")
    {
      using (Stream stream = entry.Open())
      {
        ProcessFile(stream);
      }
    }
  }
}
```

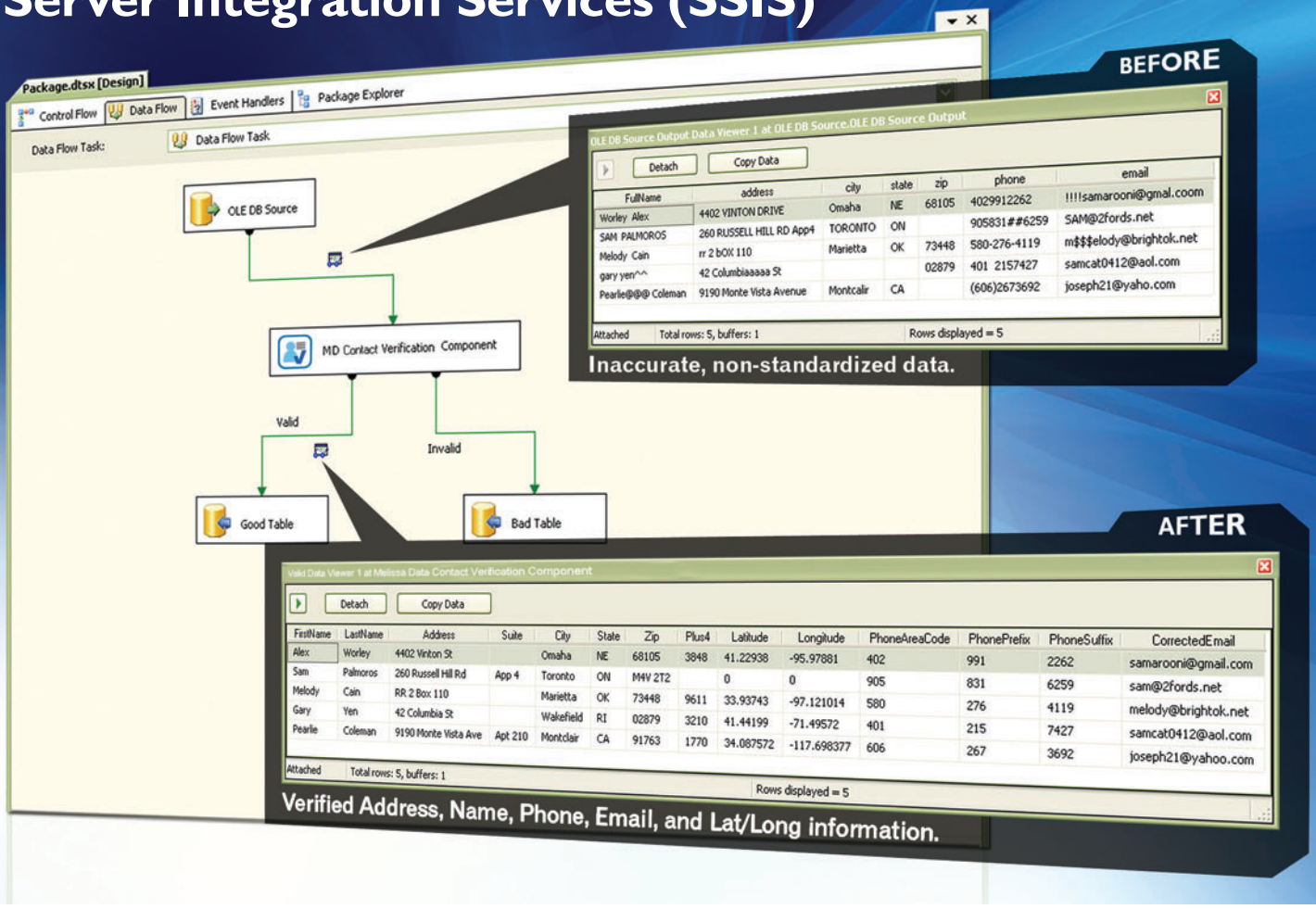
In this case, the only part loaded into memory is the .zip archive's table of contents. The file being extracted is fully streamed; that is, it does not require fitting into the memory. This enables processing arbitrarily large .zip archives, even if memory is limited.

Creating .zip archives works similarly. In order to create a .zip archive from a directory, you need only write a single line:

```
ZipFile.CreateFromDirectory(@"P:\files", @"P:\files.zip");
```

Of course, you can also manually construct a .zip archive, which gives you full control over the internal structure. The following code shows how to create a .zip archive where only C# source code files are added (in addition, the .zip archive now also has a subdirectory called SourceCode):

# Data Cleansing Tools for SQL Server Integration Services (SSIS)




### Contact Verification Component

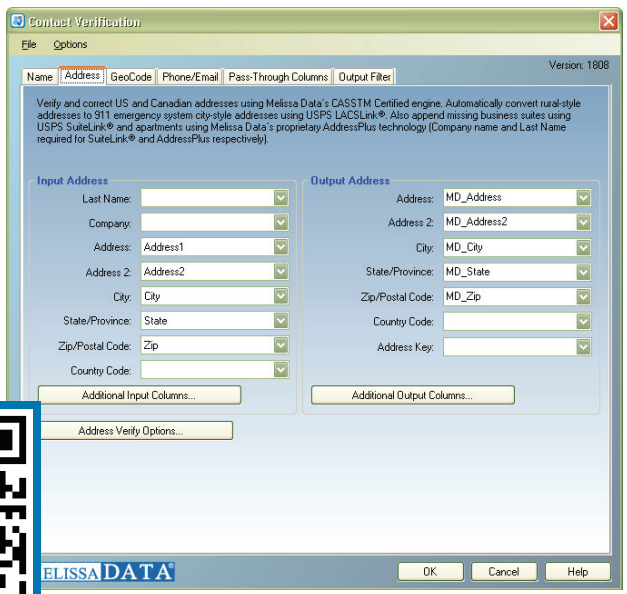
Parse, validate, correct and geocode addresses, phone numbers, email addresses, and full names.

### 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

```
IEnumerable<string> files = Directory.EnumerateFiles(@"P:\files", "*.cs");

using (ZipArchive zipArchive = ZipFile.Open(@"P:\files.zip",
ZipArchiveMode.Create))
{
  foreach (string file in files)
  {
    var entryName = Path.Combine("SourceCode", Path.GetFileName(file));
    zipArchive.CreateEntryFromFile(file, entryName);
  }
}
```

Using streams you can also construct .zip archives where the inputs are not backed by an actual file. The same is true for the .zip archive itself. For example, instead of the ZipFile.Open, you can use the ZipArchive constructor that takes a stream. You could use this, for example, to build a Web server that creates .zip archives on the fly from content stored in a database and that also writes the results directly to the response stream, instead of to the disk.

One detail about the API design is worth explaining a bit more. You've probably noticed that the static convenience methods are defined on the class ZipFile, whereas the actual instance has the type ZipArchive. Why is that?

Starting with the .NET Framework 4.5, we designed the APIs with portability in mind. Some .NET platforms don't support file paths, such as .NET for Metro-style apps on Windows 8. On this platform, the file system access is brokered to enable a capability-based model. In order to read or write files, you can no longer use regular Win32 APIs; instead, you must use WinRT APIs.

As I showed you, .zip functionality itself doesn't require file paths at all. Therefore, we split the functionality into two pieces:

1. **System.IO.Compression.dll.** This assembly contains general-purpose .zip functionality. It doesn't support file paths. The main class in this assembly is ZipArchive.
2. **System.IO.Compression.FileSystem.dll.** This assembly provides a static class ZipFile that defines extension methods and static helpers. Those APIs augment the .zip functionality with convenience methods on .NET platforms that support file system paths.

To learn more about writing portable .NET code, have a look at the Portable Class Libraries page in the MSDN Library at bit.ly/z2r3eM.

## Miscellaneous Improvements

Of course, there's always more to say. After working on a release for so long, naming the most important features sometimes feels like being asked to name your favorite child. In the following, I'd like to highlight a few areas that are worth mentioning but didn't quite get the space they deserve in this article:

**AssemblyMetadataAttribute** One thing we learned in the .NET Framework about attributes is that no matter how many you have, you still need more (by the way, the .NET Framework 4 already had more than 300 assembly-level attributes). AssemblyMetadataAttribute is a general-purpose assembly attribute that allows associating a string-based key-value pair with an assembly. This can be used to point to the homepage of the product or to the version control label that corresponds to the source the assembly was built from.

**WeakReference<T>** The existing non-generic WeakReference type has two issues: First, it forces consumers to cast whenever the target needs to be accessed. More importantly, it has a design flaw that makes using it prone to race conditions: it exposes one API to check whether the object is alive (IsAlive) and a separate API to get to the actual object (Target). WeakReference<T> fixes this issue by providing the single API TryGetTarget, which performs both operations in an atomic fashion.

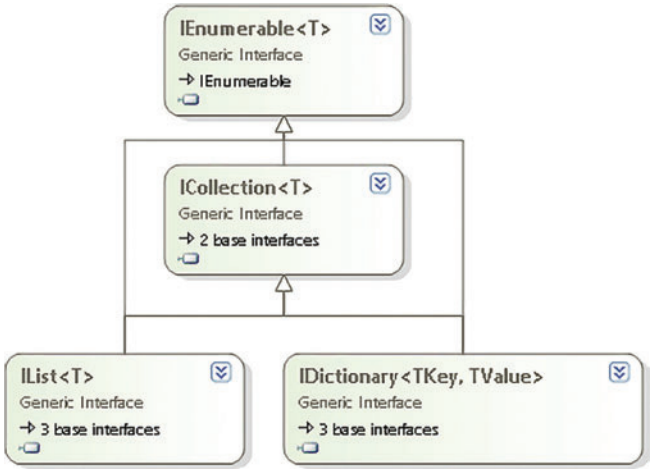



Figure 3 **Mutable Collection Interfaces**

**Comparer<T>.Create(Comparison<T>)** The BCL provides two ways to implement comparers of collections. One is via the interface IComparer<T>; the other is the delegate Comparison<T>. Converting an IComparer<T> to Comparison<T> is simple. Most languages provide an implicit conversion from a method to a delegate type, so you can easily construct the Comparison<T> from the IComparer<T> Compare method. However, the other direction required you to implement an IComparer<T> yourself. In the .NET Framework 4.5 we added a static Create method on Comparer<T> that, given a Comparison<T>, gives you an implementation for IComparer<T>.

**ArraySegment<T>** In the .NET Framework 2.0 we added the struct ArraySegment<T>, which lets you represent a subset of a given

## Different Notions of Read-Only Collections

- **Mutable (or Not Read-Only)** The most common collection type in the .NET world. These are collections such as List<T> that allow reading, as well as adding, removing and changing items.
- **Read-Only** These are collections that can't be modified from the outside. However, this notion of collections doesn't guarantee that its contents will never change. For example, a dictionary's keys and values collections can't be updated directly, but adding to the dictionary indirectly updates the keys and values collections.
- **Immutable** These are collections that, once created, are guaranteed to never be changed. This is a nice property for multithreading. If a complicated data structure is fully immutable, it can always be safely passed to a background worker. You don't need to worry about someone modifying it at the same time. This collection type is not provided by the Microsoft .NET Framework base class library (BCL) today.
- **Freezable** These collections behave like mutable collections until they're frozen. Then they behave like immutable collections. Although the BCL doesn't define these collections, you can find them in Windows Presentation Foundation.

Andrew Arnott has written an excellent blog post that describes the different notions of collections in more detail (bit.ly/pDNNdM).


Reach new heights.
Take your application's scalability to the next level with ScaleOut StateServer®, the industry's leading in-memory data grid.
Announcing Version 5.0
With version 5.0, ScaleOut StateServer again takes the lead in scaling memory-based storage for enterprise applications. Check out its fully parallel LINQ query, intuitive map/reduce analytics engine, seamless global data access, and integrated support for public clouds. Can your .NET distributed cache do all that?
▶ Download a free trial today!
ScaleOut Software
In-Memory Data Grids for the Enterprise
www.scaleoutsoftware.com/evaluate | 503.643.3422

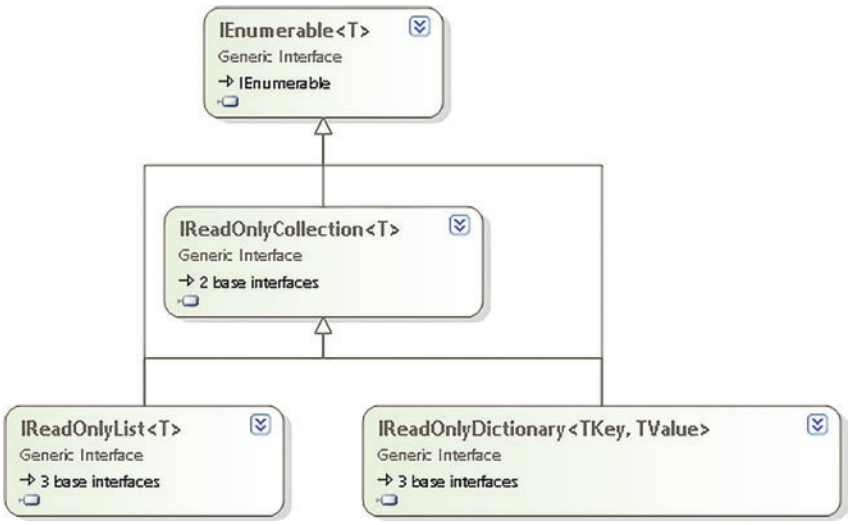



Figure 4 **Read-Only Collection Interfaces**

array without copying. Unfortunately, ArraySegment<T> did not implement any of the collection interfaces, which prevented you from passing it to methods that operate over the collection interfaces. In this release, we fixed that and ArraySegment<T> now implements IEnumerable, IEnumerable<T>, ICollection<T> and IList<T>, as well as IReadOnlyCollection<T> and IReadOnlyList<T>.

**SemaphoreSlim.WaitAsync** This is the only synchronization primitive that supports awaiting the lock to be taken. If you want to learn more about the rationale, read "What's New for Parallelism in .NET 4.5 Beta" (bit.ly/AmAUIF).

**ReadOnlyDictionary<TKey,TValue>** Since the .NET Framework 2.0, the BCL has provided ReadOnlyCollection<T>, which serves as a read-only wrapper around a given collection instance. This enables implementers of object models to expose collections that the consumer can't change. In this release, we added the same concept for dictionaries.

**BinaryReader, BinaryWriter, StreamReader, StreamWriter: Option for not disposing the underlying stream** The higher-level reader and writer classes all accept a stream instance in their constructors. In previous releases, this meant that the ownership of that stream passed on to the reader/writer instance, which implied that disposing the reader/writer also disposed the underlying stream. This is all nice and handy when you have only a single reader/writer, but it makes it harder in cases where you need to compose several different APIs that all operate on streams but need to use readers/writers as part of their implementation. The best you could do in previous releases was not dispose the reader/writer and leave a comment in the source that explained the issue (this approach also forced you to manually flush the writer to avoid data loss). The .NET Framework 4.5 allows you to express this contract by using a reader/writer constructor that takes the parameter leaveOpen where you can explicitly specify that the reader/writer must not dispose the underlying stream.

**Console.IsInputRedirected, Console.IsOutputRedirected and Console.IsErrorRedirected** Command-line programs support redirecting input and output. To most applications, this is transparent. However, sometimes you want a different behavior when redirection is active. For example, colored console output is not useful and setting the cursor position will not succeed. These properties allow querying whether any of the standard streams were redirected.

**Console.OutputEncoding and Console.InputEncoding can now be set to Encoding.Unicode** Setting Console.OutputEncoding to Encoding.Unicode allows the program to write characters that could not be represented in the OEM code page associated with the console. This feature also makes it easier to display text in multiple scripts on the console. More details will be available in the upcoming revised documentation in the MSDN Library for the Console class.

**ExceptionDispatchInfo** Error handling is an important aspect in building framework components. Sometimes re-throwing exceptions (in C# via "throw;") is not sufficient, because it can only happen within an exception handler. Some framework components, such as the Task infrastructure, have to re-throw the exception at a later point (for example, after marshaling back to the originating thread). In the past, that meant the original stack trace and Windows Error Reporting (WER) classification (also known as Watson buckets) was lost, as throwing the same exception object again would simply overwrite this information. ExceptionDispatchInfo allows capturing an existing exception object and throwing it again without losing any of the valuable information recorded in the exception object.

**Regex.Timeout** Regular expressions are a great way to validate input. However, it's not widely known that certain regular expressions can be incredibly expensive to compute when applied to specific text inputs; that is, they have exponential time complexity. This is especially problematic in server environments in which the actual regular expressions are subject to configuration. Because it's hard, if not impossible, to predict the runtime behavior of a given regular expression, the most conservative approach for those cases is to apply a constraint on how long the Regex engine will try to match a given input. For this reason, Regex now has several APIs that accept a timeout: Regex.IsMatch, Regex.Match, Regex.Matches, Regex.Split and Regex.Replace.

## Get Involved

The features discussed in this article are available in the Visual Studio 11 beta. It meets our high standards for pre-release software, so we support it in production environments. You can download the beta from bit.ly/9JWDT9. Because this is prerelease software, we're interested in getting your feedback, whether you've hit any issues (connect.microsoft.com/visualstudio) or you've got ideas for new features (visualstudio.uservoice.com). I'd also suggest subscribing to my team's blog at blogs.msdn.com/b/bclteam so you're informed about any upcoming changes or announcements. ■

**IMMO LANDWERTH** *is a program manager on the CLR team at Microsoft, where he works on the Microsoft .NET Framework base class library (BCL), API design and portable class libraries. You can reach him via the BCL team blog at blogs.msdn.com/b/bclteam.*

**THANKS** to the following technical experts for reviewing this article: *Nicholas Blumhardt, Greg Paperin Daniel Plaisted, Evgeny Roubinchtein, Alok Shriram, Chris Szurgot, Stephen Toub and Mircea Trofin*

# 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

# An Attribute-Free Approach to Configuring MEF

Alok Shriram

The Managed Extensibility Framework (MEF) was designed to give Microsoft .NET Framework developers an easy way to build loosely coupled applications. The primary focus of MEF in version 1 was extensibility, such that an application developer could expose certain extension points to third-party developers, and the third-party developers could build add-ons or extensions to those components. The Visual Studio plug-in model for extending Visual Studio itself is an excellent use case of this, which you can read about on the MSDN Library page, "Developing Visual Studio Extensions" (bit.ly/IkJQsZ). This method of exposing extension points and defining plug-ins uses what's known as an attributed programming model, in which a developer can decorate properties, classes and even methods with attributes to advertise either a requirement for a dependency of a specific type or the ability to satisfy the dependency of a specific type.

Despite the fact that attributes are very useful for extensibility scenarios where the type system is open, they're overkill for closed-type systems that are known at build time. Some fundamental issues with the attributed programming model are:

1. Configuration for many similar parts involves a lot of needless repetition; this is a violation of the Don't Repeat Yourself (DRY) principle, and in practice can lead to human error and source files that are more difficult to read.
2. Authoring an extension or part in the .NET Framework 4 means dependence on MEF assemblies, which ties the developer to a specific dependency injection (DI) framework.
3. Parts that were not designed with MEF in mind need to have the attributes added to them in order to be appropriately identified in applications. This can serve as a significant barrier to adoption.

The .NET Framework 4.5 provides a way to centralize configuration so that a set of rules can be written on how extension points and components are created and composed. This is achieved using a new class called RegistrationBuilder (bit.ly/HsCLrG), which can be found in the System.ComponentModel.Composition.Registration namespace. In this article I'll first consider some reasons for using a system such as MEF. If you're an MEF veteran, you might be able to

This article discusses a prerelease version of the Managed Extensibility Framework. All information is subject to change.

**This article discusses:**

- Issues with the attributed programming model
- Dependency injection
- Using convention-driven configuration for the Managed Extensibility Framework
- Working with collections
- Adding metadata to parts

**Technologies discussed:**

Managed Extensibility Framework, Microsoft .NET Framework

Figure 1 **WeatherServiceView—the Results Display Class**

```
[Export]
public class WeatherServiceView
{
  private IWeatherServiceProvider _provider;
  [ImportingConstructor]
  public WeatherServiceView(IWeatherServiceProvider providers)
  {
    _providers = providers;
  }

  public void GetWeatherForecast(int zipCode)
  {
    var result=_provider.GetWeatherForecast(zipCode);
      // Some display logic
  }
}
```

skip this part. Next, I'll don the role of a developer who's been given a set of requirements and create a simple console application using the MEF attributed programming model. I'll then convert this application to the convention-based model, showing how to implement some typical scenarios using RegistrationBuilder. Finally, I'll discuss how convention-driven configuration is already being incorporated into application models and how it makes using MEF and DI principles out of the box a trivial task.

## Background

As software projects grow in size and scale, maintainability, extensibility and testability become key concerns. As software projects mature, components might need to be replaced or refined. As projects grow in scope, requirements often change or get added. The ability to add functionality to a large project in a simple manner is extremely critical to the evolution of that product. Moreover, with change being the norm during most software cycles, the ability to rapidly test components that are a part of a software product, independently of other components, is crucial—especially in environments where dependent components are developed in parallel.

With these driving forces, the notion of DI has become popular in large-scale software development projects. The idea behind DI is to develop components that advertise the dependencies they need without actually instantiating them, as well as the dependencies they satisfy, and the dependency-injection framework will figure out and "inject" the correct instances of the dependencies into the component. "Dependency Injection," from the September 2005 issue of *MSDN Magazine* (msdn.microsoft.com/magazine/cc163739), is an excellent resource if you need more background information.

Figure 2 **IWeatherServiceProvider (WeatherUnderground) Data Parsing Service**

```
[Export(typeof(IWeatherServiceProvider))]
class WeatherUndergroundServiceProvider:IWeatherServiceProvider
{
  private IDataSource _source;

  [ImportingConstructor]
  public WeatherUndergroundServiceProvider(IDataSource source)
  {
    _source = source;
  }

  public string GetWeatherForecast(int zipCode)
  {

    string val = _source.GetData(GetResourcePath(zipCode));
      // Some parsing logic here
    return result;
  }

  private string GetResourcePath(int zipCode)
  {
    // Some logic to get the resource location
  }
}
```

## The Scenario

Now let's get to the scenario I described earlier: I'm a developer looking at a specification I've been given. At a high level, the goal of the solution that I'm going to implement is to provide a weather forecast to a user based on his ZIP code. Here are the required steps:

1. The application requests a ZIP code from the user.
2. The user enters a valid ZIP code.
3. The application contacts an Internet weather service provider for forecast information.
4. The application presents formatted forecast information to the user.

From a requirements perspective, it's clear there are some unknowns at this point, or aspects that have the potential to change later on in the cycle. For example, I don't yet know which weather service provider I'm going to use, or what method I'll employ for getting data from the provider. So to start designing this application, I'll break the product into a couple of discrete functional units: WeatherServiceView, IWeatherServiceProvider and IDataSource. The code for each of these classes is shown in **Figure 1**, **Figure 2** and **Figure 3**, respectively.

Finally, in order to create this hierarchy of parts, I need to use a Catalog from which I can discover all the parts in the application, and then use the CompositionContainer to get an instance of WeatherServiceView, on which I can then operate, like so:

```
class Program
{
  static void Main(string[] args)
  {

    AssemblyCatalog cat = new AssemblyCatalog(typeof(Program).Assembly);
    CompositionContainer container = new CompositionContainer(cat);
    WeatherServiceView forecaster =
                        container.GetExportedValue<WeatherServiceView>();
    // Accept a ZIP code and call the viewer
    forecaster.GetWeatherForecast(zipCode);
  }
}
```

Figure 3 **IDataSource (WeatherFileSource)**

```
[Export(typeof(IDataSource))]
class WeatherFileSource :IDataSource
{
  public string GetData(string resourceLocation)
  {
    Console.WriteLine("Opened ----> File Weather Source ");
    StringBuilder builder = new StringBuilder();
    using (var reader = new StreamReader(resourceLocation))
    {
      string line;
      while((line=reader.ReadLine())!=null)
      {
        builder.Append(line);
      }
    }
    return builder.ToString();
  }
}
```

Figure 4 **WeatherUnderground Data Parsing Class Converted into Simple C# Class**

```
class WeatherUndergroundServiceProvider:IWeatherServiceProvider
{
  private IDataSource _source;

  public WeatherUndergroundServiceProvider(IDataSource source)
  {
    _source = source;
  }

  public string GetWeatherForecast(int zipCode)
  {

    string val = _source.GetData(GetResourcePath(zipCode));
    // Some parsing logic here
    return result;
  }
      ...
}
```

Figure 5 **Setting up Conventions**

```
1.  RegistrationBuilder builder = new RegistrationBuilder();
2.    builder.ForType<WeatherServiceView>()
3.      .Export()
4.      .SelectConstructor(cinfos => cinfos[0]);
5.    builder.ForTypesDerivedFrom<IWeatherServiceProvider>()
6.      .Export<IWeatherServiceProvider>()
7.      .SelectConstructor(cinfo => cinfo[0]);
8.    builder.ForTypesDerivedFrom<IDataSource>()
9.      .Export<IDataSource>();
```

All of the code I've presented so far is fairly basic MEF semantics; if you're unclear about how any of this works, please take a look at the "Managed Extensibility Framework Overview" MSDN Library page at bit.ly/JLJI8y, which details the MEF attributed programming model.

## Convention-Driven Configuration

Now that I have the attributed version of my code working, I want to demonstrate how you convert these pieces of code to the convention-driven model using RegistrationBuilder. Let's start by removing all the classes to which MEF attributes were added. As an example, take a look at the code fragment in **Figure 4**, modified from the WeatherUnderground data parsing service shown in **Figure 2**.

The code in **Figure 1** and **Figure 3** will change in the same way as in **Figure 4**.

Next, I use RegistrationBuilder to define certain conventions in order to express what we specified using the attributes. **Figure 5** shows the code that does this.

Each declaration of a rule has two distinct parts. One part identifies a class or a set of classes to be operated on; the other specifies the attributes, metadata and sharing policies to apply to the selected classes, properties of the classes or constructors of the classes. Thus you can see that lines 2, 5 and 8 start the three rules I'm defining, and the first part of each rule identifies the type on which the rest of the rule is going to be applied. In line 5, for instance, I want to apply a convention for all types that are derived from IWeatherServiceProvider.

Now let's take a look at the rules and map them back to the original attributed code in **Figures 1**, **2** and **3**. WeatherFileSource (**Figure 3**) was simply exported as IDataSource. In **Figure 5**, the rule in lines 8 and 9 specifies picking all the types that are derived from IDataSource and exporting them as contracts of IDataSource. In **Figure 2**, observe that the code exports type IWeatherServiceProvider and requires an import of IDataSource in its constructor, which was decorated with an ImportingConstructor attribute. The corresponding rule for this in **Figure 5** is specified in lines 5, 6 and 7. The added piece here is the method SelectConstructor, which accepts a Func<ConstructorInfo[], ConstructorInfo>. This gives me a way to specify a constructor. You can specify a convention, such as the constructor with the smallest or largest number of arguments will always be the ImportingConstructor. In my example, because I have only one constructor, I can use the trivial case of picking the first and only constructor. For the code in **Figure 1**, the rule in **Figure 5** is specified in lines 2, 3 and 4, and is similar to the one just discussed.

With the rules established, I need to apply them to the types present in the application. To do this, all catalogs now have an overload that accepts a RegistrationBuilder as a parameter. So you'd modify the previous CompositionContainer code as shown in **Figure 6**.

## Collections

Now I'm done and my simple MEF application is up and running without attributes. If only life were so simple! I'm now told my app needs to be able to support more than one weather service and that it needs to display forecasts from all the weather services. Luckily, because I used MEF, I don't have to panic. This is simply a scenario with multiple implementers of an interface, and I need to iterate through them. My example now has more than one implementation of IWeatherServiceProvider and I want to display the results from all of these weather engines. Let's take a look at the changes I need to make, as shown in **Figure 7**.

Figure 6 **Consuming the Conventions**

```
class Program
{
  static void Main(string[] args)
  {
    // Put the code to build the RegistrationBuilder here
    AssemblyCatalog cat = new AssemblyCatalog(typeof(Program).Assembly,builder);
    CompositionContainer container = new CompositionContainer(cat);
    WeatherServiceView forecaster =
                      container.GetExportedValue<WeatherServiceView>();
    // Accept a ZIP code and call the viewer
    forecaster.GetWeatherForecast(zipCode);
  }
}
```

Figure 7 **Enabling Multiple IWeatherServiceProviders**

```
public class WeatherServiceView
{
  private IEnumerable<IWeatherServiceProvider> _providers;
  public WeatherServiceView(IEnumerable<IWeatherServiceProvider> providers)
  {
    _providers = providers;
  }

  public void GetWeatherForecast(int zipCode)
  {
    foreach (var _provider in _providers)
    {
      Console.WriteLine("Weather Forecast");
      Console.WriteLine(_provider.GetWeatherForecast(zipCode));
    }
    }
}
```


# Aspose.Total just got BIGGER

## Aspose.Diagram

Working with Visio files? Easily create, modify and convert diagrams in your applications.

**Supported Files**

VSD VTX
VSS VDW
VST VDX
VSX

NEW

## Aspose.OCR

Extract text from images. Supports popular fonts and styles. Scan a whole image or part of an image.

**Supported Files**

BMP
TIFF

NEW

## Aspose.Imaging

Add advanced drawing features to your applications, plus support for PSD files.

**Supported Files**

PSD BMP
TIFF PNG
JPEG
GIF

NEW

## Already own Aspose.Total for .NET?

### These are yours for FREE!

Free Evaluations at www.aspose.com

US Sales: 1.888.277.6734
sales@aspose.com
EU Sales: +44 (0)800 098 8425
sales.europe@aspose.com

ASPOSE
Your File Format Experts

Figure 8 **Resource Description Definition**

```
public class ResourceInformation
{
  public string Google_Web_ResourceString
  {
    get { return "http://www.google.com/ig/api?weather="; }

  }

  public string Google_File_ResourceString
  {
    get { return @".\GoogleWeather.txt"; }
  }
  public string WeatherUnderground_Web_ResourceString
  {
    get { return
      "http://api.wunderground.com/api/96863c0d67baa805/conditions/q/"; }
  }
}
```

Figure 9 **The Rule for Exporting Properties and Adding Metadata**

```
1.  builder.ForType<ResourceInformation>()
2.       .ExportProperties(pinfo => pinfo.Name.Contains("ResourceString"),
3.    (pinfo, eb) =>
4.      {
5.        eb.AsContractName("ResourceInfo");
6.        string[] arr = pinfo.Name.Split(new char[] { '_' },
7.          StringSplitOptions.RemoveEmptyEntries);
8.        eb.AddMetadata("ResourceAffiliation", arr[0]);
9.        eb.AddMetadata("ResourceLocation", arr[1]);
10.     });
```

That's it! I changed the WeatherServiceView class to accept one or more IWeatherServiceProvider implementations and in the logic section I iterated through that collection. The conventions I established earlier will now capture all the implementations of IWeatherServiceProvider and export them. However, something seems to be missing in my convention: At no point did I have to add an ImportMany attribute or equivalent convention when I was configuring the WeatherServiceView. This is a little bit of RegistrationBuilder magic in which it figures out that if your parameter has an IEnumerable<T> on it, it must be an ImportMany, without you having to explicitly specify it. So using MEF made the job of extending my application simple, and, by using RegistrationBuilder, as long as the new version implemented IWeaterServiceProvider, I didn't have to do anything to make it work with my app. Beautiful!

## Metadata

Another really useful feature in MEF is the ability to add metadata to parts. Let's assume for the sake of the discussion that in the example we've been looking at, the value returned by the GetResourcePath method (shown in **Figure 2**) is governed by the concrete type of IDataSource and IWeatherServiceProvider being used. So I define a naming convention specifying that a resource will be named as an underscore-delimited ("_") combination of the weather service provider and the data source. With this convention, the Weather Underground services provider with a Web data source will have the name WeatherUnderground_Web_ResourceString. The code for this is shown in **Figure 8**.

Using this naming convention I can now create a property in the WeatherUnderground and Google weather service providers that will import all these resource strings and, based on their current configurations, pick the appropriate one. Let's look first at how to write the RegistrationBuilder rule to configure the ResourceInformation as an Export (see **Figure 9**).

Line 1 simply identifies the class. Line 2 defines a predicate that picks all the properties in this class that contain ResourceString, which is what my convention dictated. The last argument of ExportProperties is an Action<PropertyInfo,ExportBuilder>, in which I specify that I want to export all properties that match the predicate specified in line 2 as a named contract called ResourceInfo, and want to add metadata based on parsing the name of that property using the keys ResourceAffiliation and ResourceLocation. On the consuming side, I now need to add a property to all implementations of IWeatherServiceProvider, like this:

```
public IEnumerable<Lazy<string, IServiceDescription>> WeatherDataSources
{ get; set; }
```

And then add the following interface to use strongly typed metadata:

```
public interface IServiceDescription
  {
    string ResourceAffiliation { get; }
    string ResourceLocation { get; }
  }
```

To learn more about metadata and strongly typed metatdata, you can read the helpful tutorial at bit.ly/HAOwwW.

Now let's add a rule in RegistrationBuilder to import all parts that have the contract name ResourceInfo. To do this, I'll take the existing rule from **Figure 5** (lines 5-7) and add the following clause:

```
5.  builder.ForTypesDerivedFrom<IWeatherServiceProvider>()
6.          .Export<IWeatherServiceProvider>()
7.          .SelectConstructor(cinfo => cinfo[0]);
8.          .ImportProperties<string>(pinfo => true,
9.                                    (pinfo, ib) =>
10.                                     ib.AsContractName("ResourceInfo"))
```

Lines 8 and 9 now specify that all types derived from IWeatherServiceProvider should have an Import applied on all properties of type string, and the import should be made on the contract name ResourceInfo. When this rule runs, the previously added property becomes an Import for all contracts with the name ResourceInfo. I can then query the enumeration to filter out the correct resource string, based on metadata.

Figure 10 **Overriding Conventions Using Attributes**

```
public class ResourceInformation
{
  public string Google_Web_ResourceString
  {
    get { return "http://www.google.com/ig/api?weather="; }

  }

  public string Google_File_ResourceString
  {
    get  { return @".\GoogleWeather.txt"; }
  }

  public string WeatherUnderground_Web_ResourceString
  {
    get { return "http://api.wunderground.com/api/96863c0d67baa805/conditions/q/"; }
  }

  [Export("ResourceInfo")]
  [ExportMetadata("ResourceAffiliation", "WeatherUnderground")]
  [ExportMetadata("ResourceLocation", "File")]
  public string WunderGround_File_ResourceString
  {
    get { return @".\Wunder.txt"; }
  }
}
```

## The End of Attributes?

If you consider the samples I've discussed, you'll see that we don't really seem to require attributes anymore. Anything you could do with the attributed programming model can now be achieved using the convention-based model. I mentioned some common use cases where RegistrationBuilder can help, and Nicholas Blumhardt's great write-up on RegistrationBuilder at bit.ly/tVQA1J can give you more information. However, attributes can still play a key role in a convention-driven MEF world. One significant issue with conventions is that they're great only as long as they're followed. As soon as an exception to the rule occurs, the overhead of maintaining the conventions can get prohibitively expensive, but attributes can help with overriding conventions. Let's assume that a new resource was added in the ResourceInformation class, but its name did not adhere to the convention, as shown in **Figure 10**.

In **Figure 10**, you can see that the first part of the convention is incorrect, according to the naming specification. However, by going in and explicitly adding a contract name and metadata that is correct, you can override or add to the parts discovered by RegistrationBuilder, thus making MEF attributes an effective tool for specifying exceptions to the conventions defined by RegistrationBuilder.

## Seamless Development

In this article I've looked at convention-driven configuration, a new feature in MEF exposed in the RegistrationBuilder class that greatly streamlines development associated with MEF. You'll find beta versions of these libraries at mef.codeplex.com. If you don't yet have the .NET Framework 4.5, you can go to the CodePlex site and download the bits.

Ironically, RegistrationBuilder can make your day-to-day development activities revolve less around MEF, and your use of MEF in your projects extremely seamless. A great example of this is the integration package built into Model-View-Controller (MVC) for MEF, which you can read about on the BCL Team Blog at bit.ly/ysWbdL. The short version is that you can download a package into your MVC application, and this sets up your project to use MEF. The experience is that whatever code you have "just works" and, as you start following the convention specified, you get the benefits of using MEF in your application without having to write a line of MEF code yourself. You can find out more about this on the BCL Team blog at bit.ly/ukksfe. ■

**ALOK SHRIRAM** *is a program manager for the Microsoft .NET Framework team at Microsoft, where he works in the Base Class Libraries team. Before that he worked as a developer on the Office Live team, which later became the Office 365 team. After a stint of grad school at the University of North Carolina, Chapel Hill, he's currently based in Seattle. In his spare time he likes to explore all that the Pacific Northwest has to offer with his wife Mangal. He can be found lurking on the MEF CodePlex site, on Twitter at twitter.com/alokshriram and occasionally posting on the .NET blog.*

**THANKS** to the following technical experts for reviewing this article: *Glenn Block, Nicholas Blumhardt and Immo Landwerth*

# Using Windows Azure Service Bus for … Things!

Clemens Vasters

When you stroll (or browse) through a well-stocked electronics store these days, you'll find an amazing array of "things" that have the ability to connect to a network. Don't just think phones, tablets, notebooks or desktops, or just TVs, Blu-ray players and set-top boxes. Think espresso makers, refrigerators and picture frames. Think garage door openers, air conditioning and alarm systems. If you look around and behind the cover panels in industrial or commercial environments such as your own office building, you'll find temperature and humidity sensors, motion sensors, surveillance cameras and a multitude of other kinds of sensors or control switches inside equipment.

Many of these devices generate useful data: temperature readings; number of cups of brewed coffee and how the grinder has been set; infrared images showing that no one is in the conference room and therefore the lights can be turned off.

It's easy to imagine a scenario where you'd like to upload some data into a "thing" as well, such as pushing the latest pictures of your children (or your pet) to a picture frame sitting on grandma's sideboard; or one where you want to flip a switch from a distance—even if that distance only means your phone connected via 3G/4G mobile carrier network—to turn the temperature in the house up a notch. From a networking perspective that's three worlds away, but from the consumer perspective there's no appreciable difference between flipping the switch at home or while sitting in a cab on the way back from the airport returning from a two-week vacation.

It's easy to imagine a scenario where you'd like to upload some data into a "thing."

The opportunities around connected devices are enormous. Supplying services for special-purpose devices might indeed provide more monetization potential for forward-looking cloud developers than apps on general-purpose screen devices tailored for human interaction, such as phones, tablets or the many PC form factors. This seems especially true when you combine such services with cloud technologies emerging around "big data" analysis.

This article discusses:

- Building a solution to monitor and control connected devices
- Storing, retrieving and aggregating events
- Command fan-out, targeting and collecting responses
- Namespace structure, provisioning and device identities

Technologies discussed:

Windows Azure Service Bus

## The Scenario

For the purpose of the following architecture discussion, let's imagine the offering is a Software as a Service (SaaS) for air conditioners.

While the scenario is fictitious and so are all the numbers, the patterns and magnitudes are fairly close to actual scenarios that the Windows Azure team is discussing with partners and customers.

What's nice about air conditioners—from a business perspective—is that there's healthy demand, and global climate trends indicate they won't be going out of fashion anytime soon. Less nice is that they're enormously hungry for electricity and can overload the electrical grid in hotter regions, resulting in rolling brownouts.

The SaaS solution, for which I'll outline the architecture, targets electricity companies looking for analytic insight into air conditioner use for the purpose of capacity management and for a mechanism that allows them to make broad emergency adjustments to air conditioning systems hanging on their electricity grid when the grid is at the verge of collapse.

The bet is that utility customers would prefer their room temperatures forcibly adjusted upward to a cozy 80° F/27° C, rather than having the power grid cut out, leaving them with no defense against the scorching 100° F/38° C outside temperatures.

Let's further assume the SaaS will be paired with a number of air conditioner manufacturers to integrate the required hardware and protocols. Once the devices are installed and connected to the service, there will be opportunities to sell electricity-company- or manufacturer-branded companion apps through mobile app stores that allow the customers to monitor and control their air conditioners from their phones or tablets. **Figure 1** shows an overview of the scenario.

As you're architecting the solution, you need to accommodate two general directions of message traffic. Inbound (from the cloud perspective), you need to collect metrics and events from the devices, aggregate the data and flow it toward the analysis system. Outbound, you need to be able to send control commands to the devices.

Each air conditioning device should be able to send humidity, temperature and device status readings ranging from once per hour to once every 10 minutes as required. Device status changes should also cause events to be raised and sent. Control commands will be much rarer, probably not more than one or two events per day and device.

For the initial ramp-up, set a target scale of 250,000 devices in the first year, accelerating to 2.5 million devices after the third year. At that scale, you can expect 0.25 to 1.5 million events per hour for the first year and around 2.5 to 15 million events per hour by the third year.

## Architecting the Solution

You need to address three major areas of concern in the architecture: provisioning, event fan-in and command fan-out.

**Provisioning** Relates to setting up new devices, assigning them a unique identity within the system and giving each device a way to prove that identity. Furthermore, the devices must be associated with particular resources in the system and must receive a configuration document containing all of this information.

**Event Fan-In** Involves designing the system so it can handle the desired throughput for event ingestion. 15 million events per hour translates (generously rounded) to some 4,200 events per second, which justifies thinking hard about how to partition the system. This is especially the case when those events originate from 4,200 distinct sources with each client connecting remotely, establishing a fresh session with potentially significant network latency.

Because the events and resulting statistics from each particular housing unit matter not only at the macro level, but also to the residents who purchase the mobile application and want to have accurate trend graphs, you not only have to flow rolled-up data for analysis but also figure out how to retain 360 million events per day.

**Command Fan-Out** Deals with the flow of commands to the devices. Commands might instruct a device to change its configuration or they might be status inquiries that tell the device to emit a status event. In cases where the electricity company needs to make adjustments across the board to alleviate pressure on the grid, they might want to send a single command to all devices at once and as fast as possible. In the mobile phone case, where a resident wants to adjust the temperature for comfort, the command is targeted at a particular device or all devices in a housing unit.

You need to address three major areas of concern in the architecture: provisioning, event fan-in and command fan-out.

## Event Fan-In

**Figure 2** illustrates the general layout of the fan-in model for the air conditioning scenario.

The first and most obvious aspect of the architecture is the presence of partitions. Instead of looking at a population of millions of connected devices as a whole, you're subdividing the device population in much smaller and therefore more manageable partitions of several thousand devices each. But manageability isn't the only reason for introducing partitions.

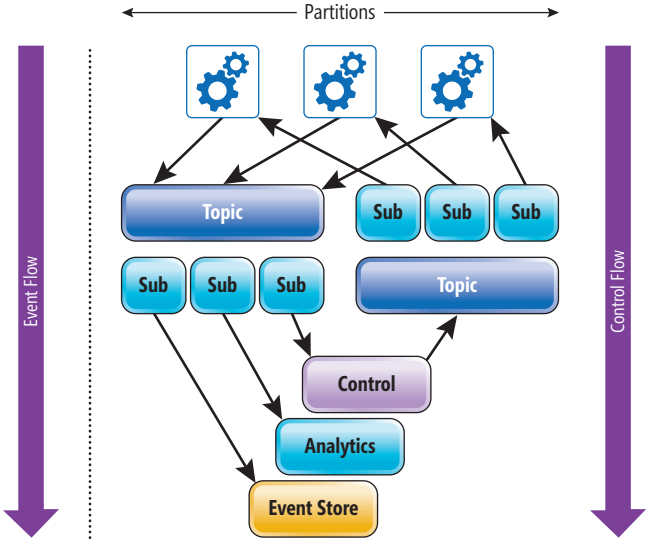



Figure 1 **Overview of the Software as a Service Solution**

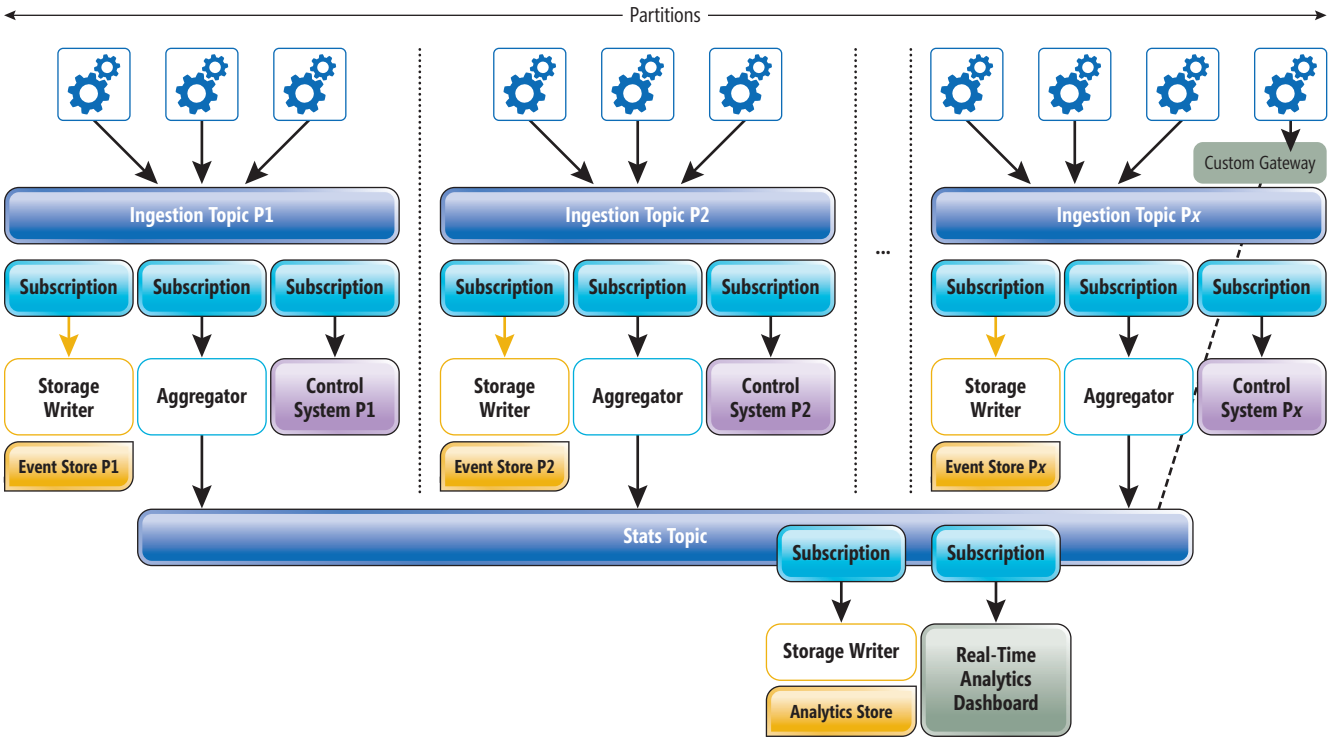



Figure 2 **The Fan-in Model**

First, each resource in a distributed system has a throughput- and storage-capacity ceiling. Therefore, you want to limit the number of devices associated with any single Service Bus Topic such that the events sent by the devices won't exceed the Topic's throughput capacity, and any message backlog that might temporarily build up doesn't exceed the Topic's storage capacity. Second, you want to allocate appropriate compute resources and you also don't want to overtax the storage back end with too many writes. As a result, you should bundle a relatively small set of resources with reasonably well-known performance characteristics into an autonomous and mostly isolated "scale-unit."

Each scale-unit supports a maximum number of devices—and this is important for limiting risks in a scalability ramp-up. A nice side effect of introducing scale-units is that they significantly reduce the risk of full system outages. If a system depends on a single data store and that store has availability issues, the whole system is affected. But if, instead, the system consists of 10 scale-units that each maintain an independent store, a hiccup in one store affects only 10 percent of the system.

The architecture decision shown in **Figure 2** is that the devices drop all events into Windows Azure Service Bus Topics instead of into a service edge that the solution provides directly. Windows Azure Service Bus already provides a scaled-out and secure network service gateway for messaging, and it makes sense to leverage that functionality whenever possible.

For this particular scenario you'll assume the devices are capable of supporting HTTPS and therefore can talk directly to Windows Azure Service Bus with its existing protocol support. However, as the devices get cheaper and smaller and have only a few kilobytes of memory and a slow processor, supporting SSL—and therefore HTTPS—can become a significant challenge, and even HTTP can start to appear enormously heavy. These are cases where building and running a custom network gateway (see the far right of **Figure 2**) with a custom or vertical-industry protocol is a good idea.

Each ingestion Topic is configured with three subscriptions. The first subscription is for a storage writer that writes the received event into an event store; the second subscription is for an aggregation component that rolls up events and forwards them into the global statistics system; and the third subscription is for routing messages to the device control system.

> A nice side effect of introducing scale-units is that they significantly reduce the risk of full system outages.

As a throughput ceiling for each of these Topics, assume an average of 100 messages per second at the entity level. Given you're creating and routing three copies of every submitted event, you can ingest at most some 33 messages per second into each Topic. That's obviously a (shockingly) defensive number. The reason to be very cautious is that you're dealing with distributed devices that send messages every hour, and it would be somewhat naïve to assume perfect, random distribution of event submissions across any given hour. If you assume the worst-case scenario of a 10-minute event interval with one extra control interaction feedback message per device and hour, you

# DevPartner

www.devpartner.com

**Start your Free Trial NOW!**

Built up from the legendary BoundsChecker, TrueTime, and TrueCoverage tools, today DevPartner VC++ Edition - BoundsChecker Suite offers durable troubleshooting capabilities for modern day Visual C++ development.

- Isolate and trap your most costly and effort cosuming bugs.
- Perform runtime inspections with unprecedented scalability.
- Handle your 64-bit processes on Windows desktop and server platforms.

**BoundsChecker**
**TrueTime**
**TrueCoverage**
**Forever True**

"We embed BoundsChecker 10.6 into our products and processes and don't let our products out of the door without being checked."

*- VP and CTO, Synergex*

**US Toll Free 877.772.4450** **www.devpartner.com**

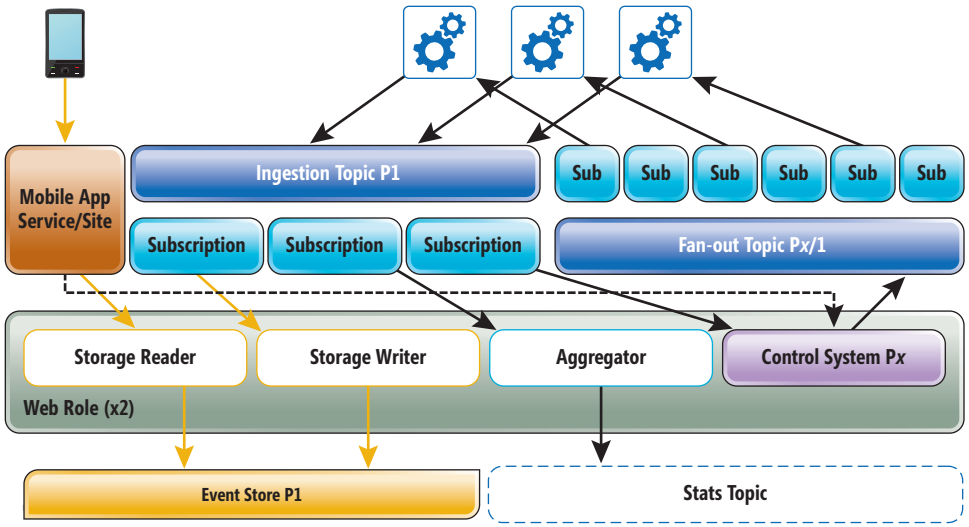



Figure 3 **A Scale-Unit**

can expect seven messages per hour from each device and, therefore, rounding down, you can hook some 50,000 devices onto each Topic.

While this flow rate is nothing to worry about for storage throughput, storage capacity and the manageability of the event store are concerns. The per-device event data at a resolution of one hour for 50,000 devices amounts to some 438 million event records per year. Even if you're very greedy about how to pack those records and use, say, only 50 bytes per record, you'll still be looking at 22GB of payload data per year for each of the scale-units. That's still manageable, but it also underlines that you need to keep an eye on the storage capacity and storage growth as you think about sizing scale-units.

> For storing events, Windows Azure Table Storage is a great candidate.

Based on the one-year ramp-up projection, you'll need five of these scale-units for the first year and 50 after the third.

## Storing, Retrieving and Aggregating Events

Now let's look in more detail at how the incoming events will be handled and used. To illustrate that, I'm zooming in on one of the scale-units in **Figure 3**.

The messages originating from the devices can be split into two broad categories: control replies and events. The control replies are filtered by the subscription for the control system. The events, containing the current device status and sensor readings, are routed to the storage writer and to the aggregator via separate subscriptions.

The storage writer's job is to receive an event from the subscription and write it into the event store.

For storing events, Windows Azure Table Storage is a great candidate. To support the partitioning model, a storage account will be shared across all partitions in the system and have one table per partition. The table store requires a partition key to allocate data rows to a storage section, and you'll use the device's identifier for that key, which will yield nice lookup performance when you need to serve up historical stats to plot graphs. For the row key, simply use a string-encoded timestamp in a compact YYYYMMDDHHMMSS format that yields chronological sorting and allows for range queries for plotting graphs. Also, because individual devices will, in this case, never send events in intervals of less than a second, you don't need to worry about potential collisions. The timestamp will be derived from the message's EnqueuedTimeUtc property that the Windows Azure Service Bus broker automatically stamps on each incoming message, so you don't have to trust the device's clock. If you were building a high-throughput solution where collisions were possible, you could additionally leverage the message's SequenceNumber property, which is a unique, sequential 64-bit identifier the broker stamps on each message.

The aggregator's job is to take incoming events and roll them up so that they can be forwarded into one of the system-wide statistics Topics that feed the analytics infrastructure. This is often done by computing averages or sums across a set of events for a particular duration, and forwarding only the computed values. In this case, you might be interested in trends on energy and temperature readings on a per-neighborhood basis with a 15-minute resolution. Thus the aggregator, running across two Web roles that each see roughly half of the data, will maintain a map of devices to houses and neighborhoods, aggregate data into per-neighborhood buckets as device events arrive, and emit an event with the computed aggregates every 15 minutes. Because each aggregator instance gets to see only half of the events, the analytics back end behind the statistics Topic might have to do a final rollup of the two events to get the full picture for the neighborhood.

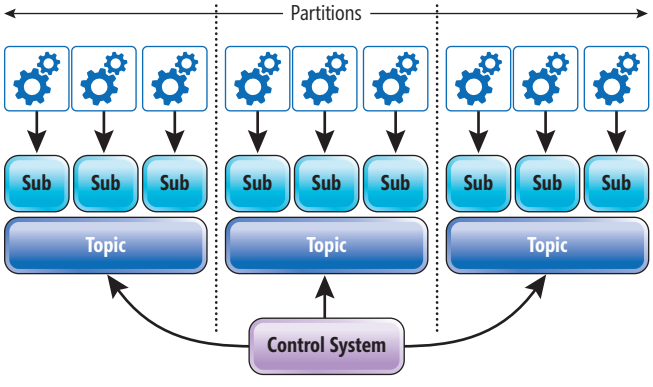



Figure 4 **The Fan-out Model**

# 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

Copyright 1996-2012 Infragistics, Inc. All rights reserved. Infragistics and NetAdvantage are registered trademarks of Infragistics, Inc. The Infragistics logo is a trademark of Infragistics, Inc. All other trademarks or registered trademarks are the respective property of their owners.

HTML 5

The first *MSDN Magazine* article in this series, "Building the Internet of Things" (msdn.microsoft.com/magazine/hh852591), is about using Microsoft StreamInsight in the device context. It's a great illustration of the rollup stage; the example described in the article could easily take the place of the aggregator in this architecture.

## Command Fan-out, Targeting and Collecting Responses

As **Figure 4** shows, the Control System will send commands into Topics and devices will receive messages from subscriptions for fan-out of commands.

The key argument for using a subscription per device is reliability. When you send a command, you want the command to eventually arrive at the device, even if the device is turned off or not currently connected. So you need the equivalent of a mailbox for the device.

The cost for that extra reliability and flexibility is somewhat greater complexity. The throughput ceiling for each entity I've discussed also manifests in the number of subscriptions Windows Azure Service Bus allows on each Topic; the system quota currently limits a Topic to 2,000 subscriptions. How many subscriptions you actually allocate on each Topic for the fan-out scenario depends on the desired flow rate. The cost associated with filtering and selecting a message into a subscription can be looked at as a copy operation. Therefore, a Topic with 1,000 subscriptions will yield a flow rate of one message per second if you assume a per-entity throughput of 1,000 messages per second.

For the emergency commands, you need to send one command to all connected devices in a broadcast fashion. For targeted commands, where a consumer adjusts the temperature on a particular device or in a particular house via an app connected on 3G, the adjustment should happen within a few seconds.

To figure out whether one message per second is also good enough for targeted commands, assume that a resident will adjust or ask for the current status of his air conditioning system very rarely. You should expect most commands to occur during significant weather events. Pessimistically assume every device gets adjusted once every day and adjustments generally occur between 7 a.m. and 7 p.m. If you're assuming a limit of 1,000 subscriptions for a Topic for the broadcast case, those commands would result in a requirement for 1.4 messages per minute. Even if everyone turned their air conditioning on during a single hour, you'd be OK with some 16 messages per minute. As a result of these scale and throughput considerations, limit each fan-out Topic to 1,000 subscriptions, which means you'll need 50 fan-out Topics for each scale-unit.

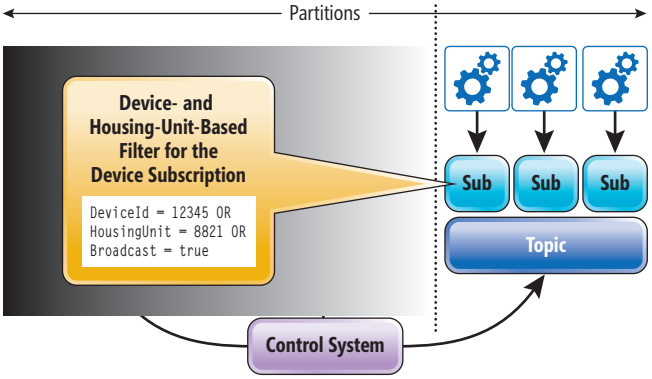



Figure 5 **Targeting Messages**

You can target messages to particular devices or housing units or issue a broadcast message to all devices using subscription filters. **Figure 5** illustrates a simple filter that allows messages carrying either a property with a particular DeviceId or a particular HousingUnit, or which have the custom Broadcast property set to true. If any of these conditions are true, the message is selected for the subscription. Thus, the control system can use the exact same Topic for broadcast and targeted messages and select the targets by setting the respective routing properties on the message.

The control system can also get replies to commands through its subscription on the fan-in side whereby command replies carry special filter properties that are handled in a similar fashion.

> Creating a new scale-unit is effectively a management script (using the Windows Azure Service Bus API) that creates a new structure of this shape.

## Namespace Structure, Provisioning and Device Identities

Windows Azure Service Bus uses the notion of namespaces to organize resources and to govern access to those resources. Namespaces are created via the Windows Azure Portal and are associated with a particular datacenter region. If your solution spans multiple large geographic regions, you can create multiple namespaces and then pin resources and clients to a particular Windows Azure datacenter with the obvious advantage of shorter network routes. For this solution I might create a namespace in the North/Central U.S. region (*clemensv-ac-ncus-prod*.servicebus.windows.net) and one in the South/Central U.S. region (*clemensv-ac-scus-prod*); or I might create namespaces for particular utility customers (*clemensv-ac-myenergy-prod*). The "prod" suffix distinguishes production namespaces from possible parallel test or staging namespaces.

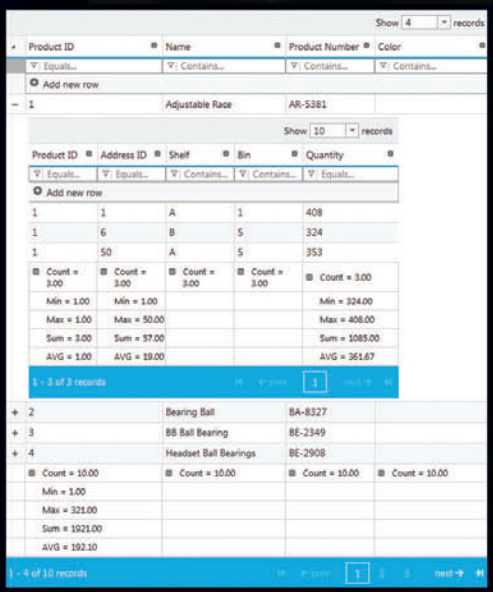
In order to make setup and management of scale-units easier, leverage the namespace's hierarchical structure. The scale-unit name, forming the base path for all entities within the scale-unit, could be just a number, or it could be associated with a particular customer (*/myenergy-3*). Because you have a single ingestion Topic, you can put it directly underneath (*/myenergy-3/fan-in*) and then put the fan-out topics underneath a separate segment by simply numbering the topics (*/myenergy-3/fan-out/01*). The fan-out subscriptions for the individual devices follow that structure, using the device-identifier as the subscription name (*/myenergy-3/fan-out/01/subscriptions/0123456*).

Creating a new scale-unit is effectively a management script (using the Windows Azure Service Bus API) that creates a new structure of this shape. The subscriptions for the devices and the

# 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

Copyright 1996-2012 Infragistics, Inc. All rights reserved. Infragistics and NetAdvantage are registered trademarks of Infragistics, Inc. The Infragistics logo is a trademark of Infragistics, Inc. All other trademarks or registered trademarks are the respective property of their owners.

HTML 5

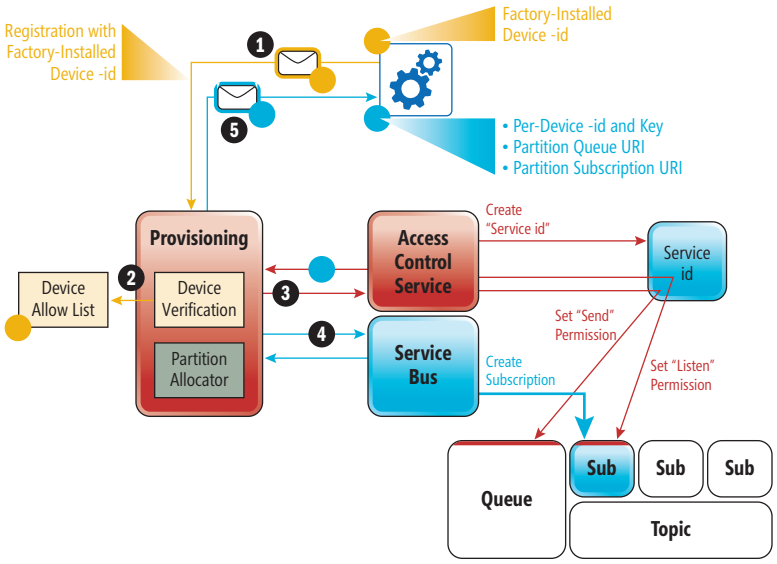



Figure 6 **The Provisioning Model**

filter rules are created during device provisioning. Of course, you also have to create, deploy and configure the matching storage and compute resources. For the event store, as noted earlier, you'll set up a new table per scale-unit.

Provisioning is about onboarding a factory-new device as it gets installed in a housing unit. For this article I'll skip over details such as device deactivation and just discuss a provisioning model in which the devices are set up with a unique device -id at the factory. The alternative model would be to have devices come from the factory entirely uninitialized and implement a scheme similar to what you might know from services such as Netflix, Hulu or Twitter, where the fresh device contacts a Web service to obtain a code and then displays the code, and the user logs on to an activation site by entering the code.

**Figure 6** shows the flow for provisioning a device that has a previously assigned identifier. The first step in setting up the device is not shown, and deals with getting the device on a network, which might happen via wired Ethernet, Wi-Fi or some wireless carrier network.

For provisioning, the system runs a special Web service that handles setting up and managing device identity and configuration. The partition allocator keeps track of how many devices have been activated, and which devices are assigned to which scale-unit. Once a device is associated with a scale-unit, the device gets provisioned into the scale-unit.

In step 1, the device establishes a connection to the provisioning service by presenting its device identifier. In this model, all issued device identifiers are on an allow list (step 2) that permits activating the device once. Once the device is verified to be good for activation, the provisioning service calls the Access Control Service (ACS) to create a new service identity and key for the device (step 3), and also creates a "Send" permission rule for the ingestion Topic and a "Listen" rule for the fan-out subscription created immediately afterward in step 4. With these, the device's account has only and exactly the rights it needs to interact with the system. After step 4, you have a collection of all the resource URIs the device will interact with, as well as an account and key that the device can use for authenticating against the ACS to obtain a token. All that information is put into the response to the device's activation call, and the device writes it into permanent configuration storage.

It's worth noting that though a good amount of scale-out partitioning has been introduced in this article, the Access Control identities aren't being partitioned in the example given here.

To alleviate the pressure on the ACS, you can simply have your client acquire fewer tokens. The default token expiration for the ACS relying party definition for Windows Azure Service Bus is 1,200 seconds, or 20 minutes. You can dial this up to 24 hours, or 86,400 seconds, and have the device's cache acquire tokens for that long. In this case, devices will have to acquire a fresh token only once per day. The only downside is that you can't revoke access to the bearer of the token if such a long-lived token were intercepted, but for tokens that are constrained to send or listen on particular entities that seems like a tolerable risk.

## In the Cloud

Connecting "things" to and through the cloud is an area that the Windows Azure Service Bus team will pay significant attention to over the next several years. The architecture outlined in this article is one it considers a good start, and it does—for instance, in the concept of scale-units—include hard-earned best practices from building Windows Azure Service Bus itself. There's a lot of room for optimizations around message flow, aggregation and distribution scenarios, not only in terms of capacity, but also for simpler programming and configuration models.

You'll start seeing some of those optimizations, such as automatic forwarding, appear in Windows Azure Service Bus within the next few months, and the team will be leveraging these new features to create better abstractions that will make scale-out scenarios like the one described here even simpler to manage. Distributing events to millions of devices in a timely fashion will always require a significant number of distinct system resources, but there's a lot of abstraction magic you can add to reduce the visible number of moving parts.

To make the scenario more tangible and more fun, the next article in this series will help you build a simple air conditioner (yes, hardware!) powered by the Microsoft .NET Micro Framework. This little do-it-yourself air conditioner will talk to a back-end service via Windows Azure Service Bus that will follow the general architecture explained here. See you then. ■

**CLEMENS VASTERS** *is the principal technical lead on the Windows Azure Service Bus team. Vasters has been on the Service Bus 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 the Web and devices. He is 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, Abhishek Lal, Colin Miller and Todd Holmquist-Sutherland*

# Just the Right Touch

**Get in touch with the new NetAdvantage for .NET 2012 V. 1 today, with a free, fully supported trial!**

www.infragistics.com/NET


Compatible with
**Microsoft® Visual Studio® 11 Beta**

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

Copyright 1996-2012 Infragistics, Inc. All rights reserved. Infragistics and NetAdvantage are registered trademarks of Infragistics, Inc. The Infragistics logo is a trademark of Infragistics, Inc. All other trademarks or registered trademarks are the respective property of their owners.

# CSS3 Effects, Transitions and Animations

Clark Sell

As a Web developer, you have three tools you can use to take your vision and make it reality: HTML, CSS and JavaScript. This wasn't always the case. In the past, seemingly simple effects like text shadows or gradients sent you off to separate image editors instead of using CSS and HTML. And though you did have JavaScript, making the Web application fluid and alive usually required considerable amounts of coding. Not only did these techniques complicate building the application initially, but any small changes were costly.

Luckily, CSS3 and HTML5 eliminate the need for the image and JavaScript gymnastics of the past. You can build smooth interactions and beautiful experiences with simple declarative markup.

Now, if you're like me, you might have just snickered a bit. Just because it's CSS and HTML doesn't mean it's simple. But as you'll see, by learning a few new CSS properties and trying them out in some cool demos, you'll save countless development hours—not to mention the hours spent negotiating the need for a particular effect with your client.

This article discusses:

- Creating shadow effects
- Using transforms to change elements
- Managing element changes with transitions
- Animating elements and controlling the animation

Technologies discussed:

CSS3, HTML5

## Shadows

Let's start off easy and look at some basic shadowing effects. For me, effects are all about altering the perception of an object. For example, take a look at the text shadow and box shadows in **Figure 1**.

This effect provides the illusion of a directional light source hitting an object and that object in turn casting a shadow on the objects around it. This helps give the perception of depth and some might even say it helps the object look like it's floating.

Adding a shadow isn't hard, so let's add a text shadow to a header:

```
h1 {
  text-shadow: black 2px 2px 12px 2px;
}
```

Here are the different text-shadow properties:

- **Shadow color (optional).** You can specify a color by name, or you can use HSL or RGBA color designations.
- **Horizontal offset (required).** This indicates the horizontal position of the shadow. Positive values move the shadow to the right of the object; negative values move it to the left.
- **Vertical offset (required).** This sets the vertical position of the shadow. Positive values move the shadow to the bottom; negative values move it to the top.
- **Blur Radius.** This value defines how clear the shadow text will appear. 0px is the font itself; increasing values increasingly blur an object's edge. Negative values are not allowed.
- **Spread Distance.** This value sets the distance away from the shadow—how the shadow's shape expands outwardly (a positive value) or contracts inwardly (a negative value).


Figure 1 **Shadow Effects**

Box shadows work the same as text shadows and have the same parameters:

```
box-shadow: red 10px 10px 0px 0px;
```

Shadows are quite common. They're used on elements such as buttons to provide depth and to give users an idea of the purpose of the element. They help point the user to the fact a particular object is somehow different from other objects visible in the UI. In the past, you likely turned to Photoshop and created PNGs to represent buttons; now you can do it in markup. Of course, I'm just scratching the surface here. You can create many more remarkable effects by adjusting opacity, border radius, gradients, typography and so forth.

## Transforms

The CSS transform property lets you transform a given element in terms of size or space. To demonstrate, let's set up an image so that when a user hovers over it, the image becomes twice as large:

```
#myImg:hover {
  transform: scale(2);
}
```

Of course this is CSS, which means you really need to include any applicable vendor prefixes. I'll omit them for the remainder of this article, but the preceding transform should've looked like this:

```
#myImg:hover {
  -ms-transform: scale(2);
  -webkit-transform: scale(2);
  -moz-transform: scale(2);
  -o- transform: scale(2);
  transform: scale(2);
}
```

You'll also want to take advantage of feature detection. This is actually one of the most important practices to implement in your Web sites today. Rather than basing the site's behavior on user agents, you should leverage a tool like Modernizr (modernizr.com), which lets you ask the browser about its capabilities. If a user's browser doesn't support a particular feature you need, you can instead use a polyfill—a shim that implements the feature for the browser. This is even possible for CSS. For more information, check out "No Browser Left Behind: An HTML5 Adoption Strategy" at msdn.microsoft.com/magazine/hh394148. Now back to transforms.

Besides scaling, you can also apply transforms on a 2D or 3D plane. With 2D, you're moving an element along its X and Y axes. Let's take some text and rotate it 45 degrees:

```
.transform2d {
  transform: translate(0px, 0px)
             rotate(45deg)
             scale(1.45)
             skew(0deg, 0deg);

}
```

**Figure 2** shows a box that has been rotated 45 degrees.

Now 2D is great, but clearly 3D is the rage today—and, no, we don't want to ask our users to put on any special glasses, but we do want people to think our boxes are about to come out of the screen. Here's some code to do that:

```
.transform3d {
  transform-origin-x: 50%;
  transform-origin-y: 50%;
  transform: perspective(110px) rotateY(45deg);
}
```

The results are shown in **Figure 3**.

## Transitions

Now let's explore transitions, which let you change an element from one style or state to another. I'll start with the hover pseudo class. Historically we've used CSS to set the states of an element, such as its initial state and its state after an event. Button elements and anchor links have two states—the initial state and the hover state. When a user activates the hover state, those properties get set. Here's a quick example for hovering over a button:

```
#boxTrans:hover {
  background-color: #808080;
  color: white;
  border-color: #4cff00;
  border-width: 3px;
}
```

Standard stuff, and no doubt you've done this before. Today's browsers are so fast that the state transition is almost instantaneous. That's great from a performance perspective, but it presents a new problem. In some cases, that state change might be jarring or not even visible. It's entirely possible the user will never see that quick and subtle change.

With CSS3 transitions, you can manage the time it takes for a transition to occur, and control other aspects of the transition as well. For example, rather than a button just automatically changing its background color, you can now specify what should happen during that change.

Adding a CSS transition is easy—you add the transitional commands to the base element. As an example, let's create a simple box with some text in it. When a user hovers over that box, the background color, text and border should change:

```
#boxTrans {
  ...

  transition: all .5s linear;

  ...
}

#boxTrans:hover {
  background-color: #808080;
    color: red;
    border-color: green;
}
```

As you can see, I've defined the transition on the #boxTrans element. I'm choosing to transition all properties of this element, and I want them all to transition linearly (that is, at a constant speed) over the course of

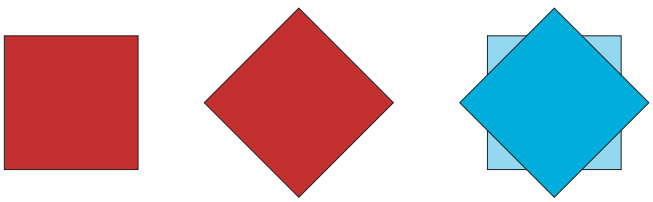


Figure 2 **Transforming an Element in 2D Space**

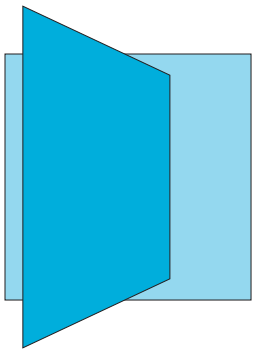


Figure 3 A 3D Transform

.5s. I can also choose specific properties to transition by indicating only those properties, like so:

```
transition: background-color .5s linear;
```

## Animations

Unlike transitions, where you tell the browser the start and end states, with animations you specify a series of CSS properties over a given time. Animations are really just extensions of transitions. To create an animation you use a keyframe. You can think of a keyframe as the state of an item at a given point in time within the duration set for that overall animation. Let's create a simple animation—a small box that scrolls back and forth. First, I'll define the element:

```
<html>
...

  <div class="box" id="boxAnimation"></div>

...
</html>
```

Let's give that div boxAnimation a bit of style so it looks like a box:

```
<style>
...

  .box {
  border: 1px solid black;
  background-color: red;

  width: 25px;
  height: 25px;
  position: relative;
  }

...
</style>
```

With that in place, I'll define the base animation. I need to set both the animation keyframe and its duration. If you don't set the duration property, the animation will never run because the default is 0. I'll also set the number of iterations I expect the animation to run; here I want it to run 10 times for 5 seconds each (if you prefer, you could set the animation-iteration-count to infinite, which would tell the animation to run as long as the page is open):

```
#boxAnimation {
  animation: 'not-knight-rider';
  animation-duration: 5s;
  animation-iteration-count: 10;
}
```

Last, I need to define the keyframe itself. I'm going to start with a simple keyframe that will move the box across the screen. To do this, I set the from and to properties and let the browser take care of the rest:

```
@keyframes not-knight-rider {
  from {
    left: -100px;
  }

  to {
    left: 100px;
  }
}
```

If you run this, you'll see a red box that floats across the screen and then repeats. As I mentioned, you have complete control of the animation. Let's update that keyframe to actually control where the box is at what time and what exactly it's doing, as shown in **Figure 4**.

In **Figure 4**, I'm defining what the keyframe will do at a given point in a single iteration. I define a starting point (to) and an ending point (from), as well as points in between, represented as percentages of its defined duration. Quite honestly, not very difficult. Now sparking up those creative juices to come up with some wicked animations—that's a different story!

Again, to support all of the different browsers on the market today you have to use the appropriate vendor prefixes. In the case of keyframes, this would look like @-webkit-keyframes and so on.

Note that you can see working examples of all the samples mentioned in this article online at bit.ly/10Pa4d, and you should also check out the incredible series of interactive, hands-on CSS3 demos at bit.ly/pF4sle, where you can explore a number of the different CSS specs without ever actually writing a line of CSS.

You might notice these demos are about Windows 8. That's because with the arrival of Windows 8, the Windows team introduced a new programming model. Web developers can now take their skills and build a native app for Windows 8 using HTML5, CSS3 and JavaScript. Microsoft has just opened a completely new surface area for you to sell your apps across the globe! I encourage you to check out the Windows Dev Center at dev.windows.com.

This is an exciting time to be a software developer, Web or otherwise. Technologies are advancing so fast, it's easy to get overwhelmed with all of the new tools in your toolbox. Regardless, it's important to look forward and understand the tools and techniques at your disposal. The last thing you want to do is start creating a bunch of pretty images when you can just add a few lines of CSS to achieve that drop shadow. ■

Figure 4 Controlling the Animation

```
@keyframes not-knight-rider {
  from {
    left: -100px;
    opacity: 0;
  }

  25% {
    left: 100px;
    opacity: 0.5;
  }

  50% {
    left: -100px;
    opacity: 0;
  }

  75% {
    left: 100px;
    opacity: 0.5;
  }

  to {
    left: -100px;    opacity: 0;
  }
}
```

**Clark Sell** *works as a senior Web and Windows 8 evangelist for Microsoft outside of Chicago. He blogs at csell.net, podcasts at developersmackdown.com and can be found on Twitter at twitter.com/csell5.*

**Thanks** to the following technical experts for reviewing this article: *John Hrvatin and Brandon Satrom*

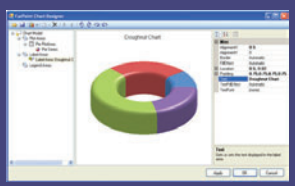
GrapeCity PowerTools

# Spreadsheets at their best!

**The world's most popular Microsoft® Excel®-compatible spreadsheet components for .NET Windows Forms and ASP.NET development.**

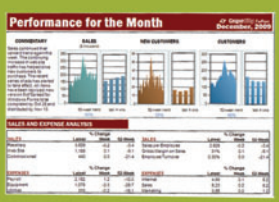
"It's fast. It does what we need it to do. It just works."

Jeffrey Baron
VP App Software Development



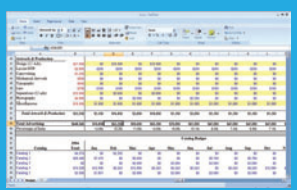
"We used Spread as a total back-end engine for doing calculations and everything"

Jay Kimble
Software Developer



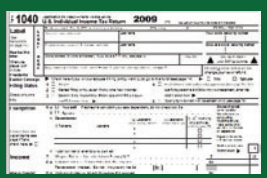

"Most people think that they just need a grid. Well, Spread is a grid, but it also has all of that Excel goodness in it."

Carl Franklin
Host, .NET Rocks!



"We built this tax software where the auditors actually go on site, enter the data and can remotely synch into the master head-quarters."

Rasesh Shah
Developer Evangelist




"Spread delivers a unique grid layout for our software solutions and lets us add user interface components to the grid layout."

Philippe Jolidon
Software Engineer RD

Download your FREE Trial Today!

GCPowerTools.com/Spread

SP Spread

GCPowerTools.com • GvTv.GCPowerTools.com

# Democratizing Video Content with Windows Azure Media Services

Bruno Terkaly and Ricardo Villalobos

One of the unmistakable trends slated for amazing growth over the next few years is the streaming of video content. According to the Cisco Visual Networking Index (VNI) 2011, Internet traffic related to this activity will quadruple by 2015. Multiple factors are behind this phenomenon, including: shrinking storage costs; scalable cloud compute and storage resources; high-bandwidth networks; and a multitude of video-enabled devices, such as tablets and smartphone devices. Creating, encoding and distributing video has never been more practical and affordable for even the casual hobbyist.

There's a major battle taking place over the way video content is being delivered and protected by copyright law. Many of us have witnessed the music industry undergo an amazing transformation, moving power away from the big record labels down to Internet-based media companies. The major carriers and cable providers today are concerned about the emerging growth of video on demand and the use of Internet channels to distribute it, including YouTube, Ustream and justin.tv, to name a few.

Microsoft isn't idly sitting by while all this happens. In May 2012, the software company released new Media Services features in Windows Azure, which will help democratize video creation, management and delivery to the masses. Windows Azure Media Services (just "Media Services" hereafter for brevity) support many different device types, from smartphones to tablets to high-end video workstations. This set of features—formerly code-named "Nimbus"—enables individual

This article discusses Windows Azure Media Services, a prerelease technology. All related information is subject to change.

**This article discusses:**

- Benefits of Windows Azure Media Services
- The open architecture of Windows Azure Media Services
- Leveraging the Windows Azure Media Services API
- Prerequisites needed to follow along
- Uploading and encoding content
- Managing content
- Controlling user access
- Downloading processed assets

**Technologies discussed:**

Windows Azure Media Services

**Code download available at:**

code.msdn.microsoft.com/mag201206AMS

Figure 1 **Two Approaches to Programmatically Use Windows Azure Media Services**

| | |
|---|---|
| REST/OData API | Developers can access the Media Services components through the Representational State Transfer (REST) API layer. Because REST is based on HTTP, it can be consumed from practically anywhere using a wide variety of languages, applications and platforms. In short, the REST API is the universal, public, programmatic interface for accessing the Media Services layer across the Web.<br>REST is an architectural strategy for building networked client-server applications based on open standards. REST provides a stateless, cacheable and standard protocol (HTTP) to send information between networked machines. |
| .NET API | Naturally, there's also a .NET Client SDK (which wraps the REST API calls and simplifies the process of building a .NET-based Media Services client application).<br>This article will focus on the .NET API. Future articles will address REST-based approaches. |

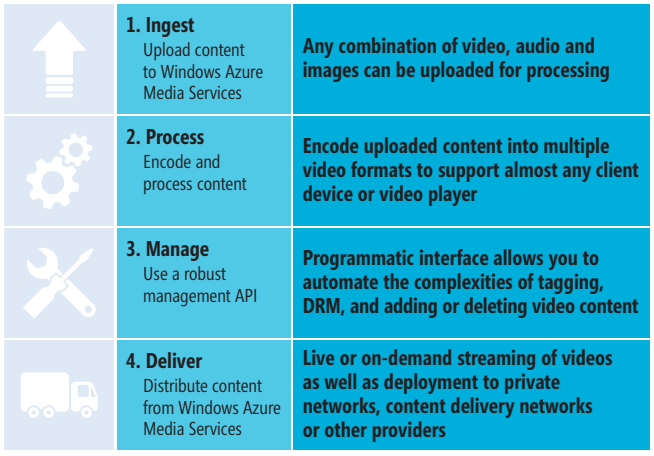



Figure 2 **Overall Workflow Required to Distribute Video Content**

developers and corporations to build cloud-based media solutions for ingesting, processing, managing and delivering media content.

## Solving Problems

These new Media Services features bring an automated, standardized and cost-effective solution to the media management space that dramatically lowers barriers and complexities by offering three main benefits:

1. Automatic scalability by leveraging the benefits that cloud computing offers, including the increase of capacity on demand, based on user needs.
2. A richly programmable interface based on a RESTful API, along with a Microsoft .NET Framework client library that allows developers to create, manage and maintain custom workflows easily.
3. A partner ecosystem that includes components from multiple companies that can be plugged into any step of the aforementioned workflows.

These comprehensive Media Services provide solutions to a variety of common problems for companies creating, managing and distributing content. The first one is cost, because building a video-capable datacenter is expensive. Having to provision servers, encoders, networking equipment and related software represents a high barrier to entry. The second problem that Media Services solve is the industry-wide issue of customized workflows. Many companies in the media segment have created their own proprietary business contracts, which makes it difficult to coordinate related services across different vendors using multiple formats and protocols. Standardizing the way video gets processed improves coordination and productivity when you have multiple participants involved in the creation and management of content.

## Open Standards

Media Services are based on open standards, meaning they can be leveraged by almost any client, including Java, PHP, Objective-C and jQuery, to name a few. The entire suite of capabilities is available through a common RESTful API, leveraging the Open Data Protocol (OData) to help with flexible querying capabilities. There's also a .NET C# SDK—built on top of the RESTful API—that facilitates and simplifies access to the framework for developers using this language (see **Figure 1**).

## Leveraging the Media Services API

In order to better understand the benefits of the Media Services features of Windows Azure, this article focuses on a specific case scenario. Imagine a home video that needs to be shared with multiple family members or friends who might have a wide variety of mobile devices. **Figure 2** depicts the overall workflow required to deliver custom processed video content to the world.

## Prerequisites

For the Windows Azure Media Services How-to Guide page, visit this link: go.microsoft.com/fwlink/?LinkID=247284.

For the Windows Azure Media Services account setup steps, visit this link: go.microsoft.com/fwlink/?LinkID=247287.

For the Windows Azure Media Services MSDN library content, visit this link: go.microsoft.com/fwlink/?linkid=245437.

Figure 3 **Uploading and Executing an Encoding Job**

```
static void Main(string[] args)
{
  // Part 1 - Connect to Media Services
  //          Setup upload progress event
  //          Upload a video to encode
  CloudMediaContext mediaContext =
    new CloudMediaContext("[ ACCOUNT NAME ]","[ ACCOUNT KEY ]");
  mediaContext.Assets.OnUploadProgress += Assets_OnUploadProgress;

  var asset = mediaContext.Assets.Create(
    @"C:\windows\Performance\WinSat\winsat.wmv");

  // Part 2 - Create a task, specify encoding details
  Console.Clear();

  IJob job = mediaContext.Jobs.CreateJob("Sample Job");

  var expressionEncoder = mediaContext.MediaProcessors.Where(
    mp => mp.Name == "Expression Encoder").Single();

  var task = job.Tasks.Add(
    mediaProcessor: expressionEncoder,
    configuration: "H.264 HD 720p VBR");

  task.Inputs.Add(asset);
  task.Outputs.Add("Sample Task Output Asset");

  // Part 3 - Submit the encoding job to begin processing
  while (job.State != JobState.Finished)
  {
    job = mediaContext.Jobs.Refresh(job.Id);
    Console.SetCursorPosition(0, 0);
    Console.WriteLine("Job Name: " + job.Name);
    Console.WriteLine("Job ID: " + job.Id);
    Console.WriteLine();
    Console.WriteLine("Job State: {0,-20}", job.State);
    Console.WriteLine("Task Progress: {0:0.00}%  ",
      job.Tasks.Single().Progress);

    Thread.Sleep(500);
  }

  Console.WriteLine();
  Console.WriteLine("Job Complete!");
  Console.ReadLine();
}

// Part 4 - Display completion progress (See Part 1, where the callback was set up)
static void Assets_OnUploadProgress(object sender, UploadProgressEventArgs e)
{
  Console.WriteLine(e.Progress);
}
```

Figure 4 **Comments for Source Code in Figure 3 (See Parts 1-4)**

| | |
|---|---|
| Part 1 | This code accomplishes three basic goals:<br>1. Creates a CloudMediaContext object, providing credentials to authenticate against the Media Services. This context is used throughout the rest of the code to manage and process the media assets.<br>2. Sets up the callback code called Assets_OnUploadProgress, which is used to report back to the user the percentage completion of the uploading process.<br>3. Uploads the original video file (winsat.wmv) so that it can be processed.<br>Note: The terminology used here is "assets," which, in this case, refers to the original video. |
| Part 2 | The goal of this code is to create a task, which comprises a media processor, a configuration, and inputs and outputs. Note that encoding is done with the configuration setting of "H.264 HD 720p VBR." We'll use Microsoft Expression Encoder. Additional media processors will be available as additional media industry partners add value on top of Media Services. |
| Part 3 | This is where the actual processing begins. You can think of a job as being a workflow with a set of tasks. Tasks can be chained together and they can be run in parallel. |
| Part 4 | This code is simply the callback that gets executed automatically during the video upload process. It's typically used to report back the percentage completion of the upload process. |

## Uploading and Encoding Content

Let's start with the basic tasks for distributing video content. The code in **Figure 3** accomplishes two important goals of anyone who works with video. The first one is to upload the original content so it can be processed. The second goal is to actually encode the content in such a way that it can be consumed by multiple devices. There are many different formats for video, such as MPEG, MPEG-2 (QuickTime), RealMedia, H.264 (MP4) and Windows Media, just to name a few—even YouTube has its own encoding format. Encoding formats aren't equal. Many of them are designed to be small, enabling fast Web downloads and renderings. Others, such as H.264, emphasize quality without a concern for video size.

Figure 5 **Various Assets Associated Within an Account**

```
static void Main(string[] args)
{
  // Part 1 - Connect to Media Services
  //          Loop through assets, displaying file information
  CloudMediaContext mediaContext = new CloudMediaContext("[ ACCOUNT NAME ]",
    "[ ACCOUNT KEY ]");

  Console.WriteLine("ASSET INFORMATION");
  foreach (IAsset asset in mediaContext.Assets)
  {
    // Display the collection of assets.
    Console.WriteLine("Asset ID: " + asset.Id);
    Console.WriteLine("Name: " + asset.Name);

    // Display the files associated with each asset.
    foreach (IFileInfo fileItem in asset.Files)
    {
      Console.WriteLine("File Name: " + fileItem.Name);
      Console.WriteLine("File Create Date: " + fileItem.Created);
      Console.WriteLine("File Size: " + fileItem.ContentFileSize);
    }
  }
}
```

**Figure 4** explains the code in **Figure 3** according to the sections labeled Part 1 through Part 4.

## Managing Content

One of the challenges of managing video-based projects is keeping track of all the assets, jobs and tasks. Producing content often involves dozens of original files and potentially hundreds more created as the encoded output. Being able to track and locate a specific item for additional processing can be a daunting experience. Automating this process with standard interfaces is a game changer. The management interface in Media Services will open the door for greater collaboration among stakeholders, making it possible to coordinate long-running workflows much more efficiently. The CloudMediaContext object provides an intuitive interface to locate any of the elements associated with the workflow, organized as a series of collections that can be queried using common libraries, such as LINQ.

The code in **Figure 5** demonstrates the ability to loop through the assets and display information about video files as well as other file types, such as images. Note that the context object has a collection of Assets and that an Asset object has a collection of File objects.

The code in **Figure 6** illustrates the ability to perform LINQ queries to search for specific assets. Keeping in mind that thousands of files are potentially being managed, search functionality like this can be a tremendous help.

## Access Policies

Controlling which users can view and access specific assets is a built-in ability of the framework for Media Services. To control security rights, owners of the content can programmatically specify an access policy, where details such as read/write permissions, as well as how long the access will be available, are specified. These are powerful capabilities in the context of sharing workflows with other video stakeholders.

This section shows how to perform basic tasks with access policies, such as creating and assigning them to an asset, as well as listing the ones already created. The purpose of the code in

Figure 6 **The Amazing Simplicity of LINQ When Searching for Assets Using a Name or ID**

```
// Returns an asset, given an asset name
static IAsset GetAssetNameFromContext(
  CloudMediaContext mediaContext, string assetName)
{
  IAsset asset = mediaContext.Assets.Where(a => a.Name ==
    assetName).FirstOrDefault();
  if (asset != null)
    return asset;
  else
    return null;
}
// Returns an asset, given an asset Id
static IAsset GetAssetIdFromContext(
  CloudMediaContext mediaContext, string assetId)
{
  IAsset asset = mediaContext.Assets.Where(a => a.Id ==
    assetId).FirstOrDefault();
  if (asset != null)
    return asset;
  else
    return null;
}
```

Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS

YOUR MAP TO THE .NET DEVELOPMENT PLATFORM


Redmond, WA | **August 6-10** | Microsoft Headquarters

# CODE ON CAMPUS

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


Check out the hot track topics that will make YOU a more valuable part of your company's development team:

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

vslive.com/redmond


- In-depth training for all levels of developers
- Stellar speaker lineup includes top industry experts and Microsoft insiders
- 10 tracks and 60+ sessions cover today's hot topics
- Full-day workshops
- Exclusive access to the Microsoft Campus—including the Microsoft Company Store!
- Special events and networking opportunities

vslive.com/redmond

**Register Before June 27th and Save $300!**

Use Promo Code RMTIP

Scan the QR code for more information on Visual Studio Live!

PLATINUM SPONSOR esri

SUPPORTED BY Microsoft, Microsoft Visual Studio, msdn, Visual Studio MAGAZINE

PRODUCED BY 1105 MEDIA

MEDIA SPONSOR THE CODE PROJECT

Figure 7 **Temporarily Exposing Some Video for Download**

```
static void Main(string[] args)
{
  // Part 1 - Specify an asset to upload
  //          Connect to Media Services
  string inputFilePath =
    @"C:\windows\Performance\WinSat\winsat.wmv";

  CloudMediaContext mediaContext =
    new CloudMediaContext("[ ACCOUNT NAME ]", "[ ACCOUNT KEY ]");

  // Part 2 - Upload an asset
  //          Create a policy for the asset
  //          Link access policy to asset
  IAsset asset = mediaContext.Assets.Create(inputFilePath);

  // Because this is a single-file asset, get the name of first file.
  string fileName = asset.Files[0].Name;

  IAccessPolicy readPolicy = mediaContext.AccessPolicies.Create(
    "CanReadFor30Minutes",
    TimeSpan.FromMinutes(30),
    AccessPermissions.Read);

  // Part 3 - Define a locator based on the read policy
  //          and expiration date
  //          Print the path for the locator you created
  //          Get the locator path Uri
  //          Build the full path to the file associated
  //          with the locator.
  ILocator locator = mediaContext.Locators.CreateSasLocator(
    asset, readPolicy);

  Console.WriteLine("Locator path: " + locator.Path);

  var uriBuilder = new UriBuilder(locator.Path);

  uriBuilder.Path += Path.AltDirectorySeparatorChar + fileName;

  Uri fullUrl = uriBuilder.Uri;

  // Part 4 - Print the full path to the file
  Console.WriteLine("Full URL to file: " + fullUrl);
}
```

Figure 8 **Downloading Assets from Windows Azure Media Services**

```
static void DownloadAsset(string assetId, string outputMediaFilesFolder)
{
  // Part 1 - Connect to Media Services
  //          Get an existing asset based on assetId passed in.
  CloudMediaContext mediaContext =
    new CloudMediaContext("[ ACCOUNT NAME ]","[ ACCOUNT KEY ]");
  IAsset asset = mediaContext.Assets.Where(a => a.Id ==
    assetId).SingleOrDefault();

  // Part 2 - If the asset exists, download the first file in the asset
  //          Download to outputMediaFilesFolder
  if (asset != null)
  {
    Console.WriteLine("Asset name: " + asset.Name);
    IFileInfo theFile = asset.Files.FirstOrDefault();

    // You could iterate through the asset.Files collection with a
    // foreach statement, and download all files for assets that
    // have multiple files.
    // Example: foreach(IFileInfo file in asset.Files)

    // Download the file to the specified local folder.
    if (theFile != null)
      theFile.DownloadToFile(Path.GetFullPath(outputMediaFilesFolder +
        Path.DirectorySeparatorChar + theFile.Name));
    else
      Console.WriteLine("No files available for this asset.");
  }
  else
  {
    Console.WriteLine("Asset not available.");
  }
}
```

**Figure 7** is to set up a "read" policy, allowing the video to be accessed for only 30 minutes. The code starts by uploading winsat.wmv. From there, a new access policy is created, called CanReadFor30Minutes, which means the file can be read only within 30 minutes of the upload process. The next step is to link the access policy to the uploaded video file.

A *locator* is a persistent copy of the policy file whose location is expressed as a URI. The locator allows access to the asset under the terms specified by the policy. In this case, in order to access the asset (the uploaded video) within the 30-minute time limit, we need to use the full URL, along with the locator file. See *fullUrl* in **Figure 7**.

## Downloading Processed Assets

Finally, the purpose of the code in **Figure 8** is to download the video content to local storage. In this case, the file winsat.mp4 is the H.264-encoded output, generated from the original video stored as winsat.wmv. You can call the function like this:

```
DownloadAsset("nb:cid:UUID:a0297fe4-7080-4393-b874-7ddf0f759c40", @"c:\temp");
```

The assetId of winsat.mp4 is nb:cid:UUID:a0297fe4-7080-4393-b874-7ddf0f759c40. It's a simple case of requesting the downloadable resource by its assetId and specifying a destination folder for downloading.

## Cloud Computing Benefits

Wrapping up, the Media Services features of Windows Azure simplify the process of encoding and distributing video content to multiple devices or Web channels that require different formats and resolutions. This is achieved by leveraging the benefits that cloud computing offers, including compute and storage scalability based on variable demand. This, along with a rich REST-based API and .NET client library, facilitate the process of creating, managing and maintaining workflows that can be enhanced with components created in a vast partner ecosystem.

Media Services represent a giant leap forward in the production, management and delivery of video content, allowing you to automate and pull together a wide variety of digital assets and organize them in a centralized location or even distributed locations. As video is poised for phenomenal growth on the Web over the next few years, Media Services come at a perfect time.

Later we'll cover the REST API in detail as well as some of the third-party support for custom encoders and high-end capabilities. ■

**Bruno Terkaly** *is a developer evangelist for Microsoft. His depth of knowledge comes from years of experience in the field, writing code using a multitude of platforms, languages, frameworks, SDKs, libraries and APIs. He spends time writing code, blogging and giving live presentations on building cloud-based applications, specifically using the Windows Azure platform.*

**Ricardo Villalobos** *is a seasoned software architect with more than 15 years of experience designing and creating applications for companies in the supply chain management industry. Holding different technical certifications, as well as an MBA in supply chain management from the University of Dallas, he works as a Windows Azure architect evangelist for Microsoft.*

**Thanks** to the following technical experts for reviewing this article: *John Deutscher, Samuel Ng and Tim Teebken*

# 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 and Save $300!

Use Promo Code REDJUNE

## 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

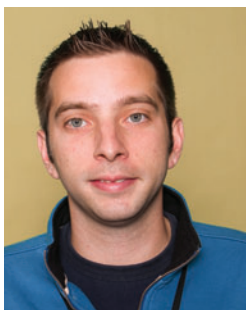
# What 2011's Visual Studio Live! Redmond Attendees Have to Say:

"Visual Studio Live! was a great place to learn about many different subjects in very little time. It was great fun to be on the Microsoft Campus too."
***–Swaroop Panda, Integration Architect, Hess Corporation***

"This was a phenomenal conference and they couldn't have picked a better venue than the Mothership [Microsoft Campus]. I'll definitely be back next year!"
***–Nma-Sie-Anyene Davenport, Software Developer, BGSU***

"The Visual Studio Live! event is an electrifying moment for a software developer. It's one of the few places where I can bond with other developers, share memorable moments, and just have an amazing time doing what we love — coding. Microsoft is, as always, a wonderful host and the event has definitely surpassed my expectations."
***–Tim Moore, Software Engineer, Civil Air Patrol***

"Visual Studio Live! is very informative. It helps you know which products are most helpful for your particular needs and helps you sharpen your skills. I always have a good time and learn a lot!"
***–Sandie Rives, Programming Consultant, Southern Farm Bureau Life Insurance***



"I have been extremely impressed with the VSLive! Conference and Microsoft campus. The Silverlight, WPF, and XAML classes have been very informative and have made me excited to go home to get coding. I strongly recommend this conference and enjoyed presentations by Billy Hollis and Rocky [Lhotka]. The "tips and tricks" sessions were also helpful in realizing how things are changing in VS for the best!"
***–Jacob Long, Sr. Developer Analyst, Levi, Ray, & Shoup, Inc.***

"Since leaving the industry in 2003 to join the ranks of academia it has been a struggle to maintain contact with the pulse of platform and tool evolution. VSLive is an ideal way to be totally immersed in an environment that provides a 'rapid' recharge of all the latest news & technologies from a user-centric viewpoint. I get more in one week at VSLive than I can get from a full three months of research over summer break."
***–Robert Logan, Assistant Professor, Kent State University***

"This is a great conference to learn about the newer technologies and to get to know the people who are experts in the field. It is also a time to network with other people/companies that have similar interests."
***–Mauanne Jennings, Business/ Technology Analyst, Jennings Information Systems***


**Register at vslive.com/redmond**
Use Promo Code REDJUNE

# VISUAL STUDIO LIVE! REDMOND AGENDA AT-A-GLANCE

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

## 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 **To Be Announced** | W15 **Microsoft Session To Be Announced** |

**Sponsored Break - Visit Exhibitors**

| | | | | |
|---|---|---|---|---|
| W16 **Live/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 **To Be Announced** | 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 **To Be Announced** | 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.*

# Behind the Scenes: A Windows Phone Feed-Reader App

Matt Stroshane

I'm a feed addict. I love the magic of RSS and Atom feeds, how news comes to me rather than the other way around. But with convenient access to so much information, consuming it in a meaningful way has become a challenge. So when I learned that some Microsoft interns were developing a Windows Phone feed-reader app, I was excited to find out how they approached the problem.

As part of their internship, Francisco Aguilera, Suman Malani and Ayomikun (George) Okeowo had 12 weeks to develop a Windows Phone app that included some new Windows Phone SDK 7.1 features. Being new to Windows Phone development, they were good test subjects for our platform, tools and documentation.

This article discusses:

- Using the app
- Parts of the app
- Using a local database
- Maintaining concurrency
- Consuming data
- Using a background agent
- Using Tiles

Technologies discussed:

Windows Phone

Code download available at:

code.msdn.microsoft.com/mag201206FeedReader

After considering their options, they decided on a feed-reader app that would demonstrate a local database, Live Tiles and a background agent. They demonstrated a lot more than that! In this article, I'm going to walk you through how they used those features. So install the Windows Phone SDK 7.1, download the code and get it up on your screen. Let's get going!

## Using the App

The central hub of the app is the main page, MainPage.xaml (**Figure 1**). It consists of four panorama panels: "what's new," "featured," "all" and "settings." The "what's new" panel shows the latest updates to the feeds. "Featured" displays six articles it thinks you'd like based on your reading history. The "all" panel lists all of your categories and feeds. To download articles only via Wi-Fi, use the setting on the "settings" panel.

The "what's new" and "featured" panels provide a way to navigate directly into an article. The "all" panel provides a list of categories and feeds. From the "all" panel, you can navigate to a collection of articles that are grouped by feed or category. You can also use the application bar on the "all" panel to add a new feed or category. **Figure 2** shows how the main page relates to the other eight pages of the app.

Similar to pivoting, you can navigate horizontally on the Category, Feed or Article pages. When you're on one of these pages, arrows appear in the application bar (see **Figure 3**). The arrows allow you to display the data for the previous or next category, feed or article in the database. For example, if you're viewing the


Figure 1 **The Main Page of the App After Creating a Windows Phone News Category**

Business category on the Category page, tapping the "next" arrow displays the Entertainment category on the Category page.

However, the arrow keys don't actually navigate to another Category page. Instead, the same page is bound to a different data source. Tapping the phone's Back button returns you to the "all" panel without the need for any special navigation code.

From the Article page, you can navigate to the Share page and send a link via messaging, e-mail or a social network. The application bar also provides the ability to view the article in Internet Explorer, "add to your favorites" or remove it from the database.

## Under the Hood

When you open the solution in Visual Studio, you'll see that it's a C# app divided into three projects:

1. FeedCast: The part the user sees—the foreground app (View and ViewModel code).
2. FeedCastAgent: The background agent code (periodic scheduled task).
3. FeedCastLibrary: The shared networking and data code.

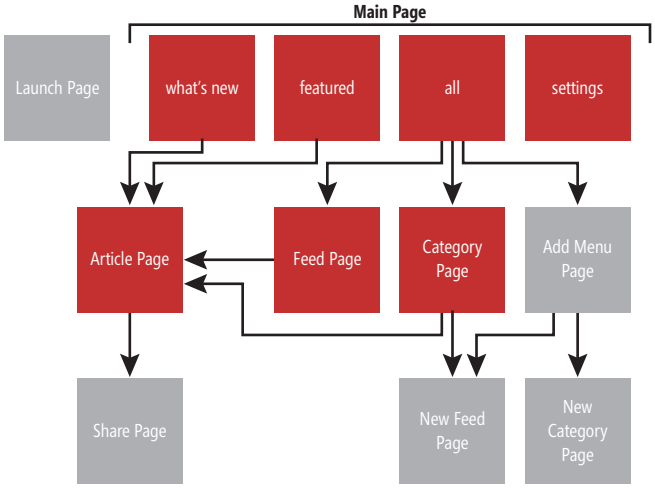



Figure 2 **The Page Navigation Map, with Auxiliary Pages in Gray**

The team used the Silverlight for Windows Phone Toolkit (November 2011) and the Microsoft Silverlight 4 SDK. Controls from the toolkit—Microsoft.Phone.Controls.Toolkit.dll—are used in most pages of the app. For example, HubTile controls are used to display articles in the "featured" panel of the main page. To help with networking, the team used System.ServiceModel.Syndication.dll from the Silverlight 4 SDK. This assembly isn't included in the Windows Phone SDK and isn't specifically optimized for phone apps, but the team members found it worked well for their needs.

The foreground app project, FeedCast, is the largest of the three in the solution. Again, this is the part of the app that the user sees. It's organized into nine folders:

1. Converters: Value converters that bridge the gap between data and UI.
2. Icons: Icons used by the application bars.
3. Images: Images used by HubTiles when articles have no images.
4. Libraries: The Toolkit and Syndication assemblies.
5. Models: Data-related code not used by the background agent.
6. Resources: Localization resource files in English and Spanish.
7. Themes: Customizations for the HeaderedListBox control.
8. ViewModels: ViewModels and other helper classes.
9. Views: Code for each page in the foreground app.

This app follows the Model-View-ViewModel (MVVM) pattern. Code in the Views folder focuses primarily on the UI. The logic and data associated with individual pages is defined by code in the ViewModels folder. Although the Models folder contains some data-related code, the data objects are defined in the FeedCastLibrary project. The "Model" code there is reused by the foreground app and background agent. For more about MVVM, see wpdev.ms/mwvmpnp.

The FeedCastLibrary project contains the data and networking code used by the foreground app and the background agent. This project contains two folders: Data and Networking. In the Data folder, the FeedCast "Model" is described by partial classes in four files: LocalDatabaseDataContext.cs, Article.cs, Category.cs and Feed.cs. The DataUtils.cs file contains the code that performs common database operations. A helper class for using isolated storage settings is in the Settings.cs file. The Networking folder of

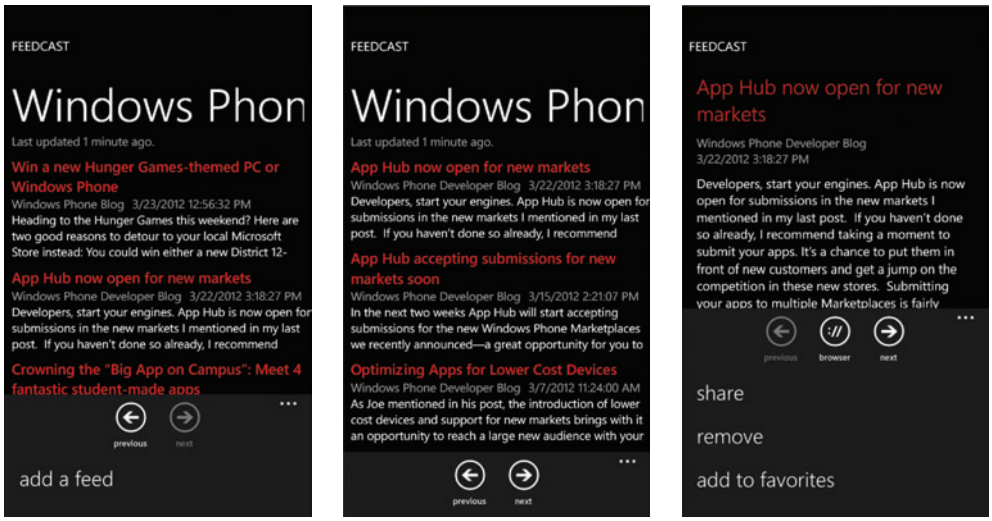



Figure 3 **The Category, Feed and Article Pages with Their Application Bars Expanded**

the FeedCastLibrary project contains the code used to download and parse content from the Web, the most significant of which are the Download methods in the WebTools.cs file.

There's only one class in the FeedCastAgent project, ScheduledAgent.cs, which is the background agent code. The OnInvoke method is called when it runs and the SendToDatabase method is called when downloads complete. I'll discuss more about downloading later.

## Local Database

For maximum productivity, each of the team members focused on a different area of the app. Aguilera focused on the UI, Views and ViewModels in the foreground app. Okeowo worked on the networking and getting data out of the feeds. Malani worked on the database architecture and operations.

In Windows Phone you can store your data in a local database. The local part is because it's a database file residing in isolated storage (your app's on-device storage bucket, isolated from other apps). Essentially, you describe your database tables as Plain Old CLR Objects, with the properties of those objects representing the database columns. This allows each object of that class to be stored as a row in the corresponding table. To represent the database, you create a special object referred to as a data context that inherits from System.Data.Linq.DataContext.

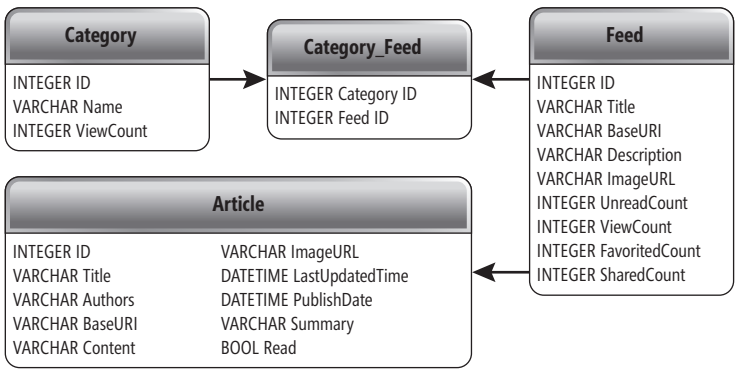



Figure 4 **The Database Schema**

The magical ingredient of the local database is the LINQ to SQL runtime—your data butler. You call the data context CreateDatabase method, and LINQ to SQL creates the .sdf file in isolated storage. You create LINQ queries to specify the data you want, and LINQ to SQL returns strongly typed objects that you can bind to your UI. LINQ to SQL allows you to focus on your code while it handles all the low-level database operations. For more about using a local database, see wpdev.ms/localdb.

Rather than type up all of the classes, Malani used Visual Studio 2010 Ultimate to take a different approach. She created the database tables visually, using the Server Explorer Add Connection dialog to create a SQL Server CE database and then the New Table dialog to build the tables.

Once Malani had her schema designed, she used SqlMetal.exe to generate a data context. SqlMetal.exe is a command-line utility from desktop LINQ to SQL. Its purpose is to create a data context class based on a SQL Server database. The code it generates is quite similar to a Windows Phone data context. Using this technique, she was able to build the tables visually and generate the data context quickly. For more about SqlMetal.exe, see wpdev.ms/sqlmetal.

The magical ingredient of the local database is the LINQ to SQL runtime—your data butler.

The database Malani built is shown in **Figure 4**. The three primary tables are Category, Feed and Article. Also, a linking table, Category_Feed, is used to enable a many-to-many relationship among categories and feeds. Each category can be associated with multiple feeds, and each feed can be associated with multiple categories. Note that the app's "favorite" feature is just a special category that can't be deleted.

However, the data context generated by SqlMetal.exe still contained some code that isn't supported on Windows Phone. After Malani added the data context code file to the Windows Phone project, she compiled the project to locate which code wasn't valid. She remembers having to remove one constructor, but the rest compiled fine.

Upon inspection of the data context file, LocalDatabaseDataContext.cs, you might notice that all of the tables are partial classes. The rest of the code associated with these tables (that wasn't autogenerated by SqlMetal.exe) is stored in the code files Article.cs, Category.cs and Feed.cs.

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

Copyright 1998- 2012 Developer Express Inc. All rights reserved. All trademarks are property of their respective owners.

Figure 5 **The GetResults Method in NewFeedPageViewModel.cs Queries Bing for New Feeds**

```
public void GetResults(string query, Action<int> Callback)
{
  // Clear the page ViewModel.
  Clear();

  // Get the search string and put it into a feed.
  Feed feed = new Feed { FeedBaseURI = GetSearchString(query) };

  // Lambda expression to add results to the page
  // ViewModel after the download completes.
  // _feedSearch is a WebTools object.
  _feedSearch.AllDownloadsFinished += (sender, e) =>
    {
      // See if the search returned any results.
      if (e.Downloads.Count > 0)
      {
        // Add the search results to the page ViewModel.
        foreach (Collection<Article> result in e.Downloads.Values)
        {
          if (null != result)
          {
            Deployment.Current.Dispatcher.BeginInvoke(() =>
              {
                foreach (Article a in result)
                {
                  lock (_lockObject)
                  {
                    // Add to the page ViewModel.
                    Add(a);
                  }
                }
                Callback(Count);
              });
          }
        }
      }
      else
      {
        // If no search results were returned.
        Deployment.Current.Dispatcher.BeginInvoke(() =>
          {
            Callback(0);
          });
      }
    };

  // Initiate the download (a Bing search).
  _feedSearch.Download(feed);
}
```

By separating the code this way, Malani could make changes to the database schema without affecting the extensibility method definitions that she wrote by hand. Had she not done this, she would have had to re-add the methods each time she autogenerated LocalDatabaseDataContext.cs (because SqlMetal.exe overwrites all code in the file).

## Maintaining Concurrency

As with most Windows Phone apps that aim to provide a responsive, fluid experience, this one uses multiple concurrent threads to do its work. In addition to the UI thread, which accepts user input, multiple background threads might be dealing with downloading and parsing the RSS feeds. Each of these threads will ultimately need to make changes to the database.

While the database itself offers robust concurrent access, the DataContext class is not thread-safe. In other words, the single global DataContext object used in this app can't be shared across multiple threads without adding some form of concurrency model. To address this problem, Malani used the LINQ to SQL concurrency APIs and a mutex object from the System.Threading namespace.

Figure 6 **The Background Agent Initiates a Download (Without Debug Comments)**

```
protected override void OnInvoke(ScheduledTask task)
{
  // Run the periodic task.
  List<Feed> allFeeds = DataBaseTools.GetAllFeeds();
  _remainingDownloads = allFeeds.Count;
  if (_remainingDownloads > 0)
  {
    Deployment.Current.Dispatcher.BeginInvoke(() =>
      {
        WebTools downloader = new WebTools(new SynFeedParser());
        downloader.SingleDownloadFinished += SendToDatabase;
        try
        {
          downloader.Download(allFeeds);
        }
        // TODO handle errors.
        catch { }
      });
  }
}
```

In the DataUtils.cs file, the mutex WaitOne and ReleaseMutex methods are used to synchronize access to data in cases where there could be contention between the DataContext classes. For example, if multiple concurrent threads (from the foreground app or the background agent) were to call the SaveChangesToDB method at about the same time, whichever code executes WaitOne first gets to continue. The WaitOne call from the other doesn't complete until the first code calls ReleaseMutex. For this reason, it's important to put your ReleaseMutex call in the finally statement when using try/catch/finally for database operations. Without a call to ReleaseMutex, the other code will wait at the WaitOne call until the owning thread exits. From the user's perspective, it could take "forever."

As with most Windows Phone apps that aim to provide a responsive, fluid experience, this one uses multiple concurrent threads to do its work.

Rather than a single global DataContext object, you can also design your app to create and destroy smaller DataContext objects on a per-thread basis. However, the team members found that the global-DataContext approach simplified development. I should also note that because the app only needed to protect against cross-thread access, not cross-process access, they could've also used a lock instead of the mutex. The lock might have offered better performance.

## Consuming Data

Okeowo focused his efforts on bringing data into the app. The WebTools.cs file contains the code where most of the action happens. But the WebTools class isn't only used for downloading

ComponentSource®
The Definitive Source of Software Components
www.componentsource.com

BEST SELLER

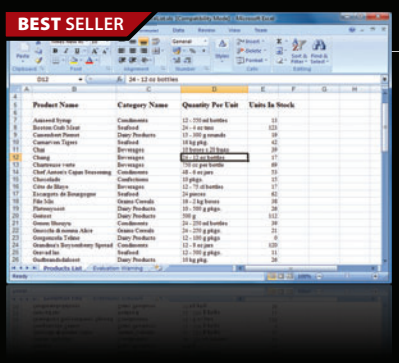

## GdPicture.NET | from $3,135.02

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

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 Medical Imaging SDKs V17.5 | from $4,495.50

LEAD TECHNOLOGIES

**Add powerful medical imaging to your Desktop, Tablet, Mobile & Web applications.**

- Comprehensive DICOM Data Set and metadata support
- High level PACS SDK with an OEM-Ready, fully customizable PACS Storage Server application
- 2D Viewers plus 3D Volume Reconstruction
- Zero footprint HTML5 / JavaScript viewer for any desktop, tablet or mobile device

BEST SELLER


## ActiveReports 6 | from $685.02

GrapeCity PowerTools

**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

© 1996-2012 ComponentSource. All Rights Reserved. All prices correct at the time of press. Online prices may vary from those shown due to daily fluctuations & online discounts.

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

MasterCard VISA DISCOVER

GSA Schedule Contract GS-35F-0188R

Figure 7 **The Background Agent Saves Articles to the Database (Without Debug Comments)**

```
private void SendToDatabase(object sender, SingleDownloadFinishedEventArgs e)
{
  // Ensure download is not null!
  if (e.DownloadedArticles != null)
  {
    DataBaseTools.AddArticles(e.DownloadedArticles, e.ParentFeed);
    _remainingDownloads--;
  }

  // If no remaining downloads, tell scheduler the background agent is done.
  if (_remainingDownloads <= 0)
  {
    NotifyComplete();
  }
}
```

feeds—it's also used on the new-feed page to search for new feeds on Bing. He accomplished this by creating a common interface, IXmlFeedParser, and abstracting the parsing code into different classes. The SynFeedParser class parses the feeds and the SearchResultParser class parses the Bing search results.

However, the Bing query doesn't actually return articles (despite the collection of Article objects returned by the IXmlFeedParser interface). Instead, it returns a list of feed names and URIs. What gives? Well, Okeowo realized that the Article class already had the properties he needed to describe a feed; he didn't need to create another class. When parsing search results, he used ArticleTitle for the feed name and ArticleBaseURI for the feed URI. See SearchResultParser.cs in the accompanying code download for more.

> While the agent gets a lot of attention because it does the heavy lifting, it would never run if it weren't for the scheduled task.

The code in the new page ViewModel (NewFeedPageViewModel.cs in the sample code) shows how the Bing search results are consumed. First, the GetSearchString method is used to piece together the Bing search string URI based on the search terms that the user enters on NewFeedPage, as shown in the following code snippet:

```
private string GetSearchString(string query)
{
  // Format the search string.
  string search = "http://api.bing.com/rss.aspx?query=feed:" + query +
    "&source=web&web.count=" + _numOfResults.ToString() +
    "&web.filetype=feed&market=en-us";
  return search;
}
```

The _numOfResults value limits how many search results are returned. For more about accessing Bing through RSS, see the MSDN Library page, "Accessing Bing Through RSS," at bit.ly/kc5uY0.

The GetSearchString method is called in the GetResults method, where the data is actually retrieved from Bing (see **Figure 5**). The GetResults method looks a little backward because it lists a lambda expression that handles the AllDownloadsFinished event "inline," before the code to initiate the download is actually called. When the Download method is called, the WebTools object queries Bing per the URI that was constructed with GetSearchString.

The WebTools Download method is also used by the background agent (see **Figure 6**), but in a different way. Rather than download from just one feed, the agent passes to the method a list of several feeds. For retrieving results, the agent takes a different strategy. Rather than wait until articles from all feeds are downloaded (via the AllDownloadsFinished event), the agent saves the articles as soon as each feed download is complete (via the SingleDownloadFinished event).

The job of the background agent is to keep all of your feeds up-to-date. To do this, it passes to the Download method a list of all feeds. The background agent only has a short amount of time to run, and when its time is up, the process is stopped immediately. So as the agent downloads feeds, it sends the articles to the database one feed at a time. This way the background agent has a much higher probability of saving new articles before it's stopped.

The single-feed and multiple-feed Download methods are actually overloads for the same code. The download code initiates an HttpWebRequest for each feed (asynchronously). As soon as the first request returns, it calls the SingleDownloadFinished event

Figure 8 **Adding Articles to the Database in the DataUtils.cs File**

```
public void AddArticles(ICollection<Article> newArticles, Feed feed)
{
  dbMutex.WaitOne();
  // DateTime date = SynFeedParser.latestDate;
  int downloadedArticleCount = newArticles.Count;
  int numOfNew = 0;

  // Query local database for existing articles.
  for (int i = 0; i < downloadedArticleCount; i++)
  {
    Article newArticle = newArticles.ElementAt(i);
    var d = from q in db.Article
            where q.ArticleBaseURI == newArticle.ArticleBaseURI
            select q;

    List<Article> a = d.ToList();

    // Determine if any articles are already in the database.
    bool alreadyInDB = (d.ToList().Count == 0);
    if (alreadyInDB)
    {
      newArticle.Read = false;
      newArticle.Favorite = false;
      numOfNew++;
    }
    else
    {
      // If so, remove them from the list.
      newArticles.Remove(newArticle);
      downloadedArticleCount--;
      i--;
    }
  }
  // Try to submit and update counts.
  try
  {
    db.Article.InsertAllOnSubmit(newArticles);
    Deployment.Current.Dispatcher.BeginInvoke(() =>
      {
        feed.UnreadCount += numOfNew;
        SaveChangesToDB();
      });
    SaveChangesToDB();
  }
  // TODO handle errors.
  catch { }
  finally { dbMutex.ReleaseMutex(); }
}
```

nsoftware

# We didn't invent the Internet...

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

TOOLS • COMPONENTS • ENTERPRISE ADAPTERS

- **E-Business**
  AS2, EDI/X12, NAESB, OFTP …
- **Credit Card Processing**
  Authorize.Net, TSYS, FDMS …
- **Shipping & Tracking**
  FedEx, UPS, USPS …
- **Accounting & Banking**
  QuickBooks, OFX …
- **Internet Business**
  Amazon, eBay, PayPal …
- **Internet Protocols**
  FTP, SMTP, IMAP, POP, WebDav …
- **Secure Connectivity**
  SSH, SFTP, SSL , Certificates …
- **Secure Email**
  S/MIME, OpenPGP …
- **Network Management**
  SNMP, MIB, LDAP, Monitoring …
- **Compression & Encryption**
  Zip, Gzip, Jar, AES …

Microsoft .NET | Microsoft BizTalk Server | Microsoft SQL Server | Windows Mobile | activex | Microsoft SharePoint 2010 | CodeGear Delphi | CodeGear C++Builder | Java | php

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

Each day, as you click around the Web or use any connected application, chances are that directly or indirectly some bits are flowing through applications that use our components, on a server, on a device, or right on your desktop. It's your code and our code working together to move data, information, and business. We give you the most robust suite of components for adding Internet Communications, Security, and E-Business Connectivity to any application, on any platform, anywhere, and you do the rest. Since 1994, we have had one goal: to provide the very best connectivity solutions for our professional developer customers. With more than 100,000 developers worldwide using our software and millions of installations in almost every Fortune 500 and Global 2000 company, our business is to connect business, one application at a time.

connectivity
powered by

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

Microsoft .NET, | Java | ActiveX | C++ | Delphi | C++ Builder | PHP | Objective-C | Windows | Windows Mobile | Mac OS X | Linux/Unix | iPhone

Figure 9 **Scheduling the Periodic Task in BackgroundAgentTools.cs (Minus Comments)**

```
public bool StartPeriodicAgent()
{
  periodicDownload = ScheduledActionService.Find(periodicTaskName) as PeriodicTask;
  bool wasAdded = true;

  // Agents have been disabled by the user.
  if (periodicDownload != null && !periodicDownload.IsEnabled)
  {
    // Can't add the agent. Return false!
    wasAdded = false;
  }

  // If the task already exists and background agents are enabled for the
  // application, then remove the agent and add again to update the scheduler.
  if (periodicDownload != null && periodicDownload.IsEnabled)
  {
    ScheduledActionService.Remove(periodicTaskName);
  }

  periodicDownload = new PeriodicTask(periodicTaskName);
  periodicDownload.Description =
    "Allows FeedCast to download new articles on a regular schedule.";

  // Scheduling the agent may not be allowed because maximum number
  // of agents has been reached or the phone is a 256MB device.
  try
  {
    ScheduledActionService.Add(periodicDownload);
  }
  catch (SchedulerServiceException) { }

  return wasAdded;
}
```

handler. The feed information and articles are then packaged into the event using the SingleDownloadFinishedEventArgs. As shown in **Figure 7**, the SendToDatabase method is wired up to the SingleDownloadFinshed method. When that returns, SendToDatabase takes the articles out of the event arguments and passes them to the DataUtils object named DataBaseTools.

Should the agent finish all downloads within its allotted time, it calls the NotifyComplete method to notify the OS that it finished. This allows the OS to allocate those unused resources to other background agents.

Following the code one step deeper, the AddArticles method in the DataUtils class checks to make sure the article is new before it adds it to the database. Note in **Figure 8** how a mutex is used again to prevent contention on the data context. Finally, when the article is found to be new, it's saved to the database with the SaveChangesToDB method.

The foreground app uses a technique similar to that found in the background agent for consuming data with the Download method. See the ContentLoader.cs file in the accompanying code download for the comparable code.

## Scheduling the Background Agent

The background agent is just that—an agent that performs work in the background for the foreground app. As you saw earlier in **Figure 6** and **Figure 7**, the code that defines that work is a class named ScheduledAgent. It's derived from Microsoft.Phone.Scheduler.ScheduledTaskAgent (which is derived from Microsoft.Phone.BackgroundAgent). While the agent gets a lot of attention because it does the heavy lifting, it would never run if it weren't for the scheduled task.

The scheduled task is the object used to specify when and how often the background agent will run. The scheduled task used in this app is a periodic task (Microsoft.Phone.Scheduler.PeriodicTask). A periodic task runs regularly for a short amount of time. To actually get that task on the schedule and query for it and so on, you use the scheduled action service (ScheduledActionService). For more about background agents, see wpdev.ms/bgagent.

The scheduled task code for this app is in the BackgroundAgentTools.cs file, in the foreground app project. That code defines the StartPeriodicAgent method, which is called by App.xaml.cs in the application constructor (see **Figure 9**).

Prior to scheduling the periodic task, StartPeriodicAgent performs a few checks because there's always a chance you can't schedule the scheduled task. First of all, scheduled tasks can be disabled by users on the background tasks list in the Applications panel of Settings. There's also a limit to how many tasks can be enabled on a device at one time. It varies per device configuration, but it could be as low as six. If you attempt to schedule a scheduled task after that limit has been exceeded, or if your app is running on a 256MB device, or if you've already scheduled the same task, the Add method will throw an exception.

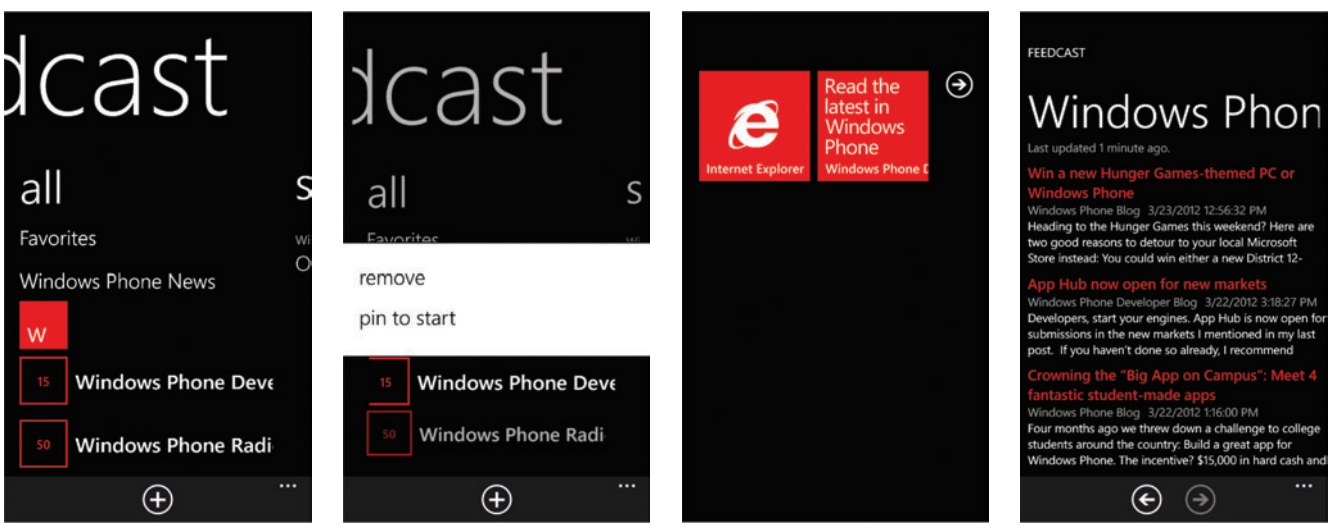



Figure 10 **Pinning the Windows Phone News Category to Start and Launching the Category Page**

# MANAGED TEAM FOUNDATION SERVER HOSTING

DiscountASP.NET is here to help development teams from all across the globe with their software development projects. With our NEW MANAGED TFS HOSTING solution, development teams no longer have to worry about the hassle of managing their own TFS server. We can do it for you! Plus,we provide all the tools and features you need to so you can effectively manage your software development projects:

- Your Own Private Instance of TFS
- 30gb of TFS disk space
- Source Code Management
- Bug/Issue Tracking
- Sharepoint
- Reporting Services
- Unlimited Projects
- TFS Build Server*
- NO LONG TERM CONTRACTS!

discountASP.net
Team Foundation Server Hosting

Microsoft Partner
Gold Hosting

**For More Info, visit**
**www.DiscountASP.NET/TFS/MSDN**

*TFS Build Server available as an add-on

Figure 11 **The Context Menu to Remove or Pin to Start a Category**

```
<toolkit:ContextMenuService.ContextMenu>
  <toolkit:ContextMenu>
    <toolkit:MenuItem Tag="{Binding}"
      Header="{Binding LocalizedResources.ContextMenuRemoveText,
               Source={StaticResource LocalizedStrings}}"
      IsEnabled="{Binding IsRemovable}"
      Click="OnCategoryRemoved"/>
    <toolkit:MenuItem Tag="{Binding}"
      Header="{Binding LocalizedResources.ContextMenuPinToStartText,
               Source={StaticResource LocalizedStrings}}"
      IsEnabled="{Binding IsPinned, Converter={StaticResource IsPinnable}}"
      Click="OnCategoryPinned"/>
  </toolkit:ContextMenu>
</toolkit:ContextMenuService.ContextMenu>
```

This app calls the StartPeriodicTask method every time it launches because background agents expire after 14 days. Refreshing the agent on every launch ensures the agent can continue running even if the app isn't launched again for several days.

The periodicTaskName variable in **Figure 9**, used to find an existing task, equals "FeedCastAgent." Note that this name doesn't identify the corresponding background agent code. It's simply a friendly name that you can use to work with ScheduledActionService. The foreground app already knows about the background agent code because it was added as a reference to the foreground app project. Because the background agent code was created as a project of type Windows Phone Scheduled Task Agent, the tools were able to wire up things correctly when the reference was added. You can see the foreground app-background agent relationship specified in the foreground app manifest (WMAppManifest.xml in the sample code), as shown here:

```
<Tasks>
  <DefaultTask Name="_default" NavigationPage="Views/MainPage.xaml" />
  <ExtendedTask Name="BackgroundTask">
    <BackgroundServiceAgent Specifier="ScheduledTaskAgent" Name="FeedCastAgent"
      Source="FeedCastAgent" Type="FeedCastAgent.ScheduledAgent"/>
  </ExtendedTask>
</Tasks>
```

## Tiles

Aguilera worked on the UI, the Views and ViewModels. He also worked on the localization and Tiles feature. Tiles, sometimes referred to as Live Tiles, display dynamic content and link to the app from Start. The application Tile of any app can be pinned to Start (without any work on the part of the developer). However, if you want to link to somewhere other than the main page of your app, you need to implement Secondary Tiles. These allow you to navigate the user deeper into your app—beyond the main page—to a page that you can customize for whatever the Secondary Tile represents.

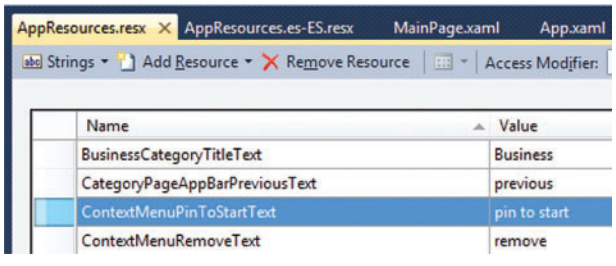



Figure 12 **The Default Resource File, AppResources.resx, Supplies the UI Text for All Languages Except Spanish**

In FeedCast, users can pin a feed or category (Secondary Tile) to Start. With a single tap, they can instantly be reading the latest articles related to that feed or category. To enable this experience, first they need to be able to pin the feed or category to Start. Aguilera used the Silverlight Toolkit for Windows Phone ContextMenu to facilitate this. Tapping and holding a feed or category in the "all" panel of the main page makes the context menu appear. From there, users can choose to remove or pin the feed or category to Start. **Figure 10** demonstrates the end-to-end process from the user's perspective.

**Figure 11** shows the XAML that makes the context menu possible. The second MenuItem displays "pin to start" (when English is the display language). When that item is tapped, the click event calls the OnCategoryPinned method to initiate the pinning. Because this app is localized, the text for the context menu actually comes from a resource file. This is why the Header value is bound to LocalizedResources.ContextMenuPinToStartText.

> If you want to link to somewhere other than the main page of your app, you need to implement Secondary Tiles.

This app has only two resource files, one for Spanish and the other for English (the default). However, because localization is in place, it would be relatively easy to add more languages. **Figure 12** shows the default resource file, AppResources.resx. For more information, see wpdev.ms/globalized.

Initially, the team wasn't quite sure how it was going to determine exactly which category or feed needed to be pinned. Then Aguilera discovered the XAML Tag attribute (see **Figure 11**). The team members figured out that they could bind it to the category or feed objects in the ViewModel and then retrieve the individual objects later, programmatically. On the main page, the category list is bound to a MainPageAllCategoriesViewModel object. When the OnCategoryPinned method is called, it uses the GetTagAs method to obtain the Category object (bound to the Tag) that corresponds with that particular item in the list, like so:

```
private void OnCategoryPinned(object sender, EventArgs e)
{
  Category tappedCategory = GetTagAs<Category>(sender);
  if (null != tappedCategory)
  {
    AddTile.AddLiveTile(tappedCategory);
  }
}
```

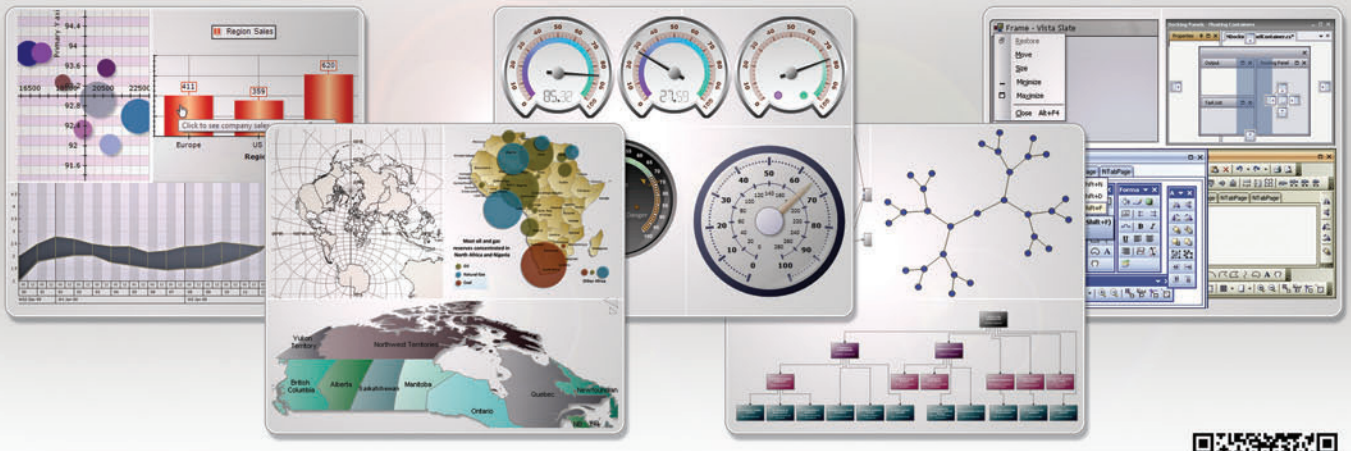
The GetTagAs method is a generic method for obtaining any object that has been bound to the Tag attribute of a container. Although this is effective, it's not actually necessary for most of its uses on MainPage.xaml.cs. The items in the list are already bound to the object, so binding them to the Tag is somewhat redundant. Rather than use Tag, you can use the DataContext of the Sender object. For example, **Figure 13** shows how OnCategoryPinned would look using the recommended DataContext approach.

NEVRON
visualize your success

NEVRON
**GAUGE** for .NET, SSRS, SharePoint

NEVRON
**CHART** for .NET, SSRS, SharePoint

NEVRON
**USER INTERFACE** for .NET



NEVRON
**MAP** for .NET

NEVRON
**DIAGRAM** for .NET

# **2012**vol.1 is here

## Features new ThinWeb controls with JQuery and ASP.NET MVC integration, SQL Server and SharePoint "Expressions Everywhere" and more.

Nevron components integrate seamlessly in **Web** and **Desktop .NET** applications, **SQL Server Reporting Services 2005/2008** reports and **SharePoint 2007/2010** portals and deliver an unmatched set of enterprise-grade features. That is why Nevron is the trusted vendor by many Fortune 500 companies for their most demanding data visualization needs.

**Download your free evaluation from**

**www.nevron.com**

DEVELOPERS

.NET

IT PROFFESIONALS

SharePoint

SSRS

Microsoft .NET ASP.net AJAX Microsoft SharePoint 2007/2010 Microsoft SQL Server 2005/2008 Microsoft Visual Studio PARTNER

Microsoft, .NET, ASP.NET, SharePoint, SQL Server and Visual Studio are registered trademarks of Microsoft Corporation in the United States and/or other countries.
Some Nevron components only available for certain platforms. For details visit www.nevron.com or send an email to support@nevron.com.

Figure 13 **An Example of Using DataContext Instead of GetTagAs**

```
private void OnCategoryPinned(object sender, EventArgs e)
{
  Category tappedCategory = null;
  if (null != sender)
  {
    FrameworkElement element = sender as FrameworkElement;
    if (null != element)
    {
      tappedCategory = element.DataContext as Category;

      if (null != tappedCategory)
      {
        AddTile.AddLiveTile(tappedCategory);
      }
    }
  }
}
```

This DataContext approach works well for all cases on MainPage.xaml.cs except for one, the OnHubTileTapped method. This is fired when you tap on a featured article in the "featured" panel of the main page. The challenge is due to the fact that the sender isn't bound to an Article class—it's bound to MainPageFeaturedViewModel. That ViewModel contains six articles, so it's not clearly known from the DataContext which one was tapped. Using the Tag property, in this case, makes it really easy to bind to the appropriate Article.

Because you can pin feeds and categories to Start, the AddLiveTile method has two overloads. The objects and Secondary Tiles differ enough that the team decided not to merge the functionality into a single generic method. **Figure 14** shows the Category version of the AddLiveTile method.

Before adding a Category Tile, the AddLiveTile method uses the ShellTile class to look at the navigation URIs from all of the active Tiles to determine if the category has already been added. If not, it continues and gets an image URL to associate with the new Tile. Whenever you create a new Tile, the background image needs to come from a local resource. In this case, it uses the ImageGrabber class to get a randomly assigned local image file. However, after you create a Tile, you can update the background image with a remote URL. But this particular app doesn't do that.

Figure 14 **Pinning a Category Object to Start**

```
public static void AddLiveTile(Category cat)
{
  // Does Tile already exist? If so, don't try to create it again.
  ShellTile tileToFind = ShellTile.ActiveTiles.FirstOrDefault(x =>
    x.NavigationUri.ToString().Contains("/Category/" + cat.CategoryID.ToString()));

  // Create the Tile if doesn't already exist.
  if (tileToFind == null)
  {
    // Create an image for the category if there isn't one.
    if (cat.ImageURL == null || cat.ImageURL == String.Empty)
    {
      cat.ImageURL = ImageGrabber.GetDefaultImage();
    }

    // Create the Tile object and set some initial properties for the Tile.
    StandardTileData newTileData = new StandardTileData
    {
      BackgroundImage = new Uri(cat.ImageURL, UriKind.RelativeOrAbsolute),
      Title = cat.CategoryTitle,
      Count = 0,
      BackTitle = cat.CategoryTitle,
      BackContent = "Read the latest in " + cat.CategoryTitle + "!",
    };

    // Create the Tile and pin it to Start.
    // This will cause a navigation to Start and a deactivation of the application.
    ShellTile.Create(
      new Uri("/Category/" + cat.CategoryID, UriKind.Relative), newTileData);
    cat.IsPinned = true;
    App.DataBaseUtility.SaveChangesToDB();
  }
}
```

All of the information you need to specify to create a new Tile is contained in the StandardTileData class. That class is used to put text, numbers and background images in the Tiles. When you create the Tile with the Create method, the StandardTileData is passed as a parameter. The other important parameter that's passed is the Tile navigation URI. This is the URI that's used to bring users to a meaningful place in your app.

In this app, the navigation URI from the Tile only takes the user as far as the app. To go further than that, a basic UriMapper class is used to route users to the right page. The App.xaml navigation element specifies all of the URI mapping for the app. In each UriMapping element, the value specified by the Uri attribute is the incoming URI. The value specified by the MappedUri attribute is where the user will be navigated to. To maintain the context of the particular category, feed or article, the id value in the brackets, {id}, is carried over from the incoming URI to the mapped URI, like so:

```
<navigation:UriMapping Uri="/Category/{id}" MappedUri=
  "/Views/CategoryPage.xaml?id={id}"/>
```

You might have other reasons to use a URI mapper—such as search extensibility, for example—but it isn't required to use a Secondary Tile. In this app, it was a style decision to use the URI mapper. The team felt that the shorter URIs were more elegant and easier to use. Alternatively, the Secondary Tiles could've specified a page-specific URI (such as the MappedUri values) for the same effect.

Regardless of the means, after the URI from the Secondary Tile is mapped to the appropriate page, the user arrives on the Category page with a list of his articles. Mission accomplished. For more about Tiles, see wpdev.ms/secondarytiles.

## But Wait, There's More!

There's a lot more to this app than what I covered here. Be sure to take a look at the code to learn more about how the team approached these and other problems. For example, SynFeedParser.cs has a nice way of cleaning up the data from feeds that are sometimes littered with HTML tags.

Just keep in mind that this is a snapshot of the interns' work at the end of 12 weeks, minus a little cleanup. Pro developers might prefer to code some parts differently. Nevertheless, I think the app did a great job of integrating a local database, background agent and Tiles. I hope you enjoyed this look "behind the scenes." Happy coding! ■

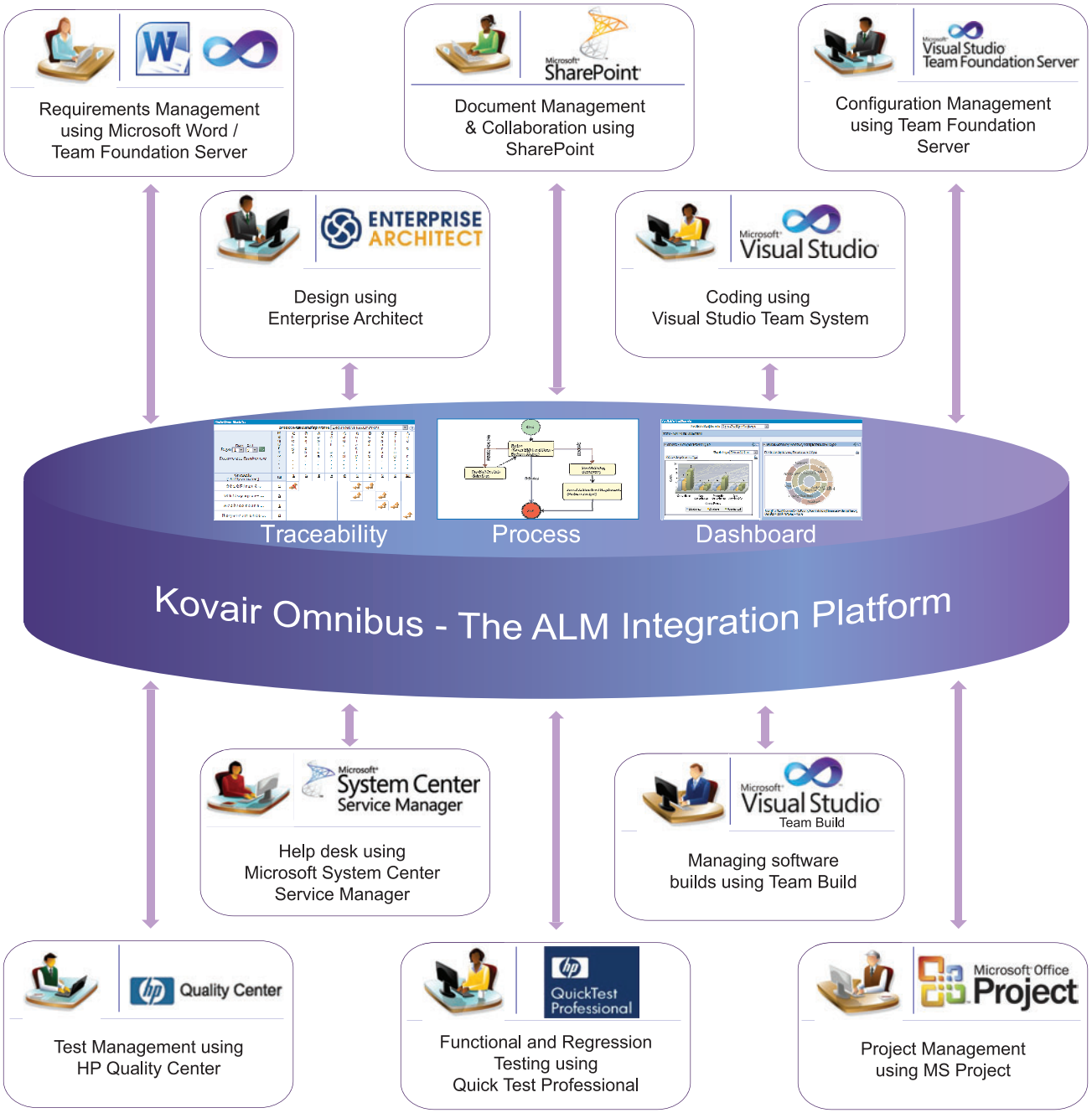
**MATT STROSHANE** *writes developer documentation for the Windows Phone team. His other contributions to the MSDN Library feature products such as SQL Server, SQL Azure and Visual Studio. When he's not writing, you might find him on the streets of Seattle, training for his next marathon. Follow him on Twitter at twitter.com/mattstroshane.*

**THANKS** to the following technical experts for reviewing this article: *Francisco Aguilera, Thomas Fennel, John Gallardo, Sean McKenna, Suman Malani, Ayomikun (George) Okeowo and Himadri Sarkar*

# Integrated ALM with Microsoft and Kovair

## Achieve End to End Traceability, Process Automation and Consolidated Reporting

## Bring Integration Between Microsoft and Other Tools





KOVAIR
www.kovair.com
email: sales@kovair.com

+1.408.262.0200 Press 1 [USA]
+91-33-4065 7016 / 17 [India]
+61 3 9680 8800 / +61 412 433 167 [AU]

# Visual Studio® LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS


YOUR MAP TO THE .NET DEVELOPMENT PLATFORM


# 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

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

LIVE! 360 IT EVENTS WITH PERSPECTIVE

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

# 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!

## 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**

## 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**

## 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**

## 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**

## SAVE UP TO $400 – Register Before October 10th

Use Promo Code DEVJUN

Live360events.com

PRODUCED BY 1105 MEDIA INC

# Custom Indexing for Latitude-Longitude Data

James McCaffrey

In this article I describe a technique to create custom indexes for geographical data that contains latitude and longitude location information. Custom indexes allow quick, real-time retrieval of data. For example, consider the following scenario. Suppose you have a very large (several million items) set of data that looks something like this:

```
0001 47.610 -122.330
0002 36.175 -115.136
0003 47.613 -122.332
...
```

Here the first column is a user ID (or the ID of any object associated with a location), the second column is the user's latitude and the third column is the user's longitude. Latitudes range from -90.0 degrees to +90.0 degrees and represent the up-down coordinate. Longitudes range from -180.0 degrees to +180.0 degrees and represent the left-right coordinate. The locations of the first and third users are near Seattle. A latitude value of zero is the equator. A longitude of zero is the Prime Meridian, which runs through Greenwich, England. Now suppose your data is stored in a SQL table and you want to find all the users who are located in the same 1-degree-by-1-degree rectangle as user 0001. You could do so with a simple SQL statement like this:

```
SELECT userID
FROM tblUsers
WHERE latitude >= 47.0 AND latitude < 48.0
AND longitude >= -123.0 AND longitude < -122.0
```

In many situations this approach works well. If you require real-time performance (perhaps a response time of less than 3 seconds) when the size of your data exceeds some threshold (depending on system resources), however, the response time starts to degrade dramatically.

One approach for getting real-time performance in situations like this is to assign each location to a sector ID. So suppose you're able to generate an auxiliary table from the initial data that looks something like this:

```
0001 49377
0002 45424
0003 49377
...
```

Here, the first column is the user ID and the second column is a sector ID. Now if the sector ID is indexed, a SQL statement like the following will be very fast even if your data set has hundreds of millions of items:

```
SELECT userID
FROM tblSectors
WHERE sector = 49377
```

This article discusses:

- Creating custom indexes for geographical data
- Mapping a latitude-longitude pair to a sector ID
- Determining sector size
- Finding adjacent sectors
- Putting together a geographical indexing scheme

Technologies discussed:

SQL Server 2008, Bing Maps

Code download available at:

code.msdn.microsoft.com/mag201206Indexing

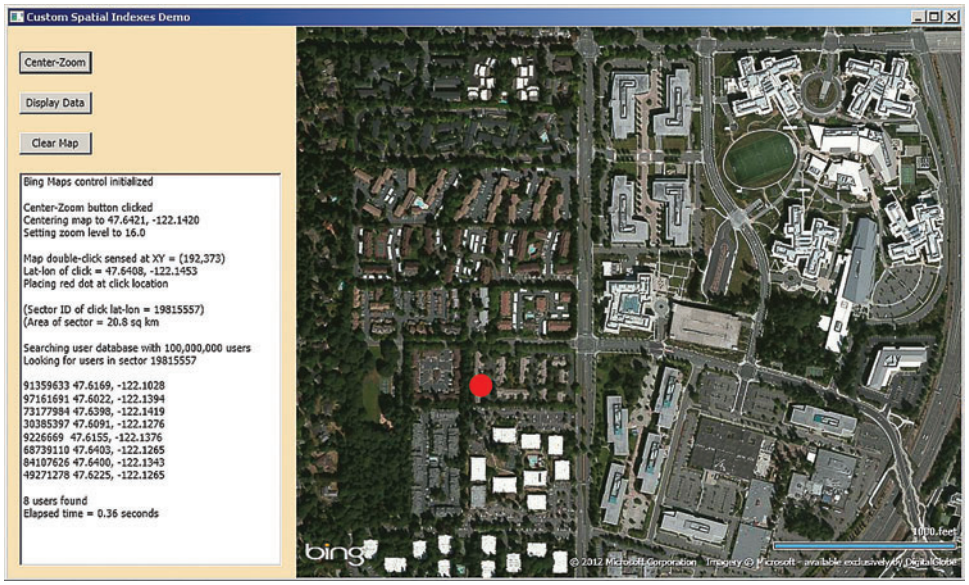



Figure 1 **Custom Spatial Indexing in Action**

The sector ID is acting as a custom index into the main data set. So the real challenge is to write a function that accepts a latitude and longitude and returns a sector.

To better understand the concept of custom indexes, take a look at the sample Windows Presentation Foundation (WPF) application in **Figure 1**. The application has an embedded Bing Maps map control. After centering the map to a location in Redmond, Wash., and zooming in, I double-clicked on the map. The double-click event was wired to an event handler that determined the latitude and longitude of the click and placed a red marker dot at the click location. Next, the app computed the sector ID of the click and then searched a back-end SQL database with 100 million random user-location items and found eight users in the same sector. The search time was 0.36 seconds—pretty impressive performance.

Microsoft SQL Server 2008 has a sophisticated spatial index feature you can use in the situation just described. And, in many cases, using a SQL Server spatial index is your best option. In at least two scenarios, however, creating custom indexes is a better approach. First, spatial indexes can be created only on a column of SQL type geometry or geography. If you have a legacy database or data source where latitudes and longitudes are stored as plain numeric values, converting those values to type geography might not be feasible. Second, even though spatial indexes in SQL Server 2008 are extremely powerful and quite customizable, in some scenarios you might need complete control over your indexing scheme.

Although creating custom indexes isn't a new concept, few concrete examples are available. In the remainder of this article, I'll present a complete example of how to create and use custom indexing. To fully understand this article, you need to have at least intermediate-level C# and T-SQL coding skills. You can get the key C# code for the example presented here from code.msdn.microsoft.com/mag201206Indexing.

## Mapping Latitude-Longitude to a Sector ID

There are many ways to map a latitude-longitude pair to a sector ID. The approach I describe here is the simplest, and is best explained with the diagram shown in **Figure 2**. The first decision in a custom indexing scheme is to choose how to segment the globe. In **Figure 2** I've divided each degree of latitude (the row values) and each degree of longitude (the columns) into two parts, as indicated by the fraction = 0.5 in the upper-left corner of the figure. This results in 360 latitude intervals, from [-90.0, -89.5) to [+89.5, +90.0], and 720 longitude intervals, from [-180.0, -179.5) to [+179.5, +180.0]. I use a square bracket to mean inclusive and a parenthesis to mean exclusive. With a fraction value of 0.5, there are 360 * 720 = 259,200 sectors numbered from 0 to 259,199. I order sectors from left to right and top to bottom, as shown in **Figure 2**.

Smaller values of the fraction parameter create more intervals per degree and generate more sectors. In this example I use the same value of fraction for both latitude and longitude intervals, but you can use different values if that better suits your data distribution. The sectors are not all the same area, however, as I explain shortly. If you have a row (latitude) index *ri* and a column (longitude) index *ci*, the corresponding sector ID *sid* can be computed as follows:

```
sid = (ri * 360 * 1/fraction) + ci
```

For example, in **Figure 2**, sector 1441 is associated with row index 2 and column index 1 and is computed as (2 * 360 * 1/0.5) + 1 = (2 * 720) + 1 = 1441. The term 360 * 1/fraction determines how many column intervals are in every row, and then multiplying by *ri* gives the first sector ID in the appropriate row. Adding the *ci* term essentially moves *ci* columns to the right from the beginning of the appropriate row and gives the final sector ID result.

The final part of the puzzle is to map a latitude value to a row index and a longitude value to a column index. A

| fraction = 0.5 | [-180.0, -179.5) 0 | [-179.5, -170.0) 1 | [-170.0, -169.5) 2 | | | [179.5, 180.0] 719 |
|---|---|---|---|---|---|---|
| [-90.0, -89.5) -> 0 | **0** | **1** | **2** | ... | | **719** |
| [-89.5, -80.0) -> 1 | **720** | **721** | **722** | ... | | **1439** |
| [-80.0, -79.5) -> 2 | **1440** | **1441** | **1442** | ... | | |
| | ... | | | | | |
| | | | | | | |
| [89.5, 90.0] -> 359 | **258,480** | | | ... | | **259,199** |

Figure 2 **Custom Indexing Design**

Figure 3 **The Row/Latitude Index of a Latitude**

```
static int LatIndex(double latitude, double fraction)
{
  latitude = Math.Round(latitude, 8);
  int lastIndex = (int)(180 * (1.0 / fraction) - 1);
  double firstLat = -90.0;

  if (latitude == -90.0) return 0;
  if (latitude == 90.0) return lastIndex;

  int lo = 0;
  int hi = lastIndex;
  int mid = (lo + hi) / 2;
  int ct = 0; // To prevent infinite loop

  while (ct < 10000)
  {
    ++ct;
    double left = firstLat + (fraction * mid); // Left part interval
    left = Math.Round(left, 8);
    double right = left + fraction;         // Right part interval
    right = Math.Round(right, 8);

    if (latitude >= left && latitude < right)
      return mid;
    else if (latitude < left)
    {
      hi = mid - 1; mid = (lo + hi) / 2;
    }
    else
    {
      lo = mid + 1; mid = (lo + hi) / 2;
    }
  }
  throw new Exception("LatIndex no return for input = " + latitude);
}
```

naive approach would be to do a simple linear search. Painful past experience with very large data sets has taught me that in this case, a binary search is a better approach. The idea is to start with a low index of zero, the middle index and a high index equal to the last possible index. Next, determine the corresponding latitude (or longitude) interval of the middle index. If the target latitude (or longitude) falls in the interval, the middle index is the correct return value. If the target latitude (or longitude) is smaller than the current interval, move the new middle index to halfway between the low index and the old middle index. If the target latitude (or longitude) is larger than the current interval, move the new middle index to halfway between the old middle index and the high index. Repeat the process until the correct middle index is found.

A C# implementation of a method LatIndex that accepts a latitude value and a fraction and returns the corresponding row index is presented in **Figure 3**. The partner LonIndex method that returns a column index for a longitude is completely symmetric except that the constant 180 is replaced by 360, and the constants -90.0 and 90.0 are replaced by -180.0 and 180.0.

Because the method in **Figure 3** works with latitudes as type double, the input latitude parameter is rounded to eight decimal places to avoid nasty equality-of-two-type-double errors. I use a somewhat-hacky *ct* variable to prevent an infinite loop. To keep the code short, I've removed most normal error checking—such as validating the input parameters—that you would use in a production scenario. Notice that the main loop checks for intervals in the form [a,b), so I must explicitly check for the last 90.0 value.

With the sector design and helper methods in place, I can map a latitude-longitude pair to a sector with a method like this:

```
static int LatLonToSector(double latitude, double longitude,
  double fraction)
{
  int latIndex = LatIndex(latitude, fraction); // row
  int lonIndex = LonIndex(longitude, fraction);  // col
  return (latIndex * 360 * (int)(1.0 / fraction)) + lonIndex;
}
```

## Sector Size

The custom indexing scheme I describe in this article divides the globe into numerically equal intervals based on the value of the fraction parameter. For example, if fraction is 0.5, each interval is 0.5 of a degree of latitude or longitude. However, this doesn't mean that all sectors have the same geographical area. Take a look at **Figure 4**. Latitude lines run parallel to each other and are all the same distance apart, about 111 kilometers. So the distance indicated by label A in **Figure 4** is the same as the distance indicated by B. But lines of longitude get closer together as they near each pole. So the distance indicated by label C is less than the distance indicated by D.

The distance, in kilometers, between any two points on the globe can be estimated using the following method:

```
static double Distance(double lat1, double lon1, double lat2, double lon2)
{
  double r = 6371.0; // approx. radius of earth in km
  double lat1Radians = (lat1 * Math.PI) / 180.0;
  double lon1Radians = (lon1 * Math.PI) / 180.0;
  double lat2Radians = (lat2 * Math.PI) / 180.0;
  double lon2Radians = (lon2 * Math.PI) / 180.0;
  double d = r * Math.Acos((Math.Cos(lat1Radians) *
    Math.Cos(lat2Radians) *
    Math.Cos(lon2Radians - lon1Radians) +
   (Math.Sin(lat1Radians) * Math.Sin(lat2Radians)));
  return d;
}
```

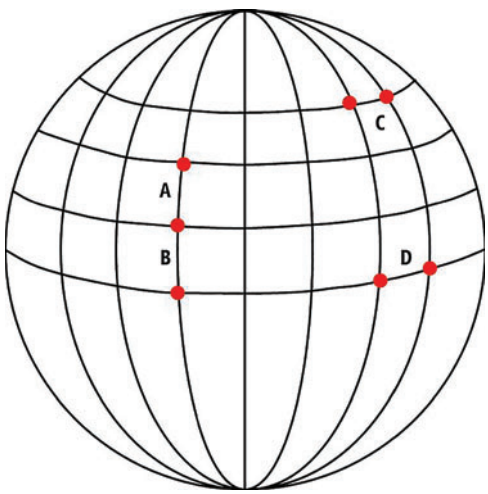



Figure 4 **Sectors Have Different Geographical Areas**

So if you know the corresponding latitude and longitude of a given sector, you can compute its approximate width and height, and then its approximate area. To determine the left endpoint of the interval containing a sector, you can use this method:

```
static double SectorToLat(int sector,
  double fraction)
{
  int divisor = 360 * (int)(1.0 / fraction);
  int row = sector / divisor;
  return -90.0 + (fraction * row);
}
```

This method essentially performs the reverse process of method LatLonToSector. For example, if the input sector is 1441 and the fraction parameter has the value 0.5, as shown in **Figure 2**, the local divisor variable is 360 * 1.0 / 0.5 = 720, which is the number of longitude

Figure 5 **Helper Methods for the AdjacentSectors Method**

```
static bool IsLeftEdge(int sector, double fraction)
{
  int numColumns = (int)(1.0 / fraction) * 360;
  if (sector % numColumns == 0) return true;
  else return false;
}

static bool IsRightEdge(int sector, double fraction)
{
  if (IsLeftEdge(sector + 1, fraction) == true) return true;
  else return false;
}

static bool IsTopRow(int sector, double fraction)
{
  int numColumns = (int)(1.0 / fraction) * 360;
  if (sector >= 0 && sector <= numColumns - 1) return true;
  else return false;
}

static bool IsBottomRow(int sector, double fraction)
{
  int numColumns = (int)(1.0 / fraction) * 360;
  int numRows = (int)(1.0 / fraction) * 180;
  int firstValueInLastRow = numColumns * (numRows - 1);
  int lastValueInLastRow = numColumns * numRows - 1;
  if (sector >= firstValueInLastRow && sector <= lastValueInLastRow)
    return true;
  else
    return false;
}
```

intervals. Then the value of variable row is 1441 / 720 = 2, which is the row index (note the integer division). Finally, -90.0 + 0.5 * 2 = -90.0 + 1.0 = -89.0, which is the left part of the [-89.0, -79.5) interval associated with sector 1441.

The method to determine the left part of the longitude interval of a sector is similar but not completely symmetric:

```
static double SectorToLon(int sector, double fraction)
{
  int divisor = 360 * (int)(1.0 / fraction);
  int col = sector % divisor;
  return -180.0 + (fraction * col);
}
```

With these two helper methods, you can determine the approximate area, in square kilometers, of a sector that has been defined by a fraction parameter:

```
static double Area(int sector, double fraction)
{
  double lat1 = SectorToLat(sector, fraction);
  double lon1 = SectorToLon(sector, fraction);
  double lat2 = lat1 + fraction;
  double lon2 = lon1 + fraction;
  double width = Distance(lat1, lon1, lat1, lon2);
  double height = Distance(lat1, lon1, lat2, lon1);
  return width * height;
}
```

## Adjacent Sectors

In some development scenarios, you might need to determine which sectors are adjacent to a given sector. For example, in **Figure 2**, if a user's location is in sector 721, you might want to determine which users are in adjacent sectors 0, 1, 2, 720, 722, 1440, 1441 and 1442. Notice that left-right sectors differ by plus or minus one, except for sectors on the left and right edges of the sector mapping. And up-down sectors, except for sectors on the top and bottom rows, differ by plus or minus the number of column intervals, which is 360 * (1 / fraction). To write a method AdjacentSectors that accepts a sector (and a generating fraction), it's useful to know whether a sector is on the left or right mapping edge, or on the top or bottom row. These four methods are presented in **Figure 5**.

You can write the method AdjacentSectors in many ways that trade off clarity and code size. One approach is to return an array of sector values where the [0] return value is the adjacent sector that's to the upper left of the input sector, [1] is directly above, [2] is above and to the right, [3] is to the left, [4] is to the right, [5] is below and to the left, [6] is directly below, and [7] is below and to the right. The code in **Figure 6** outlines one way to implement AdjacentSectors. Although the general case where the input sector isn't on a mapping edge is straightforward, some of the special cases, including the four-corner sectors, are tricky. See the code download that accompanies this article for the complete implementation.

## Putting Together a Geographical Indexing Scheme

A good way to understand the custom geographical indexing scheme described in this article is to examine a simple but complete end-to-end example. First create a dummy SQL database using T-SQL code similar to this:

```
use master
if exists(select * from sys.sysdatabases where name='dbGeoData')
  drop database dbGeoData
create database dbGeoData on
(
name=dbGeoData,
filename='D:\SomePlace\dbGeoData.mdf'
)
```

Next, create a table to hold some dummy data:

```
use dbGeoData
create table tblUsers
(
userID int primary key,
latitude real not null,
longitude real not null
)
```

Then write some C# code to generate dummy data:

```
Random r = new Random(0);
string initialDataFile = "..\\..\\UserIDLatLon.txt";
FileStream ofs = new FileStream(initialDataFile, FileMode.Create);
StreamWriter sw = new StreamWriter(ofs);
for (int i = 0; i < 1000000; ++i)
{
  double lat = (90.0 - (-90.0)) * r.NextDouble() + (-90.0);
  double lon = (180.0 - (-180.0)) * r.NextDouble() + (-180.0);
  sw.WriteLine(i.ToString().PadLeft(6,'0') + "," +
    lat.ToString("F8") + "," + lon.ToString("F8"));
}
sw.Close(); ofs.Close();
```

> The custom indexing scheme divides the globe into numerically equal intervals.

Here I'm creating 1 million lines of comma-separated data. To generate random latitudes and longitudes in the range [hi,lo), I'm using the standard pattern (hi - lo) * random.NextDouble() + lo. Next, copy the dummy data into the SQL table using the command-line bcp.exe program:

```
> bcp.exe dbGeoData..tblUsers in UserIDLatLon.txt -c -t , -r \n -S(local) -T
```

This command means, "Copy into table tblUsers, which is in database dbGeoData, the data in file UserIDLatLon.txt, which is character

Figure 6 **An Implementation of the AdjacentSectors Method**

```
static int[] AdjacentSectors(int sector, double fraction)
{
  int[] result = new int[8]; // Clockwise from upper-left
  int numCols = (int)(1.0 / fraction) * 360;
  int numRows = (int)(1.0 / fraction) * 180;

  int firstValueInLastRow = numCols * (numRows - 1);
  int lastValueInLastRow = numCols * numRows - 1;

  // General case
  if (IsLeftEdge(sector, fraction) == false &&
    IsRightEdge(sector, fraction) == false &&
    IsTopRow(sector, fraction) == false &&
    IsBottomRow(sector, fraction) == false)
  {
    result[0] = sector - numCols - 1;
    result[1] = sector - numCols;
    result[2] = sector - numCols + 1;
    result[3] = sector - 1;
    result[4] = sector + 1;
    result[5] = sector + numCols - 1;
    result[6] = sector + numCols;
    result[7] = sector + numCols + 1;
    return result;
  }

  // Deal with special cases here. See code download.

  throw new Exception("Unexpected logic path in AdjacentSectors");
}
```

data (that is, a text file) where each column is separated-terminated by the comma character and each line is return-terminated by a newline character. The database server is on the local machine, and Trusted (Windows) authentication is used to connect."

Then create the auxiliary custom indexing sector data using C# code like this:

```
string initialDataFile = "..\\..\\UserIDLatLon.txt";
string sectorDataFile = "..\\..\\ UserIDSector.txt";
FileStream ifs = new FileStream(initialDataFile, FileMode.Open);
StreamReader sr = new StreamReader(ifs);
FileStream ofs = new FileStream(sectorDataFile, FileMode.Create);
StreamWriter sw = new StreamWriter(ofs);
string line = "";
string[] tokens = null;
while ((line = sr.ReadLine()) != null)
{
  tokens = line.Split(',');
  int userID = int.Parse(tokens[0]);
  double lat = double.Parse(tokens[1]);
  double lon = double.Parse(tokens[2]);
  int sector = LatLonToSector(lat, lon, 0.5);
  sw.WriteLine(userID.ToString().PadLeft(6, '0') + "," + sector);
}
sw.Close(); ofs.Close(); sr.Close(); ifs.Close();
```

The code loops through the dummy data file one line at a time; parses out the user ID, latitude and longitude; computes the associated sector with a fraction parameter of 0.5; and writes the user ID and sector in comma-separated form to a text file.

Next, create a SQL table to hold the sector data:

```
create table tblSectors
(
userID int primary key,
sector int
)
```

Then copy the sector data into the table:

```
> bcp.exe dbGeoData..tblSectors in UserIDSector.txt -c -t , -r \n -S(local) -T
```

Finally, index the sector data:

```
if exists(select name from sys.sysindexes where name='ndxSector')
  drop index ndxSector on tblSectors
create nonclustered index ndxSector on tblSectors(sector)
```

At this point you can experiment in several ways. For example, you can select all the users who are in the same sector as user 3 with T-SQL code, like so:

```
declare @userID int
set @userID = 3
declare @sector int
select @sector = sector from tblSectors where userID=@userID
print @sector
select userID from tblSectors where sector = @sector
```

Of course, if you're accessing your SQL data from an application, you can use ADO.NET or LINQ equivalents to this kind of SQL code.

## Get Real-Time Performance

You can use custom indexing in many application scenarios. For example, suppose you have an ASP.NET Web application or a Silverlight application with a Bing Maps map control. You could create a system that allows the user to click on the map. The click event is associated with code that determines the latitude and longitude of the click, computes the associated sector and then retrieves from a back-end SQL database all users in the sector. As illustrated in **Figure 1**, even with several hundred million rows you can get real-time performance.

In some situations you might want to create several custom indexes. For example, you might want a low-granularity index with a fraction = 1.0 (so each sector is 1 degree by 1 degree), a mid-granularity index with fraction = 0.25 and a high-granularity index with fraction = 0.05. With multiple indexes you can successively filter your SQL data. First determine how many users are in the low-granularity sector. If you have too many users for your purposes, determine how many users are in the mid-granularity sector. If you have too few users, determine the seven adjacent sectors and the users in those sectors. And so on.

> In some scenarios custom indexes can lead to blazingly fast performance.

Let me reiterate that SQL Server 2008 has powerful built-in spatial indexing you should consider using before creating custom indexes. Custom indexes require explicit, additional SQL tables that impose an increased management burden on your system. That said, however, in some scenarios custom indexes can lead to blazingly fast performance. With the increasing use of GPS-enabled mobile devices, the ability to gather huge amounts of geographical data is only going to expand. The ability to create custom indexing schemes can be an important skill to help you analyze this data. ■

**DR. JAMES MCCAFFREY** *works for Volt Information Sciences, 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 also the author of ".NET Test Automation Recipes" (Apress, 2006). He can be reached jammc@microsoft.com.*

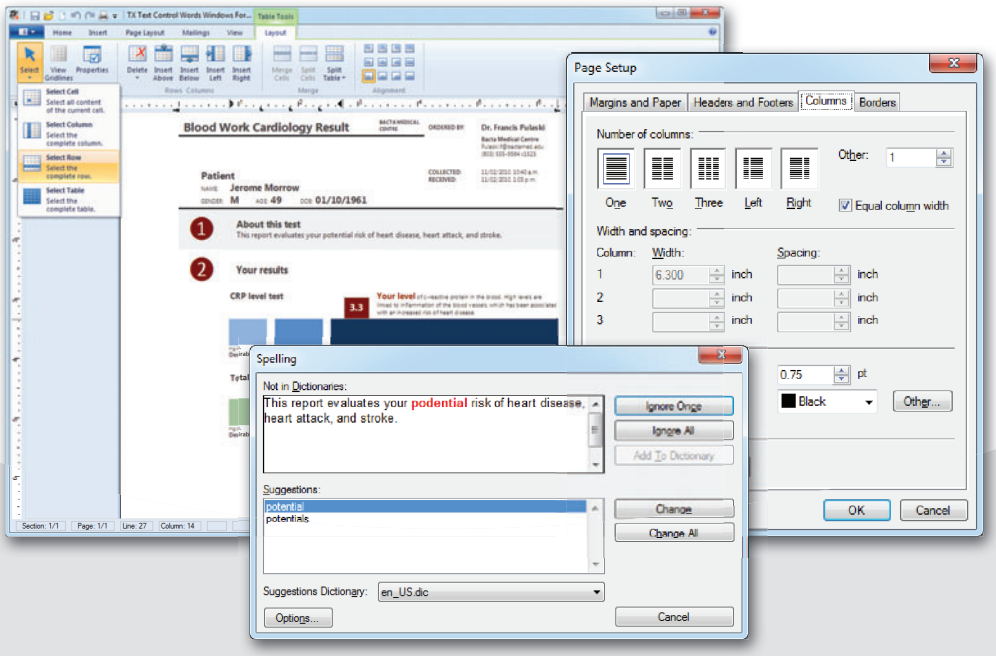
**THANKS** to the following technical experts for reviewing this article: *Isaac Kunen and Dan Liebling*

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

# Unit Testing in the Navigation for ASP.NET Web Forms Framework

Graham Mendick

The Navigation for ASP.NET Web Forms framework, an open source project hosted at navigation.codeplex.com, opens up new possibilities for writing Web Forms applications by taking a new approach to navigation and data passing. In traditional Web Forms code, the way data is passed is dependent on the navigation performed. For example, it might be held in the query string or route data during a redirection, but in control values or view state during a post back. However, in the Navigation for ASP.NET Web Forms framework ("Navigation framework" hereafter for brevity) a single data source is used in all scenarios.

In my first article (msdn.microsoft.com/magazine/hh975349), I introduced the Navigation framework and built a sample online survey application to demonstrate some of its key concepts and the advantages it provides. In particular, I showed how it enabled the generation of a set of dynamic, context-sensitive breadcrumb hyperlinks allowing a user to return to previously visited questions with the user's answers restored, overcoming the limitations of the static ASP.NET site map functionality.

Also in this first article, I claimed the Navigation framework lets you write Web Forms code that would make an ASP.NET MVC application green with envy. However, the sample survey application didn't bear this claim out because the code was crammed into the codebehinds and impenetrable to unit testing.

I'll set matters straight in this second article, editing the survey application so that it's just as well-structured as a typical ASP.NET MVC application and with a higher level of unit testability. I'll use standard ASP.NET data binding together with the Navigation framework to clear out the codebehinds and extract the business logic into a separate class, which I'll then unit test. This testing won't require any mocking and the code coverage will include the navigation logic—features rarely provided by other ASP.NET unit-testing approaches.

This article discusses:

- Data binding in Web Forms
- Navigation data binding
- Unit testing in the Navigation framework

Technologies discussed:

ASP.NET Web Forms

Code download available at:

code.msdn.microsoft.com/mag201206WebForms
navigation.codeplex.com

## Data Binding

The graveyard of Web Forms code is littered with the bloated corpses of codebehind files, but it doesn't have to be this way. Although Web Forms has featured data binding since its inauguration, it was in 2005 that Visual Studio introduced data source controls and the Bind syntax for two-way updatable binding, allowing the development

of Web Forms applications more akin in structure to a typical MVC application. The beneficial effects of such code, particularly with regard to unit testing, are widely recognized, reflected in the fact that the majority of the Web Forms development effort for the next version of Visual Studio has been spent in this area.

To demonstrate, I'll take the survey application developed in the first article and convert it to an MVC-like architecture. A controller class will hold the business logic and ViewModel classes will hold the data for communication between the controller and the views. It will require very little development effort because the code currently in the codebehinds can be cut and pasted almost verbatim into the controller.

> The graveyard of Web Forms code is littered with the bloated corpses of codebehind files, but it doesn't have to be this way.

Starting with Question1.aspx, the first step is to create a Question ViewModel class containing a string property so the selected answer can be passed to and from the controller:

```
public class Question
{
  public string Answer
  {
    get;
    set;
  }
}
```

Next comes the controller class, which I'll call SurveyController; a Plain Old CLR Object, unlike an MVC controller. Question1.aspx needs two methods, one for the data retrieval that returns the Question ViewModel class and one for the data update that accepts the Question ViewModel class:

```
public class SurveyController
{
  public Question GetQuestion1()
  {
    return null;
  }

  public void UpdateQuestion1(Question question)
  {
  }
}
```

To flesh out these methods I'll use the code in the codebehind of Question1.aspx, moving the page load logic into GetQuestion1 and the button click handler logic into UpdateQuestion1. Because the controller doesn't have access to the controls on the page, the Question ViewModel class is used to get and set the answer rather than the radio button list. The GetQuestion1 method requires a further tweak to ensure the default answer returned is "Web Forms":

```
public Question GetQuestion1()
{
  string answer = "Web Forms";
  if (StateContext.Data["answer"] != null)
  {
    answer = (string)StateContext.Data["answer"];
  }
  return new Question() { Answer = answer };
}
```

In MVC, data binding is at the request level with the request mapped to a controller method via route registration, but in Web Forms, data binding is at the control level with the mapping done using an ObjectDataSource. So to hook Question1.aspx up to the SurveyController methods, I'll add a FormView connected to an appropriately configured data source:

```
<asp:FormView ID="Question" runat="server"
  DataSourceID="QuestionDataSource" DefaultMode="Edit">
  <EditItemTemplate>
  </EditItemTemplate>
</asp:FormView>
<asp:ObjectDataSource ID="QuestionDataSource" runat="server"
SelectMethod="GetQuestion1"
  UpdateMethod="UpdateQuestion1" TypeName="Survey.SurveyController"
  DataObjectTypeName="Survey.Question" />
```

The final step is to move the question, consisting of the radio button list and button, inside the EditItemTemplate of the FormView. At the same time, two changes must be made in order for the data-binding mechanism to work. The first is to use the Bind syntax so the answer returned from GetQuestion1 is displayed and the newly selected answer is passed back to UpdateQuestion1. The second is to set the CommandName of the button to Update so UpdateQuestion1 will be automatically called when it's pressed (you'll notice the Selected attribute of the first list item has been removed because setting the default answer to "Web Forms" is now managed in GetQuestion1):

```
<asp:RadioButtonList ID="Answer" runat="server"
  SelectedValue='<%# Bind("Answer") %>'>
  <asp:ListItem Text="Web Forms" />
  <asp:ListItem Text="MVC" />
</asp:RadioButtonList>
<asp:Button ID="Next" runat="server" Text="Next" CommandName="Update" />
```

The process is complete for Question1.aspx, and its codebehind is gratifyingly empty. The same steps can be followed to add data binding to Question2.aspx, but its codebehind can't be completely cleared because the page load code related to the back navigation hyperlink must remain there for the time being. In the next section, where the integration of the Navigation framework with data binding is discussed, it will be moved into the markup and the codebehind vacated.

Adding data binding to Thanks.aspx is similar, but rather than reuse the inappropriately named Question ViewModel class, I'll create a new one called Summary with a string property to hold the answers selected:

```
public class Summary
{
  public string Text
  {
    get;
    set;
  }
}
```

Because Thanks.aspx is a read-only screen, only a data retrieval method is required on the controller and, as with Question2.aspx, all the page load code aside from the back navigation logic can be moved into this method:

```
public Summary GetSummary()
{
  Summary summary = new Summary();
  summary.Text = (string)StateContext.Data["technology"];
  if (StateContext.Data["navigation"] != null)
  {
    summary.Text += ", " + (bool)StateContext.Data["navigation"];
  }
  return summary;
}
```

Because no update functionality is required, the FormView ItemTemplate is used instead of EditItemTemplate, and the syntax for one-way binding, Eval, is used in place of Bind:

```
<asp:FormView ID="Summary" runat="server" DataSourceID="SummaryDataSource">
  <ItemTemplate>
    <asp:Label ID="Details" runat="server" Text='<%# Eval("Text") %>' />
  </ItemTemplate>
</asp:FormView>
<asp:ObjectDataSource ID="SummaryDataSource" runat="server"
  SelectMethod="GetSummary" TypeName="Survey.SurveyController" />
```

Half the unit testing battle is won because the survey application business logic has been extracted into a separate class. However, because the code has been pasted into the controller virtually unchanged from the codebehind, the power of data binding isn't yet fully utilized.

> I'll follow the AAA pattern for structuring a unit test.

## Navigation Data Binding

The survey application code still has a couple problems: Only the update methods in the SurveyController should contain navigational logic, and the codebehinds aren't empty. Unit testing shouldn't begin until these issues are resolved, as the former would result in unnecessarily complex unit tests for the *get* methods and the latter would prevent 100 percent unit test coverage.

The select parameters of data source controls make accessing the HttpRequest object in data-bound methods redundant. For example, the QueryStringParameter class allows query string data to be passed as parameters to data-bound methods. The Navigation framework has a NavigationDataParameter class that performs the equivalent job for the state data on the StateContext object.

Equipped with this NavigationDataParameter, I can revisit GetQuestion1, removing all the code that accesses the state data by making the answer a method parameter instead. This significantly simplifies the code:

```
public Question GetQuestion1(string answer)
{
  return new Question() { Answer = answer ?? "Web Forms" };
}
```

The accompanying change to Question1.aspx is to add the NavigationDataParameter to its data source. This involves first registering the Navigation namespace at the top of the page:

```
<%@ Register assembly="Navigation" namespace="Navigation" tagprefix="nav" %>
```

Then the NavigationDataParameter can be added to the data source's select parameters:

```
<asp:ObjectDataSource ID="QuestionDataSource" runat="server"
  SelectMethod="GetQuestion1" UpdateMethod="UpdateQuestion1"
  TypeName="Survey.SurveyController" DataObjectTypeName="Survey.Question" >
  <SelectParameters>
    <nav:NavigationDataParameter Name="answer" />
  </SelectParameters>
</asp:ObjectDataSource>
```

The GetQuestion1 method, having been stripped of all Web-specific code, is now easily unit tested. The same can be done for GetQuestion2.

For the GetSummary method, two parameters are needed, one for each answer. The second parameter is a bool to match how the data is passed by UpdateQuestion2, and it must be nullable because the second question isn't always asked:

```
public Summary GetSummary(string technology, bool? navigation)
{
  Summary summary = new Summary();
  summary.Text = technology;
  if (navigation.HasValue)
  {
    summary.Text += ", " + navigation.Value;
  }
  return summary;
}
```

And the corresponding change to the data source on Thanks.aspx is the addition of the two NavigationDataParameters:

```
<asp:ObjectDataSource ID="SummaryDataSource" runat="server"
  SelectMethod="GetSummary" TypeName="Survey.SurveyController" >
  <SelectParameters>
    <nav:NavigationDataParameter Name="technology" />
    <nav:NavigationDataParameter Name="navigation" />
  </SelectParameters>
</asp:ObjectDataSource>
```

The first problem with the survey application code has been addressed because now only the update methods in the controller contain navigational logic.

You'll recall that the Navigation framework improves on the static breadcrumb navigation functionality provided by the Web Forms site map, keeping track of the states visited together with their state data and building up a context-sensitive breadcrumb trail of the actual route taken by the user. To construct back navigation hyperlinks in markup—without needing codebehind—the Navigation framework provides a CrumbTrailDataSource analogous to the SiteMapPath control. When used as the backing data source for a ListView, the CrumbTrailDataSource returns a list of items, one per previously visited state, with each containing a NavigationLink URL that allows context-sensitive navigation back to that state.

> Half the unit testing battle is won because the survey application business logic has been extracted into a separate class.

I'll use this new data source to move the Question2.aspx back navigation into its markup. First, I'll add a ListView connected up to the CrumbTrailDataSource:

```
<asp:ListView ID="Crumbs" runat="server" DataSourceID="CrumbTrailDataSource">
  <LayoutTemplate>
    <asp:PlaceHolder ID="itemPlaceholder" runat="server" />
  </LayoutTemplate>
  <ItemTemplate>
  </ItemTemplate>
</asp:ListView>
<nav:CrumbTrailDataSource ID="CrumbTrailDataSource" runat="server" />
```

Next, I'll delete the page load code from the Question2.aspx codebehind, move the back navigation hyperlink inside the ListView ItemTemplate and use the Eval binding to populate the NavigateUrl property:

```
<asp:HyperLink ID="Question1" runat="server"
  NavigateUrl='<%# Eval("NavigationLink") %>' Text="Question 1"/>
```

You'll notice that the HyperLink Text property is hardcoded to "Question 1." This works perfectly well for Question2.aspx because

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 up to $300!

**Use promo code JUN1AD**

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

Produced By: 1105 MEDIA

Supported By: Redmond MAGAZINE | TechNet | VIRTUALIZATION REVIEW

the only possible back navigation is to the first question. However, the same can't be said for Thanks.aspx because it's possible to return to either the first or second question. Fortunately, the navigation configuration entered in the StateInfo.config file allows a title attribute to be associated to each state, such as:

```
<state key="Question1" page="~/Question1.aspx" title="Question 1">
```

And then the CrumbTrailDataSource makes this title available to data binding:

```
<asp:HyperLink ID="Question1" runat="server"
  NavigateUrl='<%# Eval("NavigationLink") %>' Text='<%# Eval("Title") %>'/>
```

Applying these same changes to Thanks.aspx addresses the second problem with the survey application code because all the codebehinds are now empty. However, all this effort will be wasted if the SurveyController can't be unit tested.

## Unit Testing

With the survey application now nicely structured—codebehinds are empty and all UI logic is in the page markup—it's time to unit test the SurveyController class. The GetQuestion1, GetQuestion2 and GetSummary data-retrieval methods clearly can be unit tested, as they contain no Web-specific code. Only the UpdateQuestion1 and UpdateQuestion2 methods present a unit testing challenge. Without the Navigation framework, these two methods would contain routing and redirection calls—the traditional way to move and pass data between ASPX pages—which both throw exceptions when used outside of a Web environment, causing unit testing to trip at the first hurdle. However, with the Navigation framework in place, these two methods can be fully unit tested without requiring any code change or mock objects.

For starters, I'll create a Unit Test project for the survey. Right-clicking inside any method in the SurveyController class and selecting the "Create Unit Tests ..." menu option will create a project with the necessary references included and a SurveyControllerTest class.

You'll recall that the Navigation framework requires the list of states and transitions to be configured in the StateInfo.config file. In order for the unit test project to use this same navigation configuration, the StateInfo.config file from the Web project must be deployed when the unit tests are executed. With this in mind, I'll double-click the Local.testsettings solution item and select the "Enable deployment" checkbox under the Deployment tab. Then I'll decorate the SurveyControllerTest class with the DeploymentItem attribute referencing this StateInfo.config file:

```
[TestClass]
[DeploymentItem(@"Survey\StateInfo.config")]
public class SurveyControllerTest
{
}
```

Next, an app.config file must be added to the test project pointing at this deployed StateInfo.config file (this configuration is also required in the Web project, but it was automatically added by the NuGet installation):

```
<configuration>
  <configSections>
    <sectionGroup name="Navigation">
      <section name="StateInfo" type=
        "Navigation.StateInfoSectionHandler, Navigation" />
    </sectionGroup>
  </configSections>
  <Navigation>
    <StateInfo configSource="StateInfo.config" />
  </Navigation>
</configuration>
```

With this configuration in place, unit testing can begin. I'll follow the AAA pattern for structuring a unit test:

1. Arrange: Set up the preconditions and test data.
2. Act: Execute the unit under test.
3. Assert: Verify the result.

Starting with the UpdateQuestion1 method, I'll show what's required at each of these three steps when it comes to testing the navigation and data passing in the Navigation framework.

The Arrange step sets up the unit test, creating the object under test and the parameters that need to be passed to the method under test. For UpdateQuestion1, this means creating a SurveyController and a Question populated with the relevant answer. However, an extra navigational setup condition is required, mirroring the navigation that occurs when the Web application is started. When the survey Web application is started, the Navigation framework intercepts the request for the startup page, Question1.aspx, and navigates to the dialog whose *path* attribute matches this request in the StateInfo.config file:

```
<dialog key="Survey" initial="Question1" path="~/Question1.aspx">
```

Navigating using a dialog's key goes to the state mentioned in its *initial* attribute, so the Question1 state is reached. Because it isn't possible to set a startup page in a unit test, this dialog navigation must be performed manually and is the extra condition required in the Arrange step:

```
StateController.Navigate("Survey");
```

The Act step calls the method under test. This just involves passing the Question with its answer populated to UpdateQuestion1 and so doesn't require any navigation-specific details.

The Assert step compares the results against expected values. Verifying the outcomes of navigating and data passing can be done using classes in the Navigation framework. You'll recall that the StateContext provides access to state data via its Data property, which is initialized with the NavigationData passed during navigation. This can be used to verify that UpdateQuestion1 passes the answer selected to the next state. So, assuming "Web Forms" is passed into the method, the Assert becomes:

```
Assert.AreEqual("Web Forms", (string) StateContext.Data["technology"]);
```

The StateContext also has a State property that tracks the current state. This can be used to check if a navigation occurred as expected—for example, that passing "Web Forms" into UpdateQuestion1 should navigate to Question2:

```
Assert.AreEqual("Question2", StateContext.State.Key);
```

While the StateContext holds details about the current state and associated data, the Crumb is the equivalent class for previously visited states and their data—so-called because each time a user navigates, a new one is added to the breadcrumb trail. This breadcrumb trail or list of crumbs is accessible via the Crumbs property of the StateController (and is the backing data of the CrumbTrailDataSource of the previous section). I need recourse to this list to check that UpdateQuestion1 stores the passed answer in its state data before navigating, because once the navigation occurs, a crumb is created holding this state data. Assuming the answer passed in is "Web Forms," the data on the first and only crumb can be verified:

```
Assert.AreEqual("Web Forms", (string) StateController.Crumbs[0].Data["answer"]);
```

The AAA pattern of writing a structured unit test has been covered regarding the Navigation framework. Putting all these steps together, following is a unit test to check if the Question2 state is

reached after passing an answer of "Web Forms" to UpdateQuestion1 (with a blank line inserted between the separate steps for clarity):

```
[TestMethod]
public void UpdateQuestion1NavigatesToQuestion2IfAnswerIsWebForms()
{
  StateController.Navigate("Survey");
  SurveyController controller = new SurveyController();
  Question question = new Question() { Answer = "Web Forms" };

  controller.UpdateQuestion1(question);

  Assert.AreEqual("Question2", StateContext.State.Key);
}
```

> The lofty ambition of unit-testable Web Forms code has been attained.

Although that's all you need to be able to unit test the different Navigation framework concepts, it's worth continuing with UpdateQuestion2 because it has a couple of differences in its Arrange and Act steps. The navigation condition required in its Arrange step is different because, to call UpdateQuestion2, the current state should be Question2 and the current state data should contain the "Web Forms" technology answer. In the Web application this navigation and data passing is managed by the UI because the user can't progress to the second question without answering "Web Forms" to the first question. However, in the unit test environment, this must be done manually. This involves the same dialog navigation required by UpdateQuestion1 to reach the Question1 state, followed by a navigation passing the Next transition key and "Web Forms" answer in NavigationData:

```
StateController.Navigate("Survey");
StateController.Navigate(
  "Next", new NavigationData() { { "technology", "Web Forms" } });
```

The only difference in the Assert step for UpdateQuestion2 comes when verifying its answer is stored in state data prior to navigation. When this check was done for UpdateQuestion1, the first crumb in the list was used because only one state had been visited, namely Question1. However, for UpdateQuestion2, there will be two crumbs in the list because both Question1 and Question2 have been reached. Crumbs appear in the list in the order they're visited, so Question2 is the second entry and the requisite check becomes:

```
Assert.AreEqual("Yes", (string)StateController.Crumbs[1].Data["answer"]);
```

The lofty ambition of unit-testable Web Forms code has been attained. This was done using standard data binding with assistance from the Navigation framework. It's less prescriptive than other ASP.NET unit testing approaches because the controller hasn't had to inherit or implement any framework class or interface, and its methods haven't had to return any particular framework types.

## Is MVC Jealous Yet?

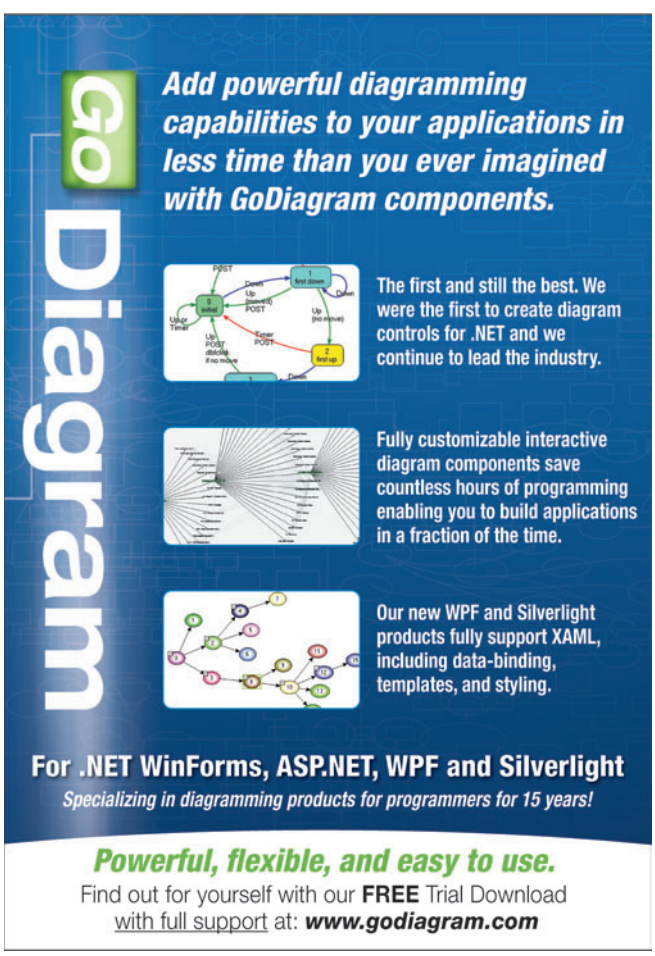
MVC should be feeling some pangs of jealousy because the survey application is just as well-structured as a typical MVC application, but has a higher level of unit testing. The survey application's navigational code appears inside the controller methods and is tested along with the rest of the business logic. In an MVC application, the navigational code isn't tested because it's contained within the return types of the controller methods, such as the RedirectResult. In my next article on the Navigation framework, I'll compound MVC's jealousy by building a Search Engine Optimization-friendly, single-page application with adherence to don't repeat yourself, or DRY, principles that's difficult to achieve in its MVC equivalent.

That said, Web Forms data binding does have problems that aren't present in its MVC counterpart. For example, it's difficult to use dependency injection on the controller classes, and nested types on the ViewModel classes aren't supported. But Web Forms has learned a lot from MVC, and the next version of Visual Studio will see a vastly improved Web Forms data-binding experience.

There's much more to the Navigation framework's integration with data binding than is shown here. For example, there's a data pager control that—unlike the ASP.NET DataPager—doesn't need hooking up to a control or require a separate count method. If you're interested in finding out more, comprehensive documentation and sample code are available at navigation.codeplex.com. ■

**GRAHAM MENDICK** *is Web Forms' biggest fan and wants to show it can be just as architecturally sound as ASP.NET MVC. He authored the Navigation for ASP.NET Web Forms framework, which he believes—when used with data binding—can breathe new life into Web Forms.*

**THANKS** to the following technical expert for reviewing this article: *Scott Hanselman*





JAMES MCCAFFREY

# Evolutionary Optimization Algorithms

An evolutionary optimization algorithm is an implementation of a meta-heuristic modeled on the behavior of biological evolution. These algorithms can be used to find approximate solutions to difficult or impossible numerical minimization problems. You might be interested in evolutionary optimization algorithms for three reasons. First, knowing how to code these algorithms can be a practical addition to your developer, manager and tester skill sets. Second, some of the techniques used, such as tournament selection, can be reused in other algorithms and coding scenarios. And third, many engineers simply find these algorithms interesting in their own right.

An evolutionary optimization algorithm is essentially a type of genetic algorithm in which the virtual chromosomes are made of real values instead of some form of bit representation. The best way for you to see where I'm headed is to take a look **Figure 1** and **Figure 2**.

The graph in **Figure 1** shows Schwefel's function, a standard benchmark numerical minimization problem. Schwefel's function is defined as:

$$f(x,y) = (-x * \sin(\sqrt{|x|})) + (-y * \sin(\sqrt{|y|}))$$

The function has a minimum value of -837.9658 when x = y = 420.9687. In **Figure 1** this minimum value is at the far-left corner of the graph. The function is deliberately designed to be difficult for optimization algorithms to solve because of the numerous false local minimum values.

The screenshot in **Figure 2** shows the output from a C# console application. After displaying some informational messages, the program sets eight parameters and then uses an evolutionary algorithm to search for the optimum solution. In this example, the algorithm found a best solution of (420.9688, 420.9687), which is extremely close to, but not quite equal to the optimal solution of (420.9687, 420.9687).

Evolutionary optimization algorithms model a solution as a chromosome in an individual. In high-level pseudo-code, the algorithm implemented in **Figure 2** is:

```
initialize a population of random solutions
determine best solution in population
loop
  select two parents from population
  make two children from the parents
  place children into population
  make and place an immigrant into population
  check if a new best solution exists
end loop
return best solution found
```

Code download available at code.msdn.microsoft.com/mag201206TestRun.

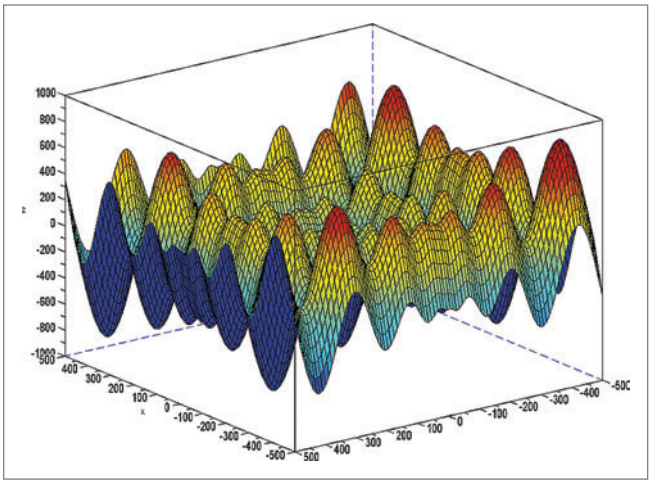


Figure 1 **Schwefel's Function**

In the sections that follow, I'll present all the code that generated the screenshot in **Figure 2**. You can access the code from code.msdn.microsoft.com/mag201206TestRun. This article assumes you have intermediate or advanced programming skills and at least a very basic understanding of genetic algorithms. I coded the demo program using C#, but you should be able to easily refactor the code into other languages such as Visual Basic .NET or IronPython.

> Evolutionary optimization algorithms model a solution as a chromosome in an individual.

## Program Structure

The overall program structure is shown in **Figure 3**. I used Visual Studio 2010 to create a new C# console application project named EvolutionaryOptimization. In the Solution Explorer window, I renamed Program.cs to EvolutionaryOptimizationProgram.cs, which automatically renamed class Program. I also deleted all the unneeded template-generated using statements.

In addition to the main program class, the EvolutionaryOptimization demo has three program-defined classes. Class Evolver contains most of the algorithm logic. Class Individual models a possible solution to the minimization problem. Class Problem

defines the function to be minimized, Schwefel's function in this case. An alternative design structure would be to place the Individual class inside the Evolver class.

## The Individual Class

The definition of class Individual begins:

```
public class Individual : IComparable<Individual>
{
  public double[] chromosome;
  public double fitness;
  private int numGenes;
  private double minGene;
  private double maxGene;
  private double mutateRate;
  private double precision;
  static Random rnd = new Random(0);
...
```

The class inherits from the IComparable interface so that arrays of Individual objects can be automatically sorted by their fitness. The class has eight data members. The chromosome array represents a possible solution to the target problem. Notice that chromosome is an array of double, rather than an array with some form of bit representation typically used by genetic algorithms. Evolutionary optimization algorithms are sometimes called real-valued genetic algorithms.

The fitness field is a measure of how good the chromosome-solution is. For minimization problems, smaller values of fitness are better than larger values. For simplicity, chromosome and fitness are declared with public scope so that they're visible to the logic in the Evolver class. A gene is one value in the chromosome array and the numGenes field holds the number of real values in a possible solution. For Schwefel's function, this value will be set to 2. With many numeric optimization problems, minimum and maximum values for each gene can be specified and these values are stored in minGene and maxGene. If these values are not known, minGene and maxGene can be set to double.MinValue and double.MaxValue. I'll explain the mutateRate and precision fields when I present the code that uses them.

The Individual class definition continues with the class constructor:

```
public Individual(int numGenes, double minGene, double maxGene,
  double mutateRate, double precision)
{
  this.numGenes = numGenes;
  this.minGene = minGene;
  this.maxGene = maxGene;
  this.mutateRate = mutateRate;
  this.precision = precision;
  this.chromosome = new double[numGenes];
  for (int i = 0; i < this.chromosome.Length; ++i)
    this.chromosome[i] = (maxGene - minGene) * rnd.NextDouble() + minGene;
  this.fitness = Problem.Fitness(chromosome);
}
```

The constructor allocates memory for the chromosome array and assigns random values in the range (minGene, maxGene) to each gene cell. Notice that the value of the fitness field is set by calling the externally defined Fitness method. Alternatively, you could pass into the constructor a reference to the Fitness method via a Delegate. The Mutate method is defined as such:

```
public void Mutate()
{
  double hi = precision * maxGene;
  double lo = -hi;
  for (int i = 0; i < chromosome.Length; ++i) {
    if (rnd.NextDouble() < mutateRate)
      chromosome[i] += (hi - lo) * rnd.NextDouble() + lo;
  }
}
```

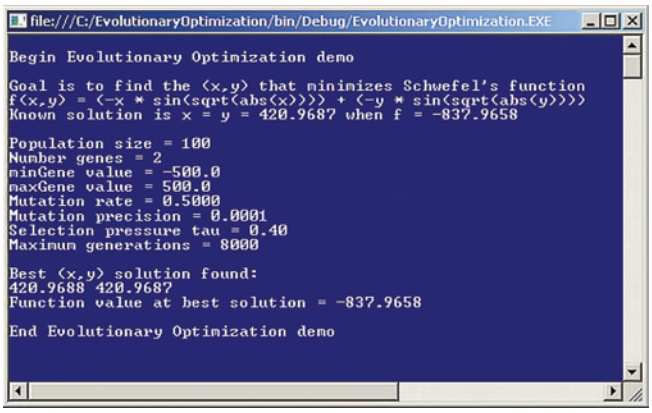



Figure 2 **Evolutionary Optimization Demo**

The mutation operation walks through the chromosome array, changing randomly selected genes to random values in the range (lo, hi). The range of values to assign is determined by the precision and maxGene member values. In the example in **Figure 2**, precision is set to 0.0001 and maxGene is set to 500. The highest possible value for a gene mutation is 0.0001 * 500 = 0.05, which means that if a gene is mutated, its new value will be the old value plus or minus a random value between -0.05 and +0.05. Notice that the precision value corresponds to the number of decimals in the solution; this is a reasonable heuristic to use for the precision value. The mutation rate value controls how many genes in the chromosome will be modified. One heuristic for the value of the mutateRate field is to use 1.0 / numGenes, so that on average one gene in the chromosome will be mutated every time Mutate is called.

The Individual class definition concludes with the CompareTo method:

```
...
public int CompareTo(Individual other)
{
  if (this.fitness < other.fitness) return -1;
  else if (this.fitness > other.fitness) return 1;
  else return 0;
}
}
```

The CompareTo method defines a default sort ordering for Individual objects, in this case from smallest fitness (best) to largest fitness.

## The Problem Class

The Problem class encapsulates the target problem for the evolutionary algorithm:

```
public class Problem
{
  public static double Fitness(double[] chromosome)
  {
    double result = 0.0;
    for (int i = 0; i < chromosome.Length; ++i)
      result += (-1.0 * chromosome[i]) *
        Math.Sin(Math.Sqrt(Math.Abs(chromosome[i])));
    return result;
  }
}
```

Because a chromosome array represents a possible solution, it's passed as an input parameter to the Fitness method. With arbitrary minimization problems, the target function to be minimized is usually called a cost function. In the context of evolutionary and genetic algorithms, however, the function is usually called a fitness function. Notice the terminology is a bit awkward because lower

Figure 3 **Overall Structure of the EvolutionaryOptimization Program**

```
using System;
namespace EvolutionaryOptimization
{
  class EvolutionaryOptimizationProgram
  {
    static void Main(string[] args)
    {
      try
      {
        int popSize = 100;
        int numGenes = 2;
        double minGene = -500.0; double maxGene = 500.0;
        double mutateRate = 1.0 / numGenes;
        double precision = 0.0001;
        double tau = 0.40;
        int maxGeneration = 8000;

        Evolver ev = new Evolver(popSize, numGenes, minGene, maxGene, mutateRate,
          precision, tau, maxGeneration);
        double[] best = ev.Evolve();

        Console.WriteLine("\nBest solution found:");
        for (int i = 0; i < best.Length; ++i)
          Console.Write(best[i].ToString("F4") + " ");

        double fitness = Problem.Fitness(best);
        Console.WriteLine("\nFitness of best solution = " + fitness.ToString("F4"));
      }
      catch (Exception ex)
      {
        Console.WriteLine("Fatal: " + ex.Message);
      }
    }
  }

  public class Evolver
  {
    // Member fields

    public Evolver(int popSize, int numGenes, double minGene, double maxGene,
      double mutateRate, double precision, double tau, int maxGeneration) { ... }
    public double[] Evolve() { ... }
    private Individual[] Select(int n) { ... }
    private void ShuffleIndexes() { ... }
    private Individual[] Reproduce(Individual parent1, Individual parent2) { ... }
    private void Accept(Individual child1, Individual child2) { ... }
    private void Immigrate() { ... }
  }

  public class Individual : IComparable<Individual>
  {
    // Member fields

    public Individual(int numGenes, double minGene, double maxGene,
      double mutateRate, double precision) { ... }
    public void Mutate() { ... }
    public int CompareTo(Individual other) { ... }
  }

  public class Problem
  {
    public static double Fitness(double[] chromosome) { ... }
  }
} // NS
```

values of fitness are better than higher values. In this example, the fitness function is completely self-contained. In many optimization problems, the fitness function requires additional input parameters, such as a matrix of data or a reference to an external data file.

## The Evolver Class

The Evolver class definition begins:

```
public class Evolver
{
  private int popSize;
  private Individual[] population;
  private int numGenes;
  private double minGene;
  private double maxGene;
  private double mutateRate;
  private double precision;
  private double tau;
  private int[] indexes;
  private int maxGeneration;
  private static Random rnd = null;
...
```

The popSize member holds the number of individuals in the population. Larger values of popSize tend to increase the algorithm's accuracy at the expense of speed. In general, evolutionary algorithms are much faster than ordinary genetic algorithms because evolutionary algorithms work on real values and don't incur the overhead of converting and manipulating bit representations. The heart of the Evolver class is an array of Individual objects named population. The tau and the indexes members are used by the Selection method, as you'll see shortly.

The Evolver class definition continues with the constructor definition shown in **Figure 4**.

The constructor allocates memory for the population array and then uses the Individual constructor to fill the array with individuals that have random gene values. The array named indexes is used by the Select method, which picks two parents. I'll explain indexes later, but notice the constructor allocates one cell per individual and sequentially assigns integers from 0 to popSize-1.

The Evolve method, listed in **Figure 5**, is surprisingly short.

The Evolve method returns the best solution found as an array of type double. As an alternative, you could return an Individual object where the chromosome holds the best solution found. The Evolve method begins by initializing the best fitness and best chromosomes to the first ones in the population. The method iterates exactly maxGenerations times, using gen (generation) as a loop counter. One of several alternatives is to stop when no improvement has occurred after some number of iterations. The Select method returns two good, but not necessarily best, individuals from the population. These two parents are passed to Reproduce, which creates and returns two children. The Accept method places the two children into the population, replacing two existing individuals. The Immigrate method generates a new random individual and places it into the population. The new population is then scanned to see if any of the three new individuals in the population is a new best solution.

The Select method is defined as:

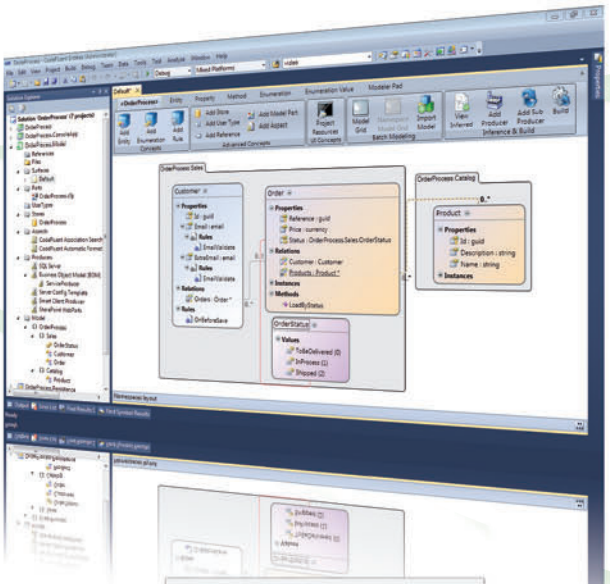
```
private Individual[] Select(int n)
{
  int tournSize = (int)(tau * popSize);
  if (tournSize < n) tournSize = n;
  Individual[] candidates = new Individual[tournSize];
  ShuffleIndexes();
  for (int i = 0; i < tournSize; ++i)
    candidates[i] = population[indexes[i]];
  Array.Sort(candidates);
  Individual[] results = new Individual[n];
  for (int i = 0; i < n; ++i)
    results[i] = candidates[i];
  return results;
}
```

# CodeFluent Entities

## LESS PLUMBING CODE, MORE FEATURES

CodeFluent Entities is a Visual Studio 2008/2010/11 integrated environment that allows you to model your business entities, and generate consistent foundation code, continuously, across all chosen layers (database, business tier, services, user interface).

Using this model-first approach, your business logic is decoupled from the technology and your foundations will automatically benefit from upcoming innovation.



- UML FREE
- TEMPLATE FREE
- FRAMEWORK FREE
- ORM FREE


CodeFluent Entities is designed for the .NET platform and empowers users to streamline developments on major Microsoft platforms and technologies such as C#, VB.NET, SQL Server, Azure, ASP.NET, WPF, WCF, JSON/REST Services, SharePoint, Silverlight,Oracle Database, MySQL and PostgreSQL.

Your application deserves rock-solid foundations, let CodeFluent Entities generate them, and keep the fun part for you! Focus on what makes the difference.

FREE FOR PERSONAL USE

**DOWNLOAD YOUR FREE LICENSE**

www.softfluent.com/landings_cfe_msdn

SoftFluent

**TOOLS FOR DEVELOPERS, BY DEVELOPERS**

SoftFluent is a software publisher providing solutions to help developers produce software code fluently, with users in more than 100 countries.

More information: *www.softfluent.com* - Contact: *info@softfluent.com*

CodeFluent Entities is a trademark of SoftFluent SAS. Other names may be trademark of their respective owners.

Figure 4 **Evolver Class Constructor**

```
public Evolver(int popSize, int numGenes, double minGene, double maxGene,
  double mutateRate, double precision, double tau, int maxGeneration)
{
  this.popSize = popSize;
  this.population = new Individual[popSize];
  for (int i = 0; i < population.Length; ++i)
    population[i] = new Individual(numGenes, minGene, maxGene,
      mutateRate, precision);

  this.numGenes = numGenes;
  this.minGene = minGene;
  this.maxGene = maxGene;
  this.mutateRate = mutateRate;
  this.precision = precision;
  this.tau = tau;

  this.indexes = new int[popSize];
  for (int i = 0; i < indexes.Length; ++i)
    this.indexes[i] = i;
  this.maxGeneration = maxGeneration;
  rnd = new Random(0);
}
```

Figure 5 **The Evolve Method**

```
public double[] Evolve()
{
  double bestFitness = this.population[0].fitness;
  double[] bestChomosome = new double[numGenes];
  population[0].chromosome.CopyTo(bestChomosome, 0);
  int gen = 0;
  while (gen < maxGeneration)  {
    Individual[] parents = Select(2);
    Individual[] children = Reproduce(parents[0], parents[1]);
    Accept(children[0], children[1]);
    Immigrate();

    for (int i = popSize - 3; i < popSize; ++i)  {
      if (population[i].fitness < bestFitness)  {
        bestFitness = population[i].fitness;
        population[i].chromosome.CopyTo(bestChomosome, 0);
      }
    }
    ++gen;
  }
  return bestChomosome;
}
```

The method accepts the number of good individuals to select and returns them in an array of type Individual. To minimize the amount of code, I've omitted normal error checking, such as verifying that the number of individuals requested is less than the population size. The Select method uses a technique called tournament selection. A subset of random candidate individuals is generated and the best *n* of them are returned. The number of candidates is computed into variable tournSize, which is some fraction, tau, of the total population size. Larger values of tau increase the probability that the best two individuals will be selected.

Notice that the precision value corresponds to the number of decimals in the solution.

Recall that the Evolver class has a member array named indexes with values 0..popSize-1. The ShuffleIndexes helper method rearranges the values in array indexes into random order. The top *n* of these random indexes are used to pick candidates from the population. The Array.Sort method then sorts the candidate individuals from smallest (best) fitness to largest. The top *n* individuals of the sorted candidates are returned. There are many different evolutionary selection algorithms. An advantage of tournament selection compared to most other techniques is that selection pressure can be tuned by modifying the tau parameter.

The ShuffleIndexes helper method uses the standard Fisher-Yates shuffle algorithm:

```
private void ShuffleIndexes()
{
  for (int i = 0; i < this.indexes.Length; ++i) {
    int r = rnd.Next(i, indexes.Length);
    int tmp = indexes[r];
    indexes[r] = indexes[i];
    indexes[i] = tmp;
  }
}
```

The Reproduce method is listed in **Figure 6**. The method begins by generating a random crossover point. The indexing is a little bit tricky, but child1 is created from the left part of parent1 and the right part of parent2. Child2 is created from the left part of parent2 and the right part of parent1. The idea is illustrated in **Figure 7**, where there are two parents with five genes and the crossover point is 2. A common design alternative is to use multiple crossover points. After the Individual children objects are created they are mutated and their new fitness is computed.

The Accept method places the two child individuals created by Reproduce into the population:

```
private void Accept(Individual child1, Individual child2)
{
  Array.Sort(this.population);
  population[popSize - 1] = child1;
  population[popSize - 2] = child2;
}
```

The population array is sorted by fitness, which places the two worst individuals in the last two cells of the array, where they're then replaced by the children. There are many alternative approaches

Figure 6 **The Reproduce Method**

```
private Individual[] Reproduce(Individual parent1, Individual parent2)
{
  int cross = rnd.Next(0, numGenes - 1);
  Individual child1 = new Individual(numGenes, minGene, maxGene,
    mutateRate, precision);
  Individual child2 = new Individual(numGenes, minGene, maxGene,
     mutateRate, precision);

  for (int i = 0; i <= cross; ++i)
    child1.chromosome[i] = parent1.chromosome[i];
  for (int i = cross + 1; i < numGenes; ++i)
    child2.chromosome[i] = parent1.chromosome[i];
  for (int i = 0; i <= cross; ++i)
    child2.chromosome[i] = parent2.chromosome[i];
  for (int i = cross + 1; i < numGenes; ++i)
    child1.chromosome[i] = parent2.chromosome[i];

  child1.Mutate(); child2.Mutate();
  child1.fitness = Problem.Fitness(child1.chromosome);
  child2.fitness = Problem.Fitness(child2.chromosome);
  Individual[] result = new Individual[2];
  result[0] = child1; result[1] = child2;
  return result;
}
```

to choosing which two individuals die. One possibility is to use a technique called roulette wheel selection to probabilistically select the two individuals to replace, where worse individuals have a higher probability of being replaced.

In this article, evolutionary optimization is used to estimate the minimum value of a mathematical equation.

The Immigrate method generates a new random individual and places it into the population just above the location of the two children that were just generated (immigration helps prevent evolutionary algorithms from getting stuck in local minima solutions; design alternatives include creating more than one immigrant and placing the immigrant at a random location in the population):

```
private void Immigrate()
{
  Individual immigrant =
    new Individual(numGenes, minGene, maxGene, mutateRate, precision);
  population[popSize - 3] = immigrant;
}
```

## Starting Point for Experimentation

In this article, evolutionary optimization is used to estimate the minimum value of a mathematical equation. Although evolutionary optimization algorithms can be used in this way, more often they're used to find the values for a set of numeric parameters in a larger optimization problem for which there's no effective deterministic algorithm. For example, when using a neural network to classify existing data in order to make predictions on future data, the main challenge is to determine a set of neuron weights and biases. Using an evolutionary optimization algorithm is one approach for estimating the optimal weight and bias values. In most cases, evolutionary optimization algorithms are not well-suited for use on non-numerical combinatorial optimization problems such as the Traveling Salesman Problem, where the goal is to find the combination of cities with the shortest total path length.

Evolutionary algorithms, like pure genetic algorithms, are meta-heuristics. This means they're a general framework and a set of conceptual guidelines that can be used to create a specific algorithm to solve a specific problem. So the example presented in this article can be viewed more as a starting point for experimentation and creating evolutionary optimization code rather than a fixed, static code base. ■

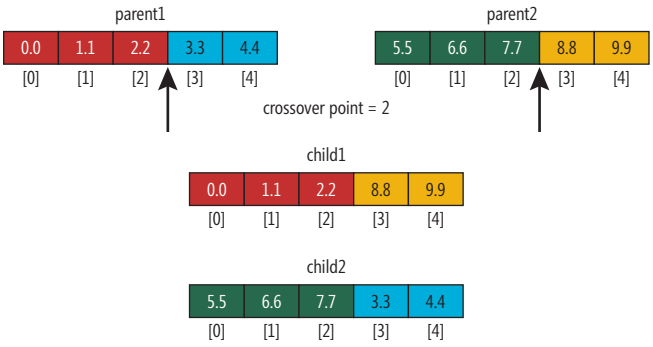



Figure 7 **The Crossover Mechanism**

**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. He's the author of ".NET Test Automation Recipes" (Apress, 2006), and can be reached at jammc@microsoft.com.*

**Thanks** to the following Microsoft technical expert for reviewing this article: *Anne Loomis Thompson*



TED NEWARD

# Talk to Me, Part 4: Feliza Gets Her Voice

We're sprinting to the finish line now. The first column in this series introduced Tropo (see msdn.microsoft.com/magazine/hh781028). The second part introduced Feliza, an F# assembly that practices some simple string parsing legerdemain to sound reasonably like a bad psychiatrist (see msdn.microsoft.com/magazine/hh852597). The third part wired Tropo up against the Feliza Web site (so Tropo was receiving its commands from the Web site rather than a script hosted on its own servers) so we could do the final wiring to get Feliza (see msdn.microsoft.com/magazine/hh975378). And now, we complete the wiring job and turn Feliza loose into the world.

Recall, from the last piece, that our dream of Feliza being able to take phone calls is, at the moment, limited by the Tropo API, but Feliza has no problems with SMS—at 425-247-3096—or with IM (using the registered Jabber handle of feliza@tropo.im), so we'll continue with that. Remember, Tropo can easily handle voice inputs if there's a bound set of options to choose from, so if we wanted to limit Feliza to fixed input, we could easily go back to voice. However, for Feliza's purposes, that doesn't strike me as a great user experience: "Hello! If you're depressed, say 'depressed'; if you're just feeling bad, say 'feeling bad'; if you're ready to kill an automated therapy system, say 'die, die, die' …"

Despite the limitation, Feliza can still help people (well, sort of), so let's continue. Recall that we need Feliza to stand behind a Web site that knows how to respond to the Tropo REST/JSON requests Tropo fires at us, so we need to be able to interpret the incoming JSON, hand it to Feliza and then generate the reply. This means we need an ASP.NET MVC structure to represent the incoming JSON, and one to represent the outgoing response.

Creating a Microsoft .NET Framework class that matches up against this (so ASP.NET MVC can do the heavy lifting of parsing the JSON) is pretty straightforward, as shown in **Figure 1**.

Not all of the properties shown in **Figure 1** will always be populated, but it provides a working exemplar of how to map the Tropo structures against a JSON object.

Thus, it would seem to be a pretty straightforward case to parse the incoming JSON (a "session" object), extract the text out of the initialText field, generate a response object and hand it back, as shown in **Figure 2**.

Unfortunately, we run into a limitation of the ASP.NET MVC JSON serializer because any null values it sees in the returned object will be turned into "null" values in the JSON-serialized response, and Tropo has some issues with null-valued JSON fields. Fortunately, we can fix this by using a custom JSON serializer (in this case, James Newton-King's excellent Json.NET serializer at bit.ly/a270u), which can be configured to not serialize null values. This changes the code slightly to return a custom ActionResult (gratefully obtained from bit.ly/1DVucR) instead of the standard one (see **Figure 3**).

Figure 1 **The .NET Class to Match Tropo Structures Against JSON Objects**

```
public class Session
{
  public string accountId { get; set; }
  public string callId { get; set; }
  public class From
  {
    public string id { get; set; }
    public string name { get; set; }
    public string channel { get; set; }
    public string network { get; set; }
  }
  public From from { get; set; }
  public class To
  {
    public string id { get; set; }
    public string name { get; set; }
    public string channel { get; set; }
    public string network { get; set; }
  }
  public To to { get; set; }
  public string id { get; set; }
  public string initialText { get; set; }
  public string timestamp { get; set; }
  public string userType { get; set; }
};
```

Figure 2 **Parsing the JSON**

```
[AcceptVerbs("GET", "POST")]
public ActionResult Index(Tropo.JSON.Session session)
{
  string incoming = session.initialText;
  object felizaResponse = new
  {
    tropo = new object[]
    {
      new
      {
        ask = new Tropo.JSON.Ask()
        {
          say = new Tropo.JSON.Say() {
          value = "I'm Feliza. What about '" + incoming +
            "' would you like to talk about?"
          },
          choices = new Tropo.JSON.Ask.Choices() {
            value = "[ANY]"
          }
        }
      }
    }
  };

  return Json(felizaResponse);
}
```

# PRECISELY PROGRAMMED FOR SPEED

## DynamicPDF—Comprehensive PDF Solutions for .NET Developers

ceTe Software's DynamicPDF products provide real-time PDF generation, manipulation, conversion, printing, viewing, and much more. Providing the best of both worlds, the object models are extremely flexible but still supply the rich features you need as a developer. Reliable and efficient, the high-performance software is easy to learn and use. If you do encounter a question with any of our components, simply contact ceTe Software's readily available, industry-leading support team.

DynamicPDF

WWW.DYNAMICPDF.COM

**TRY OUR PDF SOLUTIONS FREE TODAY!**

www.DynamicPDF.com/eval or call 800.631.5006 | +1 410.772.8620

ceTesoftware

Figure 3 **A Customized ActionResult**

```
public class JsonNetResult : ActionResult
{
  public Encoding ContentEncoding { get; set; }
  public string ContentType { get; set; }
  public object Data { get; set; }
  public JsonSerializerSettings SerializerSettings { get; set; }
  public Formatting Formatting { get; set; }
  public JsonNetResult() { SerializerSettings =
    new JsonSerializerSettings(); }
  public override void ExecuteResult(ControllerContext context)
  {
    if (context == null)
      throw new ArgumentNullException("context");
    HttpResponseBase response = context.HttpContext.Response;
    response.ContentType = !string.IsNullOrEmpty(ContentType) ?
      ContentType : "application/json";
    if (ContentEncoding != null)
      response.ContentEncoding = ContentEncoding;
    if (Data != null)
    {
      JsonTextWriter writer =
        new JsonTextWriter(response.Output) { Formatting = Formatting };
      JsonSerializer serializer =
        JsonSerializer.Create(SerializerSettings);
      serializer.Serialize(writer, Data);
      writer.Flush();
    }
  }
}
```

Then, in the Controller, we use this ActionResult instead of the JSON-based one, making sure to configure it to not send back null values:

```
[AcceptVerbs("GET", "POST")]
public ActionResult Index(Tropo.JSON.Session session)
{
  string incoming = session.initialText;
  object felizaResponse = // ...;

  JsonNetResult jsonNetResult = new JsonNetResult();
  jsonNetResult.SerializerSettings.NullValueHandling =
    NullValueHandling.Ignore;
  jsonNetResult.Data = felizaResponse;
  return jsonNetResult;
}
```

So at this point, for those readers playing with the code at home, the server is able to take incoming SMS messages, pick out the incoming text and generate a response back. You can probably guess what comes next.

Figure 4 **Generating a Response**

```
[AcceptVerbs("GET", "POST")]
public ActionResult Index(Tropo.JSON.Session session)
{
  object felizaResponse = new
  {
    tropo = new object[]
    {
      new
      {
        ask = new Tropo.JSON.Ask()
        {
          say = new Tropo.JSON.Say() {
            value = Feliza1.respond(session.initialText)
          },
          choices = new Tropo.JSON.Choices() {
            value = "[ANY]"
          }
        }
      }
    }
  };

  JsonNetResult jsonNetResult = new JsonNetResult();
  jsonNetResult.SerializerSettings.NullValueHandling =
    NullValueHandling.Ignore;
  jsonNetResult.Data = felizaResponse;
  return jsonNetResult;
}
```

## Pleased to Meet You, Feliza

It's time to complete the circuit by pulling in the Feliza F# binaries from the second column in this series. Either copy the compiled binaries over from the old code, or, if you prefer, create a (second) F# Library project in the ASP.NET MVC Tropo solution, mark the TropoApp project as depending on the Feliza project and copy over the code. (Make sure the ASP.NET MVC project also knows about the F# dependencies, particularly FSharp.Core.dll.)

Now, generating the response is as trivial as taking the incoming text, passing it in to Feliza1's *respond* method and capturing the results into the returned JSON, as shown in **Figure 4**.

It really is that easy—and if a phone is handy, pull it out and you'll see that texting Feliza results in a kind of conversation. She's not the world's smartest chat-bot, but with the F# core behind her, there's a lot we can do to improve her responses.

## Incorporating Voice or SMS into Applications

Feliza's done, and now the world can rest easy knowing that if anyone's staying up late, just wishing somebody would text them, Feliza's ready and waiting for them. Of course, it's not going to be a very satisfying conversation, because most of the time she won't have a clue about what she's reading.

The Tropo API has some interesting quirks, but clearly there's some significant power there. The twin models—the hosted scripts as well as the "Web API"—offer some interesting flexibility when thinking about how to incorporate voice, SMS or instant messaging into applications.

Feliza's Web site, too, could be enhanced in a number of ways—for example, for those who are more mobile-Web-minded, it would seem appropriate to create a set of static HTML5 pages using jQuery to hit Feliza with the typed-in text and append the responses to the page. In essence, Tropo offers a new set of "channels" through which to receive input, and ideally those channels would normalize their input into a single set of JSON structures to simplify the ASP.NET MVC endpoint code. And clearly the Tropo story would fit in well alongside other means of communication with customers or users. Other obvious channels that would be interesting to add are Twitter or Facebook, for examples. (Tropo has the ability to hook into Twitter, by the way, which would simplify that particular channel, but a Facebook channel would be something we'd have to build by hand.)

For now, other topics are pressing, so it's time to wave farewell to Feliza and move on. Don't worry, she'll still be there when you're feeling lonely. In the meantime …

Happy coding! ■

**Ted Neward** *is an architectural consultant with Neudesic LLC. He's written more than 100 articles, is a C# MVP and INETA speaker and has authored and coauthored a dozen books, including the recently released "Professional F# 2.0" (Wrox, 2010). He consults and mentors regularly. Reach him at ted@tedneward.com if you're interested in having him come work with your team, or read his blog at blogs.tedneward.com.*

# 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

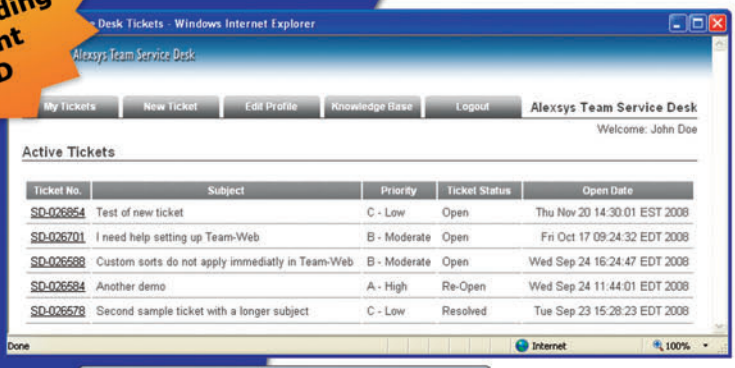
### Service Desk Features

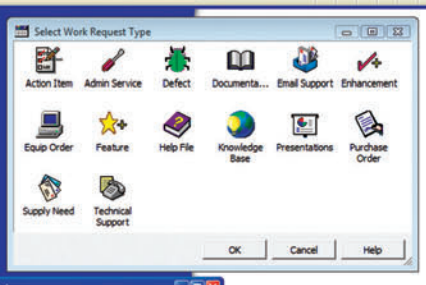
- Fully Secure
- Unlimited Users Self Registered or Active Directory
- Integrated into Your Web Site
- Fast/AJAX Dynamic Content
- Unlimited Service Desks
- Visual Service Desk Builder

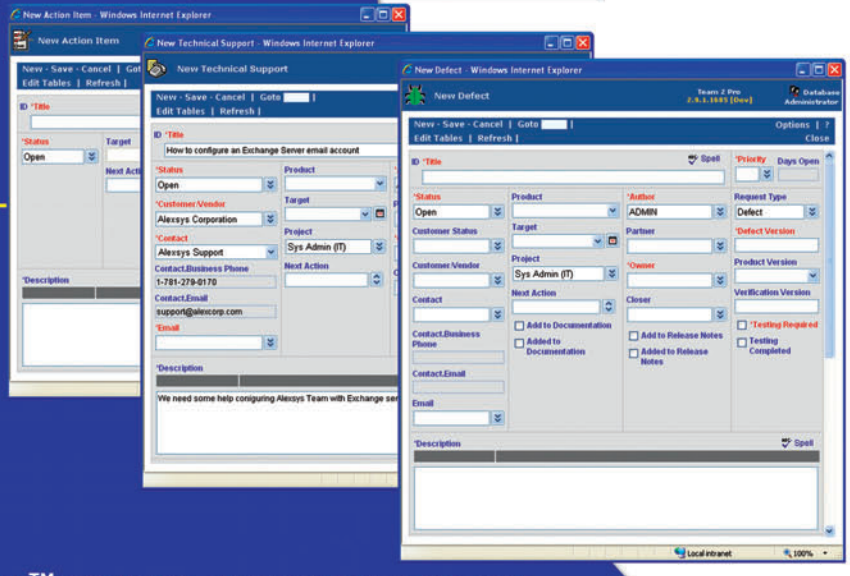
### Team 2 Features

- 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

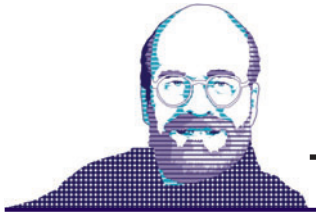



**Free Trial and Single User FreePack™ available at www.alexcorp.com.**
**FreePack™ includes a free single user Team Pro and Team-Web license.**
**Need more help? Give us a call at 1-888-880-ALEX (2539).**

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.





CHARLES PETZOLD

# Get Oriented with the Windows Phone Compass

We humans rarely use our senses entirely in isolation from each other: A combination of sight and hearing allows us to construct a mental image of our surroundings; a blend of taste, smell and touch affects our enjoyment when eating food; and we use a mix of touch, sight and hearing when playing a musical instrument.

It's the same with a smartphone: A smartphone can "see" through its camera lens, "hear" through its microphone, "feel" through its touchscreen, and know its location in the world using GPS and the orientation sensor. But start combining input from these sensors and there's no word to describe the result but *synergy*.

## The Accelerometer Problem

The accelerometer in Windows Phone is a good example of a sensor that provides some essential information but becomes much more valuable when combined with another sensor, specifically the compass.

The accelerometer hardware actually measures force, but as we know from physics, force equals mass times acceleration, so the accelerometer responds to any kind of acceleration. When the phone is held still, the accelerometer measures gravity and provides a 3D vector pointing toward the center of the earth. This vector is relative to a 3D coordinate system, as shown in **Figure 1**. This coordinate system is the same whether you're coding a Silverlight or XNA program, or running in portrait or landscape mode.

The Accelerometer class provides the acceleration vector in the form of a Vector3 value. This is an XNA type, so if you need to use it in a Silverlight program, you'll need a reference to the Microsoft.Xna.Framework assembly.

Although the acceleration vector allows a program running on the phone to determine the orientation of the phone relative to the Earth, it's missing some crucial information. Let me demonstrate what I'm talking about.

In the downloadable code for this column (available at code.msdn.microsoft.com/mag201206TouchAndGo) is a Visual Studio solution named 3DPointers that contains four XNA 3D projects, all of which look somewhat similar. The Accelerometer 3D program draws a 3D "pin" that floats in space in the direction of the accelerometer vector, as shown in **Figure 2**.

Any program that uses a Windows Phone sensor needs a reference to the Microsoft.Devices.Sensors assembly. Normally XNA programs on the phone run in landscape mode, and that can be an issue because it doesn't match the coordinate system used by the sensors. To make things easy for myself, I reoriented the XNA coordinate system for portrait mode in the constructor of the program's Game derivative:

```
graphics.IsFullScreen = true;
graphics.PreferredBackBufferWidth = 480;
graphics.PreferredBackBufferHeight = 800;
```

In Windows Phone 7.1, the sensor API has been changed a bit to provide consistency among the various sensors. The Accelerometer 3D program constructor uses this new API to create an Accelerometer instance that's saved as a field:

```
if (Accelerometer.IsSupported)
{
  accelerometer = new Accelerometer
  {
    TimeBetweenUpdates = this.TargetElapsedTime
  };
}
```

The default TimeBetweenUpdates property is 25 milliseconds; here it's set to the program's frame rate, which is 33 ms.

The program uses an override of the OnActivated method to start the Accelerometer:

```
if (accelerometer != null)
{
  try { accelerometer.Start(); }
  catch { }
}
```

Although it's difficult to imagine a scenario where the Start method will fail at this point, it's recommended that you put it in a try block. The compass is stopped in OnDeactivated:

```
if (accelerometer != null)
    accelerometer.Stop();
```

The program uses the LoadContent method to build 3D vertices for the "pin" and define a BasicEffect for storing the camera and lighting information. The pin is defined so that the base is at the origin and it extends one unit up the positive Y axis. The camera points directly at the origin from the positive Z axis.

The program's Update method then uses the accelerometer vector to define a world transform. This world transform effectively moves the pin relative to the origin. **Figure 3** shows the code.

The method obtains the Acceleration value directly from the Accelerometer object if the IsDataValid property is true. The pin must be rotated based on the angle between the Acceleration vector and the positive Y axis. The dot product of these two vectors provides that angle, while the axis of rotation is provided by the cross product.

But try this: Stand up and hold the phone in your hand at a particular angle. The pin points down. Now, while standing in place, turn around 360 degrees. While you're turning, the pin remains

Code download available at code.msdn.microsoft.com/mag201206TouchAndGo.


Figure 1 **The Phone's Sensor Coordinate System**


Figure 2 **The Accelerometer 3D Display**

in the same position (or nearly so). The Accelerometer can't discern when the phone is moving around an axis parallel with the acceleration vector. If you're writing an application that needs to know the complete 3D orientation of the phone, the accelerometer provides only part of what you need.

## The Compass to the Rescue

When Windows Phone was first released, it wasn't quite clear whether the phones even contained a compass. No one seemed to have a definitive answer. For application programmers, however, the situation was very simple: There was no programming interface for a compass, so even if it existed we couldn't use it.

The release of Windows Phone 7.1 clarified the issue: Although a Windows Phone device isn't required to have a compass, some Windows Phone devices always had one, including a smartphone running Windows Phone that I purchased in December 2010. Best of all, programmers can actually use this compass. Windows Phone 7.1 introduced a new Compass class that lets the application programmer know if the compass exists (through the static Compass.IsSupported property) and get access to its readings. Compass.IsSupported returns false on the Windows Phone emulator.

The basis of the phone's compass is a piece of hardware known as a magnetometer. This magnetometer is affected by any magnetic forces near the phone, including those that might be in speakers on your computer desk. If you keep these magnets away from the phone, the magnetometer will measure the strength and direction of Earth's magnetic field.

The Compass class provides data in the form of a CompassReading structure. The raw magnetometer data is available in a property named MagnetometerReading, which is another Vector3 relative to the coordinate system shown in **Figure 1**. In the absence of nearby magnets, this vector is aligned with Earth's magnetic field. The vector points in a northerly direction, of course, but in most locations in the northern hemisphere, it also points *into* the earth. If you hold the phone with the screen facing up, this vector will have a significant –Z component.

The 3DPointers solution contains a project named Magnetometer 3D that is similar to the Accelerometer 3D project, except it uses the Compass class rather than the Accelerometer class. **Figure 4** shows this program's display when the phone is held in my apartment in Manhattan with the screen facing up and the top of the phone pointing "uptown," that is, with the left and right sides of the phones aligned (as close as I could manage) with New York City's avenues. (A map reveals these avenues to run about 30 degrees east of north.)

The documentation states that the MagnetometerReading vector is in units of microteslas. On two of the commercial Windows Phone devices I own, this vector generally has a magnitude in the 30s, which is approximately correct. (The vector values shown in **Figure 4** have a composite magnitude of 43.) However, for a third phone I own, the MagnetometerReading vector is normalized and always has a magnitude of 1.

Now try this: Hold the phone so the Magnetometer 3D vector is nearly aligned with the positive Y axis of **Figure 1**. Now rotate the phone around an axis parallel to that vector. The vector stays the same (or nearly so), indicating that the phone's magnetometer doesn't have complete knowledge of the phone's orientation either.

Figure 3 **The Update Method in Accelerometer 3D**

```
protected override void Update(GameTime gameTime)
{
  if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
      this.Exit();

  if (accelerometer != null && accelerometer.IsDataValid)
  {
    Vector3 acceleration = accelerometer.CurrentValue.Acceleration;

    text = String.Format("X = {0:F3}\nY = {1:F3}\nZ = {2:F3}",
                         acceleration.X,
                         acceleration.Y,
                         acceleration.Z);
    textPosition = new Vector2(0, this.GraphicsDevice.Viewport.Height -
                               segoe24Font.MeasureString(text).Y);
    acceleration.Normalize();
    Vector3 axis = Vector3.Cross(acceleration, Vector3.UnitY);

    // Special case for magnetometer equal to (0, 1, 0) or (0, -1, 0)
    if (axis.LengthSquared() == 0)
        axis = Vector3.UnitX;
    else
        axis.Normalize();

    float angle = -(float)Math.Acos(Vector3.Dot(Vector3.UnitY, acceleration));
    basicEffect.World = Matrix.CreateFromAxisAngle(axis, angle);
  }
  else
  {
    basicEffect.World = Matrix.Identity;
    text = "";
  }
  base.Update(gameTime);
}
```

## Compass Headings

Normally when we're using a compass, we don't want a 3D vector aligned with Earth's magnetic field. Much more useful would be a 2D vector that's tangent to Earth's surface.

As you probably know, Earth's magnetic field does not coincide with the axis on which Earth spins. The direction of Earth's axis is called *geographic north* or *true north*, and this is the north used for maps and virtually all other purposes. Angles on a two-dimensional surface are often used for representing the direction of north. This direction is often called a *heading* or *bearing*.

The difference between magnetic north and true north varies all over the globe. In New York City, about 13 degrees must be subtracted from the magnetic bearing to get true bearing, but in Seattle, 21 degrees must be added to the magnetic bearing. The Compass class performs these calculations for you based on the phone's location. Besides the MagnetometerReading vector, CompassReading also supplies two properties of type double named MagneticHeading and TrueHeading. These are both angles in degrees ranging from 0 to 360 measured counterclockwise from the positive Y axis shown in **Figure 1**.

TrueHeading should always be interpreted as an approximate value, and even then it shouldn't be trusted entirely. On two of the phones I own, TrueHeading is usually about right, but on another phone, it's a good 70 degrees off.

I haven't been able to make sense of the MagneticHeading value. For a particular location, the difference between the values of TrueHeading and MagneticHeading should be a constant. For example, where I live, the value of TrueHeading minus MagneticHeading should be about –13 degrees. On all three of my phones, the difference between TrueHeading and MagneticHeading jumps sporadically between two values depending on the orientation of the phone. The difference is sometimes –12 (which is about correct), but mostly the difference is 92. These are the only two values of the difference that I've seen. On none of my phones is MagneticHeading consistent with the angle derived from the X and Y values of the MagnetometerReading vector.

In an XNA program, as you've seen, you can simply obtain a current value from a sensor during the Update method. When using a sensor class in a Silverlight program, you'll want to set a handler for the CurrentValueChanged event. You can then obtain a sensor reading object from the event arguments.

The downloadable code for this article contains two Silverlight programs—Arrow Compass and Dial Compass—that use the TrueHeading property to show the direction of north. All the graphics are defined in XAML. As with the XNA programs, these Silverlight programs create the Compass object in their constructors, but they also set a handler for the CurrentValueChanged property:

```
if (Compass.IsSupported)
{
  compass = new Compass();
  compass.TimeBetweenUpdates = TimeSpan.FromMilliseconds(33);
  compass.CurrentValueChanged += OnCompassCurrentValueChanged;
}
```

In Arrow Compass this handler sets the angle on a RotateTransform object attached to an arrow graphic:

```
this.Dispatcher.BeginInvoke(() =>
{
  arrowRotate.Angle = -args.SensorReading.TrueHeading;
  accuracyText.Text = String.Format("±{0}°",
                      args.SensorReading.HeadingAccuracy);
});
```

The CurrentValueChanged handler is called in a separate thread, so you'll need to use a Dispatcher to update any UI objects. Because the TrueHeading angle indicates a counterclockwise offset and Silverlight rotations are clockwise, the code uses the negative of the heading angle for the rotation.

The result is shown in **Figure 5**, again with the phone pointing uptown in New York City.

In Dial Compass, the arrow remains fixed while a dial rotates to indicate the direction, as shown in **Figure 6**. You use this variation if you want to know the direction that the top of the phone is pointing, rather than the direction of north relative to the phone.

If you run either of these two programs and hold the phone so that the screen faces the earth, the compass no longer works correctly. The rotation is opposite what it should be. If you need to correct for this, you'll want to use the positive value of the TrueHeading value when the Z value of the acceleration vector is positive.

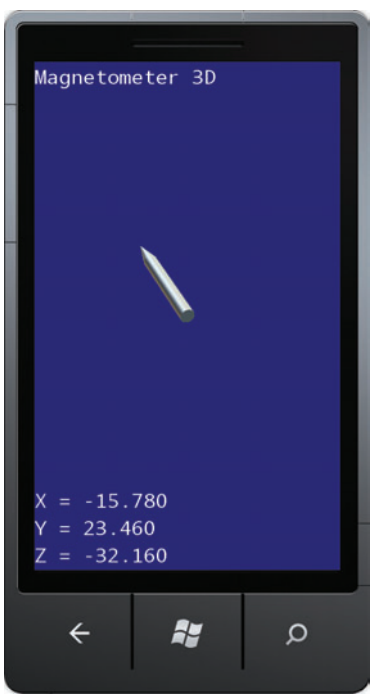



Figure 4 **The Magnetometer 3D Display**


Figure 5 **The Arrow Compass Display**

## Calibrating the Compass

In the lower-right corner, the Arrow Compass program displays the HeadingAccuracy property of the CompassReading value. In theory, this provides the accuracy of the heading values. In practice, I've seen HeadingAccuracy values ranging from 5 percent to 30 percent. (But I've only seen 5 percent on the phone that's way off!)

The Compass class also defines an event named Calibrate that's fired when the HeadingAccuracy value exceeds 20 percent.

Figure 6 **The Dial Compass Display**

Figure 7 **The Compass 3D Program**

You can perform a calibration maneuver to reduce this HeadingAccuracy: Hold the phone out from your body with the screen pointing left or right, and then sweep your arm in an infinity pattern several times. Some sample code provided with the MSDN tutorials (at bit.ly/yYrHrL) even has a graphic you can display to notify your users when the compass needs calibration.

## Combining Compass and Accelerometer

The phone's compass is a perfect example of a sensor that obviously has some utility by itself—particularly if you're lost in the woods—but which becomes much more valuable when combined with other sensors, especially the accelerometer. Together, these two sensors can provide a complete orientation of the phone in 3D space.

Indeed, Windows Phone 7.1 defines a new class that does precisely this. Besides providing the phone orientation in three different ways, the Motion class also incorporates information from the new Gyroscope class, if one is present on the phone.

Moreover, the Motion class does additional work by smoothing out the data from the Accelerometer and Compass. If you've been running the programs presented so far, you might have noticed significant jitter in the data from these classes. That jitter is all gone from the Motion class.

However, because I'm the type of person who enjoys a good challenge, I thought I'd take a shot at combining the Accelerometer and Compass data "manually," and I must admit that the experience left me with a much deeper appreciation of the Motion class!

The Compass 3D program displays four differently colored pins arranged in a circle to point north (silver), east (red), south (green) and west (blue). The program attempts to display this plane of pins parallel to Earth's surface and oriented correctly toward the four points of the compass.

The strategy I took was to derive Euler angles. These are three angles representing rotation around the X, Y and Z axes, and together they describe an orientation in 3D space. In flight dynamics, the three angles are labeled pitch, roll and yaw. From the perspective of an aircraft, pitch indicates whether the nose is up or down and by how much, while roll indicates any banking of the plane to the right or left. These rotation angles can be visualized as relative to two axes: pitch is rotation around an axis that extends through the wings, while roll is based on an axis from the front of the plane to the back. Yaw is rotation around an axis perpendicular to Earth's surface and indicates the compass heading for the plane.

To visualize these angles with respect to the phone's coordinate system in **Figure 1**, imagine yourself riding the phone like a magic carpet, sitting on the screen with the top of the phone to your front and the three buttons to your rear. Pitch is rotation around the X axis, roll is rotation around the Y axis and yaw is rotation around the Z axis.

Calculating roll and pitch from the acceleration vector turned out to be fairly easy and involves standard formulas:

```
float roll = (float)Math.Asin(-acceleration.X);
float pitch = (float)Math.Atan2(acceleration.Y, -acceleration.Z);
```

With the phone sitting flat on a table with its screen facing up, both roll and pitch are zero. Roll ranges from $-\pi/2$ to $\pi/2$, and the values go back to zero as the phone is turned facedown. Pitch ranges from $-\pi$ to $\pi$ with the maximum values when the phone is facedown. With the phone's screen facing up, yaw should be more or less the same value as the TrueHeading property from the Compass, but converted to radians for XNA purposes:

```
float yaw = MathHelper.ToRadians((float)compass.CurrentValue.TrueHeading);
```

However, as you saw with the two Silverlight compass programs, TrueHeading stops working correctly when the phone is turned with its screen toward Earth, so yaw needs to be corrected for that. After some theoretical and empirical excursions into the subject, I left it as is, and constructed a world transform from the three angles:

```
basicEffect.World = Matrix.CreateRotationZ(yaw) *
                    Matrix.CreateRotationY(roll) *
                    Matrix.CreateRotationX(pitch);
```

The results are shown in **Figure 7**.

I've also included an Orientation 3D program that obtains these three angles from the Motion class. You can see for yourself how much smoother the results are (in addition to working better when the phone is upside down).

The Motion class is too important an addition to the sensor API for just this single program. As you'll see in the next installment of this column, the class can actually serve as a portal into a 3D world. ■

**CHARLES PETZOLD** *is a longtime contributor to* MSDN Magazine. *His Web site is charlespetzold.com.*

**THANKS** to the following technical expert for reviewing this article: *Drew Batchelor*



DAVID S. PLATT

# The Silent Majority: Why Visual Basic 6 Still Thrives

Microsoft recently extended "It Just Works" compatibility for Visual Basic 6 applications through the full lifetime of Windows 8 (see this month's Editor's Note, "Old Soldiers Never Die," on p. 4). Visual Basic 6 first shipped in 1998, so its apps will have at least 24 years of supported lifetime. Contrast that with the Microsoft .NET Framework 1.0 (2002), which is incompatible with Windows 7 (2009).

A student of mine named Eric once joked that Visual Basic 6 was "the un-killable cockroach" in the Windows ecosystem. That analogy goes deeper than you might think. Cockroaches are successful because they're simple. They do what they need to do for their ecological niche and no more. Visual Basic 6 did what its creators intended for its market niche: enable very rapid development of limited programs by programmers of lesser experience. It was never meant for heavy-duty coders developing complex applications.

Visual Basic 6 accomplished its goals by abstracting away the complexity of the underlying Windows OS. Simple things were very simple to accomplish. On the other hand, complex things, such as dealing with threads, were impossible. My rule of thumb for Visual Basic 6 was: if I couldn't do it within 10 minutes, I couldn't do it at all.

> But the rapid (and therefore cheaper) development of limited (and therefore cheaper) applications by lower-skilled (and therefore cheaper) personnel is an important solution to a very large class of problems.

Another key to the success of Visual Basic 6 was the much shorter learning curve demanded by its limited feature set. Learning to drive a bus takes much less time than learning to fly a fighter jet. Becoming a good Visual Basic 6 programmer took much less time than becoming a good C++ programmer, the primary alternative at the time.

When Microsoft made Visual Basic .NET "a full-fledged language," the company loaded it up with all the power and concomitant complexity that C# has—threads, background operations and inheritance, to name just a few. It therefore required the same skill set as C# programming, the same learning curve and the same experience.

The people at Microsoft did that because that's what they thought they heard the Visual Basic 6 community demanding. But Visual Basic 6 programmers epitomize the "silent majority," a term popularized by U.S. President Richard Nixon in 1969 to describe his non-protesting, non-counterculture supporters in those turbulent years. Almost all Visual Basic 6 programmers were content with what Visual Basic 6 did. They were happy to be bus drivers: to leave the office at 5 p.m. (or 4:30 p.m. on a really nice day) instead of working until midnight; to play with their families on weekends instead of trudging back to the office; to sleep with their spouses instead of pulling another coding all-nighter and eating cold pizza for breakfast. They didn't lament the lack of operator overloading or polymorphism in Visual Basic 6, so they didn't say much.

The voices that Microsoft heard, however, came from the 3 percent of Visual Basic 6 bus drivers who actively wished to become fighter pilots. These guys took the time to attend conferences, to post questions on CompuServe forums, to respond to articles. Not content to merely fantasize about shooting a Sidewinder missile up the tailpipe of the car that had just cut them off in traffic, they demanded that Microsoft install afterburners on their busses, along with missiles, countermeasures and a head-up display. And Microsoft did.

But giving Visual Basic .NET to the Visual Basic 6 community was like raising a coyote as a domestic dog, then releasing him into the woods, shouting, "Hunt for your dinner as God intended, you magnificent, wild creature!" Most of them said, "Heck with that. I'm staying on my nice warm cushion by the fire while you open me a can of Alpo." And Visual Basic 6 kept right on going.

Visual Basic 6 was not without faults, of course. OnError Resume Next? If one thing croaks, just keep right on going and see what happens? Probably not the best idea. But the rapid (and therefore cheaper) development of limited (and therefore cheaper) applications by lower-skilled (and therefore cheaper) personnel is an important solution to a very large class of problems.

LightSwitch is now trying to fill this niche, with mixed reviews (see bit.ly/n9crJj). It is, at best, a decade late.

The things that Visual Basic 6 did still need doing. Until and unless Microsoft brings out another tool that does these things, Visual Basic 6 will keep scuttling around. I'll bet you a beer that Microsoft has to extend Visual Basic 6 support through Windows 9 and 10. ■

**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.*

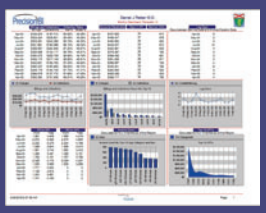
GrapeCity PowerTools

# Reporting at its best!

**Over 15 years experience in creating the leading .NET Reporting Tool for Silverlight, Windows Forms, ASP.NET, and Windows Azure.**

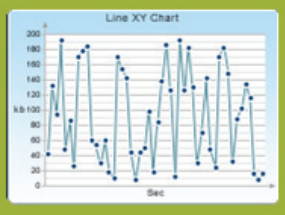
"I've used ActiveReports for a long time...It was easy to use right out of the box, really easy to configure, easy to pull data into.''

Todd Miranda
President
NxtDimension Solutions



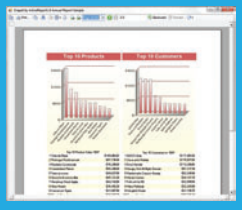
"We can actually handle thousands and thousands of customers and build very, very large hundred-page reports."

Kevin Grohoske
Director of Software Engineering
Bit-Wizards




"It just works!"

Carl Franklin
Host, .NET Rocks!



"For all of our Windows applications we always used Active Reports. It's easy, friendly and simple to use for developers."

Dave Noderer
CEO
Computer Ways, Inc.

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

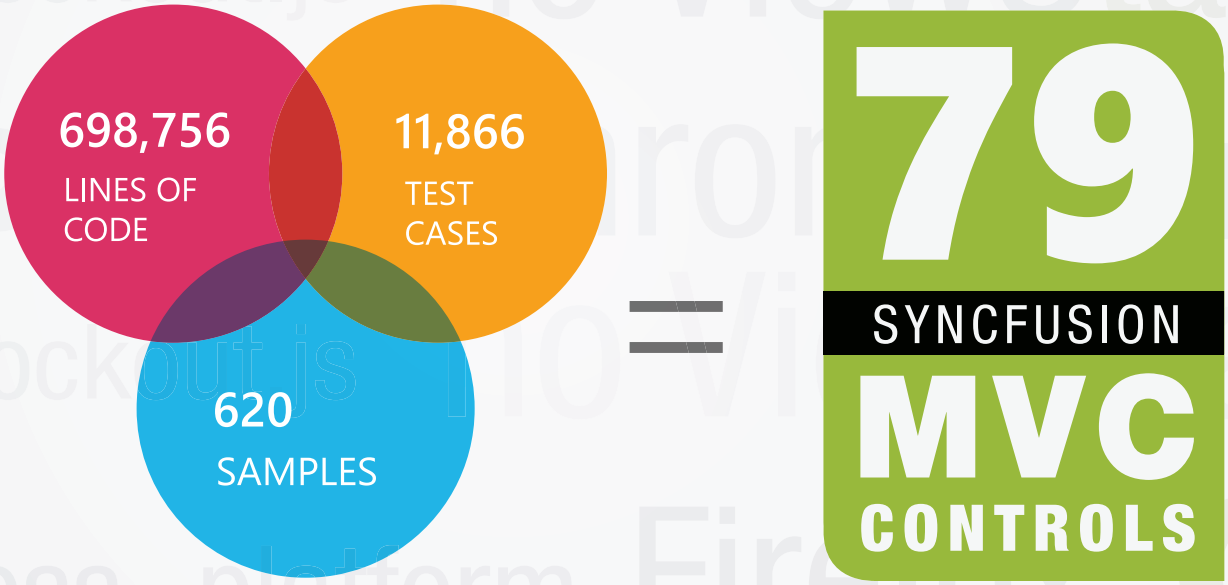
Dani Diaz
Developer Evangelist
Microsoft

Download your FREE Trial Today!

GCPowerTools.com/ActiveReports

ActiveReports

GCPowerTools.com • GvTv.GCPowerTools.com




As an MSDN subscriber, you are eligible for an exclusive 60-day trial of our MVC components and controls. That's more than one new control to test for each day of your trial. Download today!

syncfusion.com/60

Syncfusion®