



Enterprise Reporting in Minutes

# Introducing Report Server

Shipping as part of the Universal Subscription, the DevExpress Report Server is a powerhouse reporting platform: A combination of a high-performance server and an elegant client which fully exploits all the key features of the award-winning DevExpress End User Report Designer.



Download your  
30-day trial at  
[www.DevExpress.com](http://www.DevExpress.com)

## DXv2

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

Copyright 1998-2013 Developer Express, Inc. All rights reserved. All trademarks are property of their respective owners.



# msdn magazine

# C++

**A Tour of C++/CX.....30**

<b>A Tour of C++/CX</b> <b>Thomas Petchel</b> .....	<b>30</b>
<b>Exploring the JavaScript API for Office: Data Binding and Custom XML Parts</b> <b>Stephen Oliver and Eric Schmidt</b> .....	<b>40</b>
<b>Building and Validating Windows Store Apps with Team Foundation Service</b> <b>Thomas LeBrun</b> .....	<b>52</b>
<b>A Treasure Hunt Through ALM Readiness</b> <b>ALM Rangers</b> .....	<b>60</b>
<b>Classification and Prediction Using Adaptive Boosting</b> <b>James McCaffrey</b> .....	<b>68</b>

## COLUMNS

### CUTTING EDGE

Essential Facebook  
Programming:  
Widgets and Timeline  
Dino Esposito, page 6

### DATA POINTS

Why Does Entity Framework  
Reinsert Existing Objects  
into My Database?  
Julie Lerman, page 12

### WINDOWS AZURE INSIDER

NoSQL Data in the Cloud  
with Windows Azure Tables  
Bruno Terkaly and  
Ricardo Villalobos, page 18

### THE WORKING PROGRAMMER

Capturing Important  
Business Logic  
Ted Neward, page 76

### MODERN APPS

Power Your Modern Apps with  
Windows Azure Mobile Services  
Rachel Appel, page 78

### DIRECTX FACTOR

Streaming and Manipulating  
Audio Files in Windows 8  
Charles Petzold, page 82

### DON'T GET ME STARTED

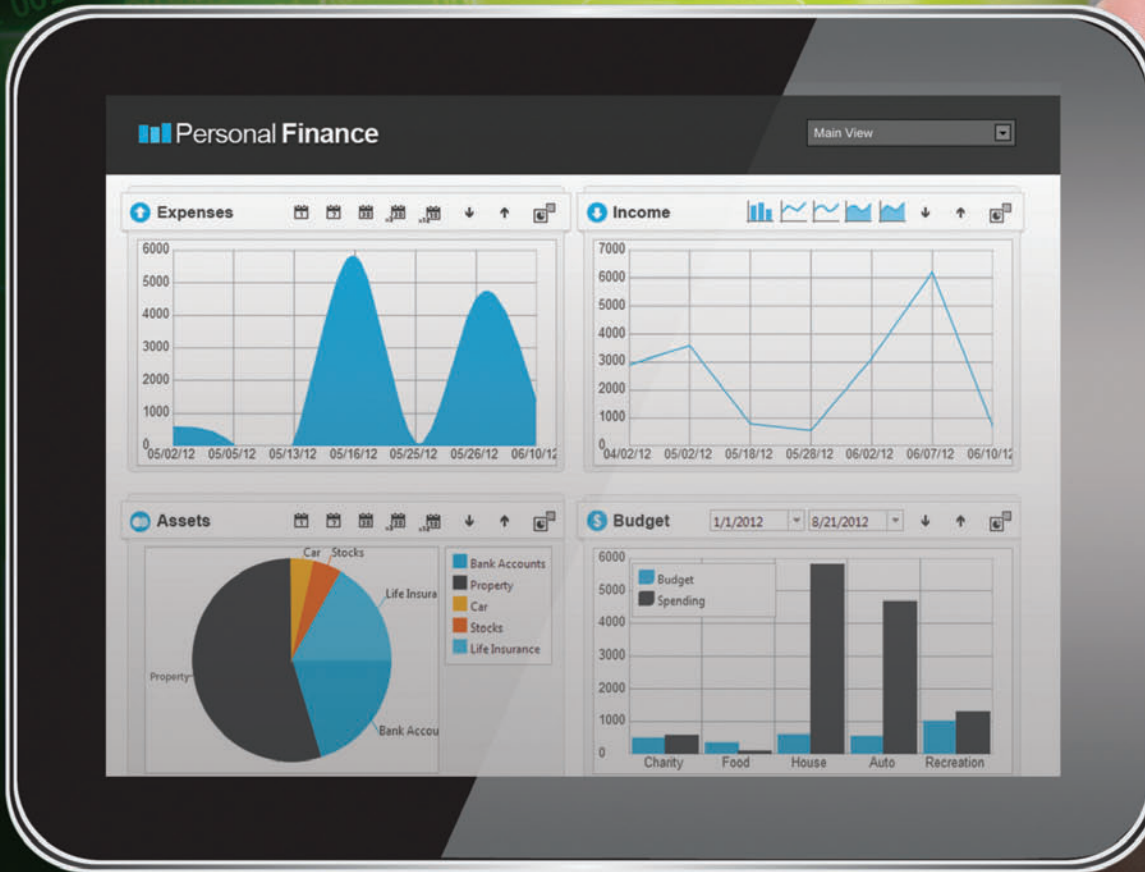
Coining Currency  
David Platt, page 88

Compatible with  
Microsoft® Visual Studio® 2012

San Francisco

Drop Filter Fields here

Country	Population	GDP Per Capita	Life Expectancy
Portugal	11	11433	79
Romania	23	2845	73
Serbia	11	1810	74
Slovak Republic	6	8591	75
Slovenia	3	13784	78
Spain	43	16306	81
Sweden	19	32338	83
Switzerland	8	37872	82
Lithuania	47	1136	68
United Kingdom	61	28955	79
Belgium	10	17766	77



# BRILLIANT UX

At Your Fingertips



Download your free trial  
[infragistics.com/EXPERIENCE](http://infragistics.com/EXPERIENCE)



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

Copyright 1996-2013 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.





# dtSearch®

## Instantly Search Terabytes of Text

- 25+ fielded and full-text search types
- dtSearch's **own document filters** support "Office," PDF, HTML, XML, ZIP, emails (with nested attachments), and many other file types
- Supports databases as well as static and dynamic websites
- Highlights hits in all of the above
- APIs for .NET, Java, C++, SQL, etc.
- 64-bit and 32-bit; Win and Linux

"lightning fast" Redmond Magazine

"covers all data sources" eWeek

"results in less than a second" InfoWorld

hundreds more reviews and developer case studies at [www.dtsearch.com](http://www.dtsearch.com)

### dtSearch products:

- ◆ Desktop with Spider
- ◆ Web with Spider
- ◆ Network with Spider
- ◆ Engine for Win & .NET
- ◆ Publish (portable media)
- ◆ Engine for Linux
- ◆ Document filters also available for separate licensing

*Ask about fully-functional evaluations*

The Smart Choice for Text Retrieval® since 1991

[www.dtSearch.com](http://www.dtSearch.com) 1-800-IT-FINDS

# msdn

magazine

APRIL 2013 VOLUME 28 NUMBER 4

**BJÖRN RETTIG** Director

**MOHAMMAD AL-SABT** Editorial Director/[mmeditor@microsoft.com](mailto:mmeditor@microsoft.com)

**PATRICK O'NEILL** Site Manager

**MICHAEL DESMOND** Editor in Chief/[mmeditor@microsoft.com](mailto: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

**SENIOR CONTRIBUTING EDITOR** Dr. James McCaffrey

**CONTRIBUTING EDITORS** Rachel Appel, Dino Esposito, Kenny Kerr, Julie Lerman, Ted Neward, Charles Petzold, David S. Platt, Bruno Terkaly, Ricardo Villalobos

### Redmond Media Group

**Henry Allain** President, Redmond Media Group

**Michele Imgrund** Sr. Director of Marketing & Audience Engagement

**Tracy Cook** Director of Online Marketing

**Irene Fincher** Audience Development Manager

ADVERTISING SALES: 818-674-3416/[dlbianca@1105media.com](mailto:dlbianca@1105media.com)

**Dan LaBianca** Vice President, Group Publisher

**Chris Kourtoglou** Regional Sales Manager

**Danna Vedder** Regional Sales Manager/Microsoft Account Manager

**Jenny Hernandez-Asandas** Director, Print Production

**Serena Barnes** Production Coordinator/[msdnadproduction@1105media.com](mailto:msdnadproduction@1105media.com)

### 1105 MEDIA

**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](mailto: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](http://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](mailto: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](mailto:1105reprints@parsintl.com), [www.magreprints.com/QuickQuote.asp](http://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, Attn: Jane Long. Phone: 913-685-1301; E-mail: [jl@meritdirect.com](mailto:jl@meritdirect.com); Web: [www.meritdirect.com/1105](http://www.meritdirect.com/1105)

All customer service inquiries should be sent to [MSDNmag@1105service.com](mailto:MSDNmag@1105service.com) or call 847-763-9560.



Printed in the USA

.NET, WIN 32/64, WinRT, HTML5, iOS, ANDROID, OS X & LINUX



The world's leading Imaging SDK  
**NOW RUNS ANYWHERE**

DICOM DATA SET

PACS CLIENT & SERVER

2D & 3D VIEWERS

WINDOW LEVEL

CUSTOM LAYOUT

DIAGNOSTIC TOOLS

DICOM ANNOTATIONS

MEDICAL IMAGE  
PROCESSING

PRINT TO PACS

ZERO FOOTPRINT  
VIEWERS

RICH CLIENT VIEWERS

RUNS ON DESKTOP,  
MOBILE & TABLET

COMPREHENSIVE IMAGING SDK FOR C++, C#, VB, JavaScript, OBJECTIVE-C & JAVA







## Patterns in Practice

Over the past several months *MSDN Magazine* has welcomed a pair of new columns—Rachel Appel's Modern Apps and Bruno Terkaly's and Ricardo Villalobos' Windows Azure Insider. We've also seen Charles Petzold rebrand his column as DirectX Factor, reflecting his shift to exploring the DirectX infrastructure of the Windows Runtime. What you may not know is that we've also been busy on the Web site. In January we introduced a new monthly online column called Patterns in Practice, written by veteran *MSDN Magazine* author Peter Vogel.

As the name of the column suggests, Patterns in Practice explores the value and potential of design patterns by applying them in working scenarios. In his inaugural column, "Adding Functionality to an Object" ([msdn.microsoft.com/magazine/jj890759](http://msdn.microsoft.com/magazine/jj890759)), Vogel looks at an application managing sales orders, and how the client can dynamically add permitted functionality to an object as it's needed. Vogel explains that his columns will present a business problem and discuss a few alternative solutions before diving into, as he writes, "a solution that addresses the problem in a testable/maintainable way, based on some design pattern." From there, readers can expect to follow along as Vogel builds out the design and implements the solution.

I asked Vogel why he wanted to focus specifically on design patterns. His response:

"I keep working with programmers who are trying to address the '-ities' that design patterns address: reusability, maintainability, extensibility, testability. But these developers don't look to the already existing solutions that design patterns provide, because they don't see design patterns as sources of useful inspiration or direction. They see patterns as being more like straightjackets: some guy yelling at you that 'You're doing it wrong!' This is compounded by many of the design pattern examples being about things that most business application developers would never build—editors, for instance.

"I want to show that design patterns should be as much a part of a developer's toolkit as relational database design or structured programming. Design patterns are, to me, all about moving from 'thinking in procedural code' to 'thinking in objects.' This column should demonstrate that design patterns, like the three

levels of data normalization, provide very helpful answers to some very common problems."

The fruits of this effort are already visible in the energetic back-and-forth in the comments section of the first Patterns in Practice column, and are shaping the direction of Vogel's coverage today. Vogel says he adjusted the design of his object model—presented in detail in his February column, "Data Design for Adding Functionality to a Class" ([msdn.microsoft.com/magazine/jj984634](http://msdn.microsoft.com/magazine/jj984634))—based on compelling arguments made in response to the first Patterns in Practice column.

"While I'm always resistant when people disagree with me, I do try to generate questions that will resolve the discussion one way or another," Vogel says. "That lets me go out and look for the answers to those questions and apply the evidence instead of just stomping my feet or falling back on 'principles.'"

Vogel says he sees several common mistakes when it comes to working with patterns, starting with developers who fail to take advantage of patterns where they would be truly useful. "Developers end up spending time reinventing the wheel and ending up with an oval when a circle would have been a much better choice," he says.

Vogel continues by noting that modern toolsets make common patterns easy to implement, yet many developers aren't aware of the available resources. Finally, he says, developers can run into the problem of misdiagnosis—they either misunderstand what a design pattern is intended to address or misdiagnose the problem they're trying to solve.

In the months to come, you can expect Vogel's Patterns in Practice column to explore the observer pattern and how a variation of it is implemented in SignalR for Web-based and service-oriented architecture (SOA) applications. Vogel says the columns will show how changing technology sets can make some patterns more attractive in an environment where the pattern would, as he says, "otherwise be discarded as un-implementable." Also look for a case study built around the decorate pattern.

Do you have a concept or pattern you'd like to see Vogel explore in his column? Write me at [mmeditor@microsoft.com](mailto:mmeditor@microsoft.com) and let us know!

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

© 2013 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.

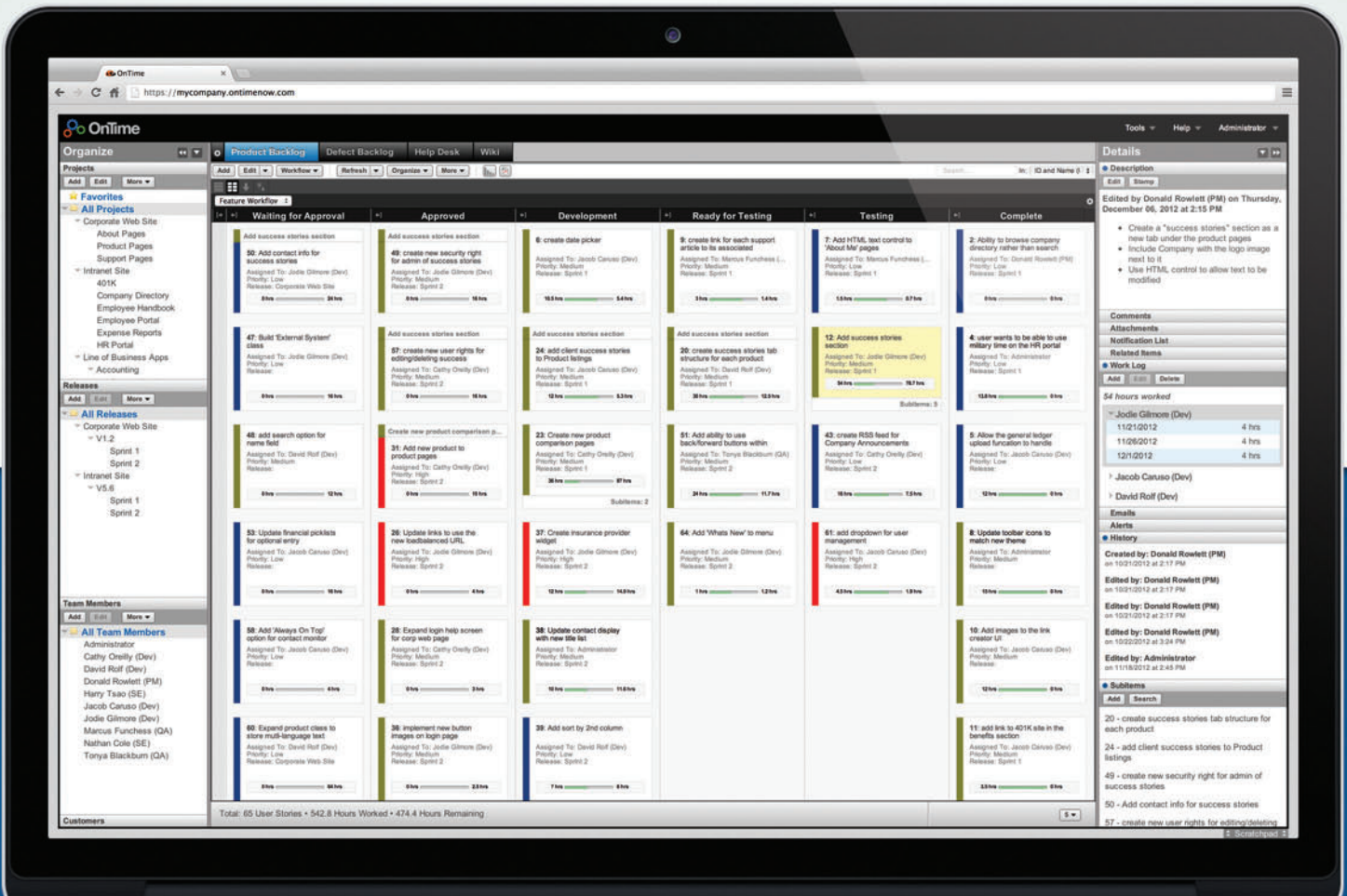
A listing of Microsoft Corporation trademarks can be found at [microsoft.com/library/toolbar/3.0/trademarks/en-us.mspx](http://microsoft.com/library/toolbar/3.0/trademarks/en-us.mspx). Other trademarks or trade names mentioned herein are the property of their respective owners.

*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.



# OnTime Scrum

Agile project management  
& bug tracking software



## Take control of your backlog now!

We help software developers ship great software on time.

The **OnTime Card View** is the ideal planning board tool for Kanban or Scrum teams. It adds a whole new dimension to user story management, bug tracking and workflow automation.

Learn more about Card View and the many other features of OnTime Scrum that your dev team will love.

[OnTimeNow.com/MSDN](http://OnTimeNow.com/MSDN)

**\$7** per user  
per month

**\$100** per year, per team  
up to 10 users

special small-team pricing



800.653.0024 • [www.ontimenow.com](http://www.ontimenow.com) • [www.axosoft.com](http://www.axosoft.com) • @axosoft





# Essential Facebook Programming: Widgets and Timeline

After covering some basic aspects of Facebook programming in previous columns, I'll now discuss tools and techniques to view and retrieve content from a Facebook wall in order to share it through other means and catalog it for something else, such as business intelligence (BI) analysis.

Not all companies have the same level of interest in the world of social communities as does Facebook. One lesson I've learned, however, is that in all companies, periodically an internal department—usually marketing—ends up with a strong interest in getting closer to customers and, maybe more important, in having customers get closer to the company. A Facebook fan page is one of the tools to attract contacts, and the number of Likes and the level of activity on the page can measure the success of the initiative.

Where does programming fit in? To keep a Facebook page alive and kicking and to stimulate user activity that increases the number of “people talking about this,” you need to post interesting content—and frequently. Sometimes the company can afford a staff of people just researching and creating content for the Facebook page. Sometimes, instead, good content for the Facebook page comes straight from the regular flow of company business. In this case, it would be a bit problematic for employees doing their regular jobs to reserve

extra time to report on a Facebook page what they're doing. Imagine, for example, that news is posted to the Web site. The internal workflow might entail preparing the text, getting it approved, publishing it in the internal system and waiting for the content management system to make it live on the site. If the same news should be published to Facebook, too, most of the time the same person opens the Facebook page as an admin and just posts content manually. It often works this way today, but it's not an approach that scales. This is just where Facebook programming fits in.

## Beyond Posting

In recent installments of this column, I addressed the main topic of posting to a Facebook wall and the basics of building a Web site and a desktop application that can interact with the Facebook account of the user (you can see all my columns at [bit.ly/hBNZAO](http://bit.ly/hBNZAO)). For an enterprise scenario where the target wall is the company's fan page, the approach isn't really different. All that changes is the account that receives posts.

So the first step toward Facebook programming is definitely finding a way to post content to specific walls in an automated way under the control of your software.

Over time, the content shared to the company's fan page, which typically includes marketing communications, becomes a useful resource for the company. It becomes valuable information that the company might want to retrieve and further share or analyze. And this is another great fit for Facebook programming.

## The Like Box Widget

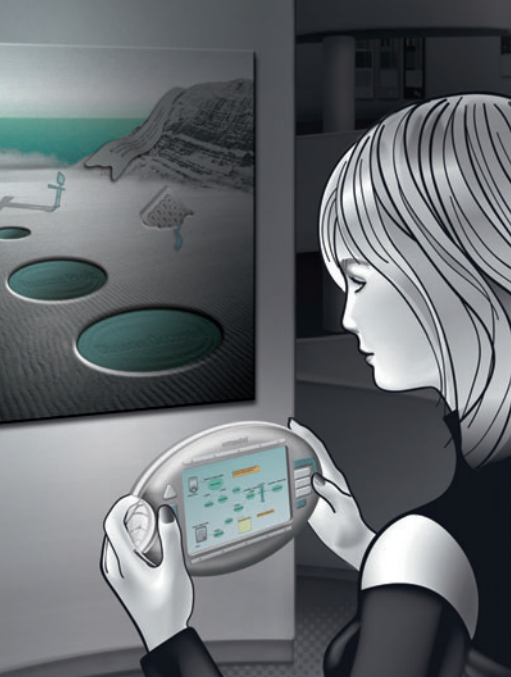
A quick and simple way to add some Facebook content to your site is through the Like Box widget. The widget lists recent posts made on a Facebook page as well as an optional list of users who like the page. For Web sites interested in using content published to Facebook, this

Figure 1 Parameters to Configure the Facebook Like Box

Parameters	Description
href	Indicates the URL of the Facebook fan page to feature in the Like Box.
width	Indicates the desired width in pixels for the plug-in. The default width is 300 pixels.
height	Indicates the desired height in pixels of the plug-in. The default height is not fixed and depends on whether faces and streams are to be displayed. In general, you should allow 500 pixels for it if you intend to have a stream of news.
colorscheme	Indicates the color scheme for the plug-in. You have only two options: light or dark.
show_faces	Indicates whether the plug-in should display profile photos. The default is true.
stream	Indicates whether the plug-in should display the latest posts from the page's wall.
header	Boolean parameter, hides or displays the default Facebook header at the top of the plug-in.
border_color	Indicates the border color of the plug-in.
force_wall	If the page refers to a place, then this parameter indicates whether the stream should contain posts from the place's wall or just check-ins from friends. The default is false.

Figure 2 Binding a URL to an Iframe

```
<iframe src="//www.facebook.com/plugins/likebox.php
?href=http://www.facebook.com/etennisnetpage
&width=292&height=490
&colorscheme=light
&show_faces=false
&stream=true
&header=true
&appId=xxxxxxxxxxxxxx"
scrolling="no"
frameborder="0"
style="border:none; overflow:hidden; width:292px; height:590px;"
allowTransparency="true">
</iframe>
```



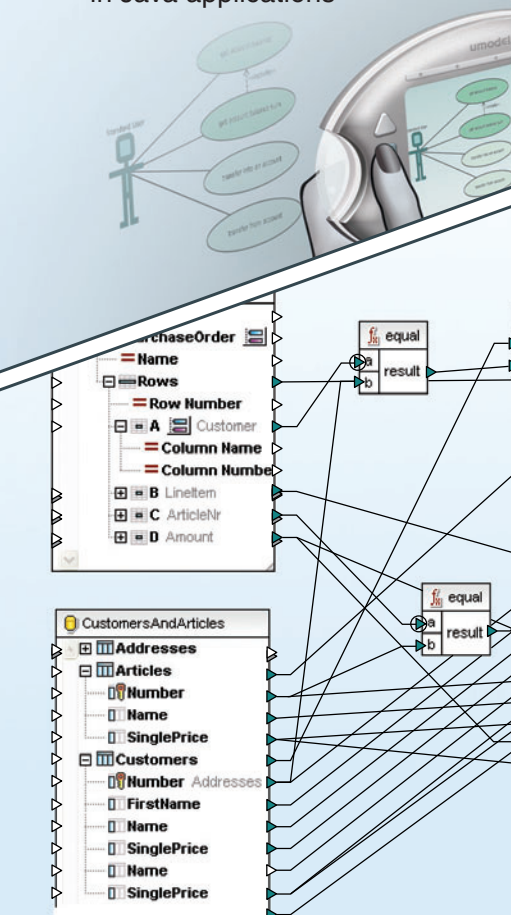
## Visualize software works of art with the complete set of tools from Altova®



**Altova MissionKit® is an integrated suite of UML, XML, and data integration tools for today's software architect.**

### **NEW** in Version 2013:

- Support for UML 2.4 & SysML 1.2
- Displaying .NET associations as UML properties
- Smart Fix XML validation with automatic error correction
- Support for SQL stored procedures in data mapping projects
- Seamless integration options in Java applications



### **Altova MissionKit includes multiple tools for software architects:**

#### **UModel®** – UML tool for software modeling

- Support for all UML diagram types, MDA, SQL database diagrams, BPMN, SysML
- Reverse engineering and code generation in Java, C#, VB.NET

#### **XMLSpy®** – XML editor & development environment

- Support for all XML-based technologies
- Royalty-free Java, C#, C++ code generation

#### **MapForce®** – any-to-any data mapping & integration tool

- Mapping between XML, databases, EDI, flat files, XBRL, Excel, Web services
- Royalty-free Java, C#, C++ code generation

### **Plus up to four additional tools...**



**Download a 30 day free trial!**

Try before you buy with a free, fully functional trial from [www.altova.com](http://www.altova.com)



Scan to learn more about these Altova MissionKit tools.



Figure 3 Sample Like Box

is the first step to accomplish. It's important to note that the Facebook Like Box social plug-in is only intended to be used with Facebook fan pages and won't work if you connect it to a personal Facebook account.

Also, note that Facebook differentiates between fan pages and profile pages. The bottom line is that fan pages are for businesses, whereas profile pages are for individuals. There are some differences between the two as far as the allowed actions are concerned. First and foremost, a team of people can have admin rights on a fan page. In addition, posts from a fan page can be specifically targeted by language and location so they reach followers (well, fans, actually) who can best receive them. Fan pages support additional features and can be promoted via ads and sponsored articles.

Conversely, profile pages are intended to let owners stay in touch with friends and family. Being a friend becomes a mandatory condition to get updates, even though through the subscription mechanism you can allow non-friends to get your updates as well.

Configuring the Like Box for a Web page couldn't be easier. You can preview the content being displayed and grab related HTML directly from the Facebook developers site. Go to [bit.ly/hFv07y](http://bit.ly/hFv07y) for a live demo. In the end, it's all about arranging a long URL to set on an iframe element.

**Figure 1** lists the parameters you can use in the URL.

All you do is arrange a URL and bind it to an iframe, as shown in **Figure 2**.

It goes without saying that you can also embed the Like Box in a desktop application (for example, a Windows Presentation Foundation—or WPF—application) through a WebBrowser control.

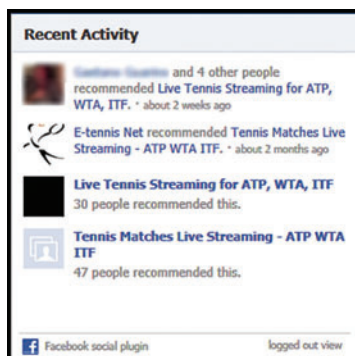


Figure 4 The Activity Plug-in in Action

As you can see, the plug-in allows for some quick styling that works most of the time. However, if you want to apply your own CSS (to the extent that it's possible and documented) you should embed the Like Box via JavaScript and HTML5. **Figure 3** shows the output of a Like Box for a sample site with custom and regular (light) style.

## The Activity Plug-In

Another quick way to incorporate existing specific Facebook content in Web pages or desktop applications (via Web-browser components) is the Activity plug-in.

This plug-in aggregates stories resulting from the interaction that users have with your site through Facebook. Notice that the target here isn't a Facebook page but rather an external site. Example actions that generate such feeds are liking content on the site, watching videos, commenting and sharing content from the site. The plug-in is also able to detect whether the current user of the site that contains the Activity plug-in is logged in to Facebook or not. If so, the displayed feed is restricted to friends of the user. Otherwise, the plug-in shows recommendations from across the site, while giving the user the option to log in to Facebook and receive more-targeted feedback. Here's the markup you need (note that the Activity plug-in can only be consumed through HTML5 markup):

```
<div class="fb-activity"
  data-site="www.yoursite.com"
  data-app-id="xxxxxxxxxxxxx"
  data-width="300"
  data-height="300"
  data-header="true"
  data-recommendations="true">
</div>
```

You must incorporate the JavaScript SDK in the page in order for this markup to produce what's in **Figure 4**.

## Accessing the Timeline

In past columns, I used the Facebook C# SDK to post to the wall both plain text and attachments. Once the post is made, friends and fans can interact with it by liking it, sharing it and commenting on it. Let's see what it takes to read the timeline of a given user.

Facebook assigns a unique and fairly long ID to any account, whether profile or fan page. Users, however, don't use this ID to identify pages. So the first thing to do in order to read a timeline is match the public name of the page (or user) to the underlying Facebook ID. As a mere exercise, you can type the following into the address bar of any browser: <https://graph.facebook.com/your-page-name>.

The placeholder "your-page-name" is just the name of the account as you would type it to reach the page. Getting the ID of the account for which you intend to read the timeline is only the first step. You also need to be authenticated to access the feed. The bottom line is that any operation that goes directly against the underlying Facebook Graph API requires OAuth authentication. This means that the same preliminary steps discussed in past columns must also be done here:

- Create a Facebook app to deal with the Facebook back end.
- Have the user authorize the Facebook app to operate on behalf of the account. This step delivers an access token that binds together (for a limited time) the user and Facebook app.



# MANAGE



# FILES

## CONVERT PRINT CREATE MODIFY & COMBINE

### Aspose.Words

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

### Aspose.Cells

XLS, XLSX, XLSM, XLTX, CSV,  
SpreadsheetML & image formats.

### Aspose.BarCode

JPG, PNG, BMP, GIF, TIF, WMF,  
ICON & other image formats.

### Aspose.Pdf

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

### Aspose.Email

MSG, EML, PST, EMLX &  
other formats.

### Aspose.Slides

PPT, PPTX, POT, POTX, XPS,  
HTML, PNG, PDF & other formats.

Follow us on  
Facebook & Twitter



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



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

US Sales: 1.888.277.6734  
[sales@aspose.com](mailto:sales@aspose.com)

EU Sales: +44 (0) 141 416 1112  
[sales.europe@aspose.com](mailto:sales.europe@aspose.com)

AU Sales: +61 2 8003 5926  
[sales.asiapacific@aspose.com](mailto:sales.asiapacific@aspose.com)





Figure 5 The JSON Structure of a Timeline Item in Facebook

- Use the access token as the key to operate against the Graph API from within the host application (for example, an ASP.NET MVC application).

Figure 6 Parsing a Dynamic Object into a Classic C# Class

```
public class FacebookPost
{
    public String PostId { get; set; }
    public String Author { get; set; }
    public String Picture { get; set; }
    public String Link { get; set; }
    public String Published { get; set; }
    public String ContentHtml { get; set; }

    private delegate String ExtractDelegate();
    public static IList<FacebookPost> Import(dynamic data)
    {
        var posts = new List<FacebookPost>();
        foreach (var item in data)
        {
            var tempItem = item;
            var fb = new FacebookPost
            {
                PostId = Extract(() => tempItem["id"]),
                Published = Extract(() => tempItem["created_time"]),
                Picture = Extract(() => tempItem["picture"]),
                Link = Extract(() => tempItem["link"]),
                Author = Extract(() => tempItem["from"]["name"])
            };
            try
            {
                fb.ContentHtml = Extract(() => tempItem["message"]);
            }
            catch
            {
            }

            if (!String.IsNullOrEmpty(fb.ContentHtml))
                posts.Add(fb);
        }
        return posts;
    }

    private static String Extract(ExtractDelegate func)
    {
        try {
            return func();
        } catch {
            return null;
        }
    }
}
```

Once acquired, the access token can be saved to a cookie and used for every further operation until it expires. Here's the required code to get the raw feed from the Facebook server:

```
var name = "name-of-the-user";
// For example, joedummy
var client =
    new FacebookClient(access_token);
dynamic user = client.Get(name);
dynamic feed =
    client.Get(name + "/feed");
```

The first call to the Facebook client doesn't strictly require the access token, as it's expected to return only public information about the user. The user variable exposes properties such as `first_name`, `last_name`, `id` and `location`. Depending on your intentions, you might not need to place this call.

The second call to the Facebook client is what really does the trick. It takes a string that denotes the path to the user's feed. You build the path concatenating the account's name with the `/feed` string. In return, you get a dynamic C# object built out of an underlying JSON stream. **Figure 5** shows the structure of the JSON stream as captured by Fiddler.

It shows that the selected post got two types of actions—it has been liked and commented upon. It also currently counts 14 likes. More details about people who commented on it or liked it are available as you expand nodes. Finally, you find the content of the post. At the JSON level, the content of the post is the `message` field.

It's important to note that not all posts have the same structure; this is the reason why the Facebook C# SDK doesn't use plain, statically defined classes as data transfer objects (DTOs). A common snag is that the post lacks a `message`, `link` and `picture`, but includes a `story` field. This is the case, for example, when the admin adds a collection of photos.

The Facebook C# SDK just hands you a dynamic C# object. Parsing that into more defined data types—or deciding that the object is good as is to trickle down to the view—is your call. **Figure 6** shows some code that attempts to parse the dynamic object into a classic C# class.

The most annoying part of this code is finding an effective way to check whether a given property is defined on the dynamic object you parse. Quite a few [stackoverflow.com](http://stackoverflow.com) users agree on the approach shown in **Figure 6**, based on a delegate.

Dealing with social networks opens up a whole new world of possibilities, which become opportunities for developers to think of and realize new creative applications. As a .NET developer, you should be a friend of the Facebook C# SDK. ■

**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](http://twitter.com/despos).

**THANKS** to the following technical expert for reviewing this article:  
Scott Densmore (Microsoft)

# Telerik DevCraft

The all-in-one toolset for professional developers targeting Microsoft platforms.



- Create web, mobile and desktop applications that impress
- Cover any .NET platform and any device
- Code faster and smarter without cutting corners

Get your 30-day free trial today:  
[www.telerik.com/all-in-one](http://www.telerik.com/all-in-one)

 **telerik**





# Why Does Entity Framework Reinsert Existing Objects into My Database?

Just as it was time to come up with an idea for this column, three people asked me, via twitter and e-mail, why Entity Framework reinserts existing objects into their databases. Deciding what to write about became easy.

Because of its state management capabilities, when Entity Framework works with graphs, its entity state behavior doesn't always align with your ideas of how it should work. Let's look at a typical example.

Suppose I have two classes, `ScreenCast` and `Topic`, where each `ScreenCast` is assigned a single `Topic`, as shown in **Figure 1**.

If I were to retrieve a list of `Topics`, assign one of them to a new `ScreenCast` and then save—with the entire set of operations contained in a single context—there would be no problem, as the following example shows:

```
using (var context = new ScreenCastContext())
{
    var dataTopic = context.Topics.FirstOrDefault(t => t.Name.Contains("Data"));
    context.ScreenCasts.Add(new ScreenCast
    {
        Title = "EF101",
        Description = "Entity Framework 101",
        Topic = dataTopic
    });

    context.SaveChanges();
}
```

A single `ScreenCast` would be inserted into the database with the proper foreign key for the chosen `Topic`.

When you're working in client applications, or performing these steps within a single Unit of Work where the context is tracking all of the activity, this is the behavior you'd probably expect. However, if you're working with disconnected data the behavior is quite different, and this has surprised many developers.

Figure 1 The `ScreenCast` and `Topic` Classes

```
public class ScreenCast
{
    public int Id { get; set; }
    public string Title { get; set; }
    public string Description { get; set; }
    public Topic Topic { get; set; }
    public int TopicId { get; set; }
}

public class Topic
{
    public int Id { get; set; }
    public string Name { get; set; }
}
```

Code download available at [archive.msdn.microsoft.com/mag201304DataPoints](http://archive.msdn.microsoft.com/mag201304DataPoints).

## Added Graphs in Disconnected Scenarios

A common pattern that I use for handling reference lists is to use a separate context, which would no longer be in scope by the time you saved any user modifications. This situation is common for Web applications and services, but it can also occur in a client-side app. Here's an example that uses a repository for reference data with the following `GetTopicList` method for retrieving a list of `Topics`:

```
public class SimpleRepository
{
    public List<Topic> GetTopicList()
    {
        using (var context = new ScreenCastContext())
        {
            return context.Topics.ToList();
        }
    }
    ...
}
```

You might then present the `Topics` in a list on a Windows Presentation Foundation (WPF) form that lets users create a new `ScreenCast`, such as the one shown in **Figure 2**.

In a client app, such as the WPF form in **Figure 2**, you might then set the selected item from the dropdown to the new `ScreenCast`'s `Topic` property with code such as this:

```
private void Save_Click(object sender, RoutedEventArgs e)
{
    repo.SaveNewScreenCast(new ScreenCast
    {
        Title = titleTextBox.Text,
        Description = descriptionTextBox.Text,
        Topic = topicListBox.SelectedItem as Topic
    });
}
```

Now the `ScreenCast` variable is a graph containing the new `ScreenCast` and the `Topic` instance. Passing that variable into the repository's `SaveNewScreenCast` method adds the graph to a new context instance and then saves it to the database, like so:

```
public void SaveNewScreenCast(ScreenCast screenCast)
{
    using (var context = new ScreenCastContext())
    {
        context.ScreenCasts.Add(screenCast);
        context.SaveChanges();
    }
}
```

Profiling the database activity reveals that not only is the `ScreenCast` inserted but, before that, a new row is inserted for the `Data Dev` topic into the `Topics` table, even though that topic already existed:

```
exec sp_executesql N'insert [dbo].[Topics]([Name])
values (@0)
select [Id]
from [dbo].[Topics]
where @@ROWCOUNT > 0 and [Id] = scope_identity()', N'@0 nvarchar(max)
', @0=N'Data Dev'
```



# RACKSPACE

*just*

# OPEN-SOURCED THE CLOUD

**The open age started with Linux. Next came Android.** Then, Rackspace and NASA created OpenStack and open-sourced the biggest platform of them all. It's called the open cloud. Now, you're no longer locked in to the pricing, service limitations, or pace of innovation of any one vendor. You're free to run your cloud anywhere you want: in your data center, in ours, or with any other OpenStack provider—and the response has been overwhelming. More than 800 organizations and 6,000 individuals are collaborating on OpenStack. This is greater than one company. It's a movement.

With over 200,000 customers and more than 60% of the FORTUNE® 100 trusting our **Fanatical Support®**, we've done big things at Rackspace before—but this is the biggest.

**Try it today.** Download the open cloud at **[rackspace.com/open](http://rackspace.com/open)**

Rackspace and Fanatical Support are service marks of Rackspace US, Inc. All trademarks, service marks, and images are the property of their respective owners.



Figure 2 A Windows Presentation Foundation Form for Entering New Screenshots

This behavior has confounded many developers. The reason it happens is that when you use the DbSet.Add method (that is, Screenscasts.Add), not only is the state of the root entity marked “Added,” but everything in the graph that the context was not previously aware of is marked Added as well. Even though the developer may be aware that the Topic has an existing Id value, Entity Framework honors its EntityState (Added) and creates an Insert database command for the Topic, regardless of the existing Id.

It's possible to control the state,  
but this requires a deeper  
understanding of EF behavior.

While many developers may anticipate this behavior, there are many who don't. And in that case, if you aren't profiling the database activity, you may not realize it's occurring until the next time you (or a user) discover duplicate items in the Topics list.

*Note: If you're not familiar with how EF inserts new rows, you may be curious about the select in the middle of the preceding SQL. That's to ensure EF will get back the Id value of the newly created Screenshot so it can set the value in the Screenshot instance.*

## Not Just a Problem When Adding Entire Graphs

Let's look at another scenario where this problem might occur.

What if, instead of passing a graph to the repository, the repository method requests both the new Screenshot and the selected Topic as parameters? Instead of adding a full graph, it adds the Screenshot entity and then sets its Topic navigation property:

```
public void SaveNewScreenshotWithTopic(Screenshot screenshot, Topic topic)
{
    using (var context = new ScreenshotContext())
    {
        context.Screenshots.Add(screenshot);
        screenshot.Topic = topic;
        context.SaveChanges();
    }
}
```

In this case, the SaveChanges behavior is the same as with the Added graph. You might be familiar with using the EF Attach method

to attach an untracked entity to a context. In that case, the entity's state starts out as Unchanged. But here, where we're assigning the Topic to the Screenshot instance, not to the context, EF considers this to be an unrecognized entity and its default behavior for unrecognized entities with no state is to mark them as Added. So, again, the Topic will be inserted into the database when SaveChanges is called.

It's possible to control the state, but this requires a deeper understanding of EF behavior. For example, if you were to attach the Topic directly to the context, instead of to the Added Screenshot, its EntityState would start out as Unchanged. Setting it to screenshot.Topic wouldn't alter the state because the context is already aware of the Topic. Here's the modified code that demonstrates this logic:

```
using (var context = new ScreenshotContext())
{
    context.Screenshots.Add(screenshot);
    context.Topics.Attach(topic);
    screenshot.Topic = topic;
    context.SaveChanges();
}
```

Alternatively, in lieu of context.Topics.Attach(topic), you could set the state of the Topic before or after the fact, explicitly setting its state to Unchanged:

```
context.Entry(topic).State = EntityState.Unchanged
```

Calling this code before the context is aware of the Topic will cause the context to attach the Topic and then set the state.

Though these are correct patterns for handling this problem, they're not obvious. Unless you've learned about this behavior and the required code patterns in advance, you're more apt to write code that seems logical, then run into this problem and only at that point start trying to figure out what the heck is going on.

## Save the Grief and Use That Foreign Key

But there's a much simpler way to avoid this state of confusion (pardon my pun), which is to take advantage of the foreign key properties.

Rather than setting the navigation property and having to worry about the state of the Topic, just set the TopicId property, because you do have access to that value in the Topic instance. This is something I find myself frequently suggesting to developers. Even on Twitter, I see the question: “Why is EF inserting data that already exists?” And I often guess correctly in reply: “Any chance u r setting a navigation property on a new entity instead of an FK? ☺”

So let's revisit the Save\_Click method in the WPF form and set the TopicId property instead of the Topic navigation property:

```
repo.SaveNewScreenshot(new Screenshot
{
    Title = titleTextBox.Text,
    Description = descriptionTextBox.Text,
    TopicId = (int)topicListBox.SelectedValue
});
```

The Screenshot that's sent to the repository method is now just the single entity, not a graph. Entity Framework can use the foreign key property to directly set the table's TopicId. Then it's simple (and faster) for EF to create an insert method for the Screenshot entity including the TopicId (in my case, the value 2):

```
exec sp_executesql N'insert [dbo].[Screenshots]([Title], [Description], [TopicId])
values (@0, @1, @2)
select [Id]
from [dbo].[Screenscasts]
where @@ROWCOUNT > 0 and [Id] = scope_identity()',
N'@0 nvarchar(max), @1 nvarchar(max), @2 int',
@0=N'EFFK101', @1=N'Using Foreign Keys When Setting Navigations', @2=2
```

# Building Blocks for Global Data Quality Success



## A strong foundation for enterprise data starts with Melissa Data.

Our powerful, scalable data cleansing and integration tools help you profile, cleanse, consolidate, and enrich your contact data. Gain a better understanding of your customer, vendor and supplier data; improve deliverability; increase cost savings; and enhance your operational efficiencies with Melissa Data.

### Advanced Functionalities:

- Verify, correct, and enhance addresses for 240+ countries
- Add lat/long coordinates to addresses all over the world
- Match and consolidate data to create the golden record
- Append missing contact data like phone numbers and email addresses
- Integrate into many technologies with multiplatform capabilities

[www.MelissaData.com](http://www.MelissaData.com)  
or call 1-800-MELISSA (635-4772)

**MELISSA DATA®**  
Your Partner in Data Quality

**Figure 3 Repository Method Designed to Protect Against Accidental Navigation Property Insertion into Database**

```
public void SaveNewScreenecastWithTopicId(Screenecast screencast, int topicId)
{
    if (topicId > 0)
    {
        screencast.Topic = null;
        screencast.TopicId = topicId;
    }
    using (var context = new ScreenecastContext())
    {
        context.Screenecasts.Add(screencast);
        context.SaveChanges();
    }
}
```

If you wanted to keep the construction logic in the repository and not force the UI developer to worry about setting the foreign key, you could specify a Screenecast and the Topic's Id as parameters for the repository method and set the value in the method as follows:

```
public void SaveNewScreenecastWithTopicId(Screenecast screencast, int topicId)
{
    using (var context = new ScreenecastContext())
    {
        screencast.TopicId = topicId;
        context.Screenecasts.Add(screencast);
        context.SaveChanges();
    }
}
```

In our never-ending worries about what *could* happen, we need to consider the possibility that a developer might set the Topic navigation property anyway. In other words, even though we want to use the foreign key to avoid the EntityState problem, what if the Topic instance is part of the graph, such as in this alternative code for the Save\_Click button:

```
repo.SaveNewScreenecastWithTopicId(new Screenecast
{
    Title = titleTextBox.Text,
    Description = descriptionTextBox.Text,
    Topic=topicListBox.SelectedItem as Topic
},
(int) topicListBox.SelectedValue);
```

Unfortunately, this brings you back to the original problem: EF sees the Topic entity in the graph and adds it to the context along with Screenecast—even though the Screenecast.TopicId property has been set. And again, the EntityState of the Topic instance creates confusion: EF will insert a new Topic and use the value for that new row's Id when it inserts the Screenecast.

The safest way to avoid this is to set the Topic property to null when you set the foreign key value. If the repository method will be

used by other UIs where you can't be sure that only existing Topics will be used, you might even want to provide for the possibility of a newly created Topic being passed in. **Figure 3** shows the repository method modified yet again to perform this task.

Now I have a repository method that covers a number of scenarios, even providing logic to accommodate new Topics being passed in to the method.

## ASP.NET MVC 4 Scaffolding-Generated Code Avoids the Problem

Although this is a problem that's inherent in disconnected apps, it's worth pointing out that if you're using ASP.NET MVC 4 scaffolding to generate views and MVC controllers, you'll avoid the problem of duplicate navigation entities being inserted into the database.

A much simpler way to avoid this state of confusion is to take advantage of the foreign key properties.

Given the one-to-many relationship between Screenecast and Topic, as well as the TopicId property that's the foreign key in the Screenecast type, the scaffolding generates the following Create method in the controller:

```
public ActionResult Create()
{
    ViewBag.TopicId = new SelectList(db.Topics, "Id", "Name");
    return View();
}
```

It has built a list of Topics to pass to the view and named that list TopicId—the same name as the foreign key property.

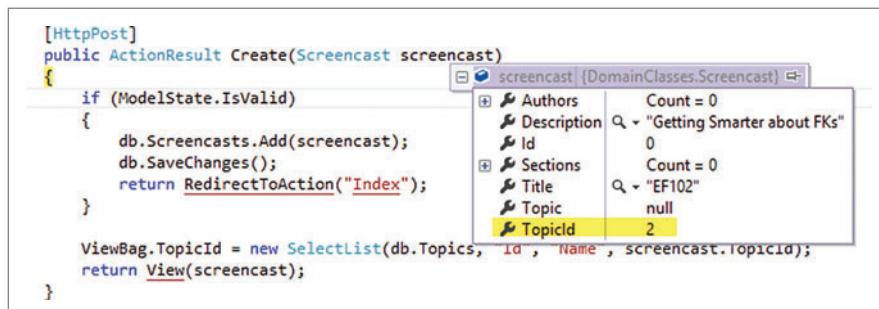
The scaffolding has also included the following List in the markup for the Create view:

```
<div class="editor-field">
    @Html.DropDownList("TopicId", String.Empty)
    @Html.ValidationMessageFor(model => model.TopicId)
</div>
```

When the view posts back, the HttpRequest.Form includes a query-string value named TopicId that comes from the ViewBag property. TopicId's value is that of the selected item from the

DropDownList. Because the query string name matches the Screenecast's property name, ASP.NET MVC model binding uses the value for the TopicId property of the Screenecast instance it creates for the method parameter, as you can see in **Figure 4**.

To verify this, you could change the controller's TopicId variables to something else, such as TopicIdX, and make the same change to the "TopicId" string in the view's @Html.DropDownList, and the query-string value (now TopicIdX) would be ignored and screencast.TopicId would be 0.



**Figure 4 The New Screenecast Gets Its TopicId from the Matching HttpRequest Query-String Value**



There's no Topic instance being passed back through the pipeline. So ASP.NET MVC depends on the foreign key property by default and avoids the particular problem of reinserting an existing duplicate Topic into the database.

## It's Not You! Disconnected Graphs Are Complicated

While the EF team has done a lot to make working with disconnected data easier from one version of EF to the next, it's still a problem that daunts many developers who aren't well-versed in the expected behavior of EF. In our book, "Programming Entity Framework: DbContext" (O'Reilly Media, 2012), Rowan Miller and I devoted an entire chapter to working with disconnected entities and graphs. And when creating a recent Pluralsight course, I added in an unplanned 25 minutes that focused on the complexity of disconnected graphs in repositories.

While the EF team has done a lot to make working with disconnected data easier from one version of EF to the next, it's still a problem that daunts many developers.

It's very convenient to work with graphs when you're querying and interacting with data, but when it comes to building relationships with existing data, foreign keys are your friends! Take a look at my January 2012 column, "Making Do with Absent Foreign Keys" ([msdn.microsoft.com/magazine/hh708747](http://msdn.microsoft.com/magazine/hh708747)), which is also about some of the pitfalls of coding without foreign keys.

In an upcoming column, I'll continue on my quest to alleviate some of the pain developers encounter when working with graphs in disconnected scenarios. That column, which will be Part 2 on this topic, will focus on controlling the EntityState in many-to-many relationships and navigation collections. ■

**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](http://thedatafarm.com/blog) and is the author of "Programming Entity Framework" (2010) as well as a Code First edition (2011) and a DbContext edition (2012), all from O'Reilly Media. Follow her on Twitter at [twitter.com/julielerman](https://twitter.com/julielerman).

**THANKS** to the following technical expert for reviewing this article:  
Diego Vega (Microsoft)



## LESS PLUMBING CODE, MORE FEATURES

CodeFluent Entities is a Visual Studio 2008/2010/2012 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).



VISUAL STUDIO  
2012 AND  
WINDOWS 8  
READY

- ✓ UML FREE
- ✓ TEMPLATE FREE
- ✓ FRAMEWORK FREE
- ✓ ORM FREE

Using this model-first approach, your business logic is decoupled from the technology and your foundations will automatically benefit from upcoming innovation.

Your application deserves rock-solid foundations, let CodeFluent Entities generate them, and keep the fun part for you! Focus on what makes the difference.



**DOWNLOAD YOUR FREE LICENSE**  
[www.softfluent.com/landings\\_cfe\\_msdn](http://www.softfluent.com/landings_cfe_msdn)





### 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](http://www.softfluent.com) - Contact: [info@softfluent.com](mailto:info@softfluent.com)  
CodeFluent Entities is a trademark of SoftFluent SAS. Other names may be trademark of their respective owners.





# NoSQL Data in the Cloud with Windows Azure Tables

The price of storing data on disk has dropped so dramatically it seems like science fiction, opening the floodgates for companies to store massive amounts of data. But being able to store lots of data economically solves only half the problem. The data has become so large and complex that traditional database management tools and data processing applications are vastly inadequate. With so much data on disk, new issues have arisen, such as ingesting the data, performing searches, sharing the data, analyzing it and, ultimately, visualizing it.

The power of cloud computing has stepped up to fill this need. The ability to run massively parallel software solutions—running on tens, hundreds or even thousands of servers—is the silver bullet that enables organizations to deal with all of that stored data.

Microsoft realized this important trend several years ago. Windows Azure Storage (WAS) was launched in November 2008 and dramatically improved the ability of businesses to get value from the massive amounts of data being stored.

In the words of Brad Calder, a distinguished engineer at Microsoft and the shepherd who guided the construction of the WAS system, “Windows Azure Storage is a cloud storage system that provides customers the ability to store seemingly limitless amounts of data for any duration of time that is highly available and durable. When using Windows Azure Storage, you have access to your data from anywhere, at any time, and only pay for what you use and store.”

WAS is used inside Microsoft for applications such as social networking search; serving video, music and game content; and managing medical records. It's also used by the Bing search engine to provide almost-immediate publicly searchable content from Facebook or Twitter posts or status updates. With around 350TB of data, the scope of Facebook and Twitter data is remarkable. When this data is being ingested, transaction throughput reaches peaks of around 40,000 transactions per second and totals between 2 to 3 billion transactions per day.

This month we'll explore one facet of WAS—Windows Azure Tables—both how it works and how developers can get it up and running quickly.

## The Landscape

The modern data scientist is faced with many choices when selecting a data platform, each with its own strengths and weaknesses. For

Figure 1 Scalability Targets for a Single Storage Account

Individual Storage Accounts	
Capacity	Up to 200TB
Transactions	Up to 20,000 entities/messages/blobs per second
Bandwidth for a Geo-Redundant Storage Account	
Ingress	Up to 5Gbps
Egress	Up to 10Gbps
Bandwidth for a Locally Redundant Storage Account	
Ingress	Up to 10Gbps
Egress	Up to 15Gbps

example, many big data solutions are based on the concept of NoSQL, which means that a relational database management system (RDBMS) model isn't used—there are no tables and no SQL statements. Instead, the data structures are typically a massive collection of key/value pairs or associative arrays. The popular choices today

Figure 2 The Entity EmailAddressEntity

```
// Our entity derives from TableEntity
public class EmailAddressEntity : TableEntity
{
    // Basic information that makes up our entity
    public string EmailAddress { get; set; }
    public string PhoneNumber { get; set; }

    // A necessary default constructor
    public EmailAddressEntity()
    {
    }

    // A two-parameter constructor
    public EmailAddressEntity(string email, string phone)
    {
        EmailAddress = email;
        PhoneNumber = phone;
        SetKeys(email, phone);
    }

    // A method that initializes the partition key and row key
    public void SetKeys(string email, string phone)
    {
        int startIndex = email.IndexOf("@");
        // Extract the mailname from the e-mail address
        string mailname = email.Substring(0, startIndex);
        // Extract the domain from the e-mail address
        string domain = email.Substring(startIndex + 1);
        // Perform the mandatory assignments to the partition key and row key
        PartitionKey = domain;
        RowKey = mailname;
        PhoneNumber = phone;
    }
}
```

Code download available at [archive.msdn.microsoft.com/mag201304AzureInsider](http://archive.msdn.microsoft.com/mag201304AzureInsider).

# *When seconds count,* you need **ScaleOut Analytics Server™**

**ScaleOut Analytics Server's powerful, memory-based map/reduce engine delivers near real-time results.**

Breakthrough ease of use  
cuts development time  
and eliminates tuning.

## ***Analyze data faster than ever:***

- In-memory data grid with map/reduce maximizes throughput.
- Powerful distributed analytics engine eliminates tuning.
- Java/C# object-oriented data model simplifies development.
- Built-in code shipping automates deployment.
- Dynamic execution environment reduces startup time.

Get a **free** trial copy of  
**ScaleOut Analytics Server**  
today!



## **SCALEOUT SOFTWARE**

In-Memory Data Grids for the Enterprise

[www.scaleoutsoftware.com](http://www.scaleoutsoftware.com) | 503.643.3422



Figure 3 Inserting an EmailAddressEntity

```
try
{
    // Use the local storage emulator
    var storageAccount = CloudStorageAccount.DevelopmentStorageAccount;
    // Create a cloud table client object
    CloudTableClient tableClient = storageAccount.CreateCloudTableClient();
    // Create an e-mail address table object
    CloudTable emailAddressTable =
        tableClient.GetTableReference("EmailAddressTable");
    // Create the table if it does not exist
    // Only insert a new record once for this demo
    if (emailAddressTable.CreateIfNotExists() == true)
    {
        // Create a new EmailAddressEntity entity
        EmailAddressEntity emailAddress = new
            EmailAddressEntity("bterkaly@microsoft.com", "555-555-5555");
        // Create an operation to add the new e-mail and phone number to
        // the emailAddressTable
        TableOperation insertEmail = TableOperation.Insert(emailAddress);
        // Submit the operation to the table service
        emailAddressTable.Execute(insertEmail);
    }
}
catch (Exception ex)
{
    // Put the message in the Web page title (for testing purposes)
    // Real error messages should go to a proper log file
    this.Title = ex.Message.ToString();
    throw;
}
```

are MongoDB, Cassandra, HBase, CouchDB, Neo4j and Windows Azure Tables. This article will focus on Windows Azure Tables.

Despite the major differences, both SQL and NoSQL databases have one thing in common: these technologies are offered as a service in the cloud, freeing developers from having to manually provision and de-provision data servers. For example, Windows Azure Tables is offered as a service and a developer never has to think in terms of separate physical servers.

In this month's column, we'll start with a brief discussion of some of the features and capabilities of Windows Azure Tables. Next, we'll provide some code to demonstrate how you might work with Windows Azure Tables in terms of inserting and querying data. And, finally, we'll take a look at some of the design goals and the high-level implementation details of WAS.

## Some Basics

One of the great features of Windows Azure Tables is that storage is offered across three geographically distributed regions, includ-

ing the United States, Europe and Asia. Every Microsoft data-center complies with the Inter-

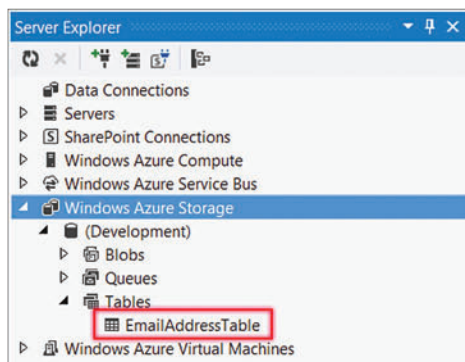


Figure 4 Server Explorer

nization for Standardization (ISO) 27001, SSAE 16 ISAE 3402, EU Model Clauses and Health Insurance Portability and Accountability Act (HIPAA) business associate agreement (BAA) standards. Another important feature is geo-redundant storage, which allows you to replicate your data in another datacenter within the same region, adding yet another level of disaster recovery.

WAS performance and capacities are correlated to storage accounts. An individual storage account includes 200TB of storage. Windows Azure Tables have been optimized to provide incredibly fast query performance under write-heavy workloads. You can read more at [bit.ly/cMAWsZ](http://bit.ly/cMAWsZ).

Windows Azure Tables can be accessed from almost anywhere.

Figure 1 shows the scalability targets for a single storage account created after June 7, 2012.

WAS analytics are also available, allowing developers to trace storage requests, analyze usage trends and optimize data-access patterns in a storage account. Read more at [bit.ly/XGLtGt](http://bit.ly/XGLtGt).

Be aware that the WAS system includes other abstractions, such as blobs and queues. We'll focus here on Windows Azure Tables, which are used to store non-relational structured and semi-structured data. The most succinct way to express the value of Windows Azure Tables is that they support NoSQL key-value lookups at scale and under write-heavy workloads. From a developer's point of view, Windows Azure Tables are for storing large collections of non-uniform objects or for serving pages on a high-traffic Web site.

Windows Azure Tables can be accessed from almost anywhere. The entire storage system is Representational State Transfer (REST)-enabled, which means that any client capable of HTTP can communicate with the WAS system. Obvious clients include iOS, Android, Windows 8 and different Linux distros. The REST API supports inserts, upserts, updates and deletes, and selects or queries.

When working with Windows Azure Tables, a key starting point is understanding how to control the data partitioning scheme. For any given Windows Azure Table, the data architect must define (up front) a PartitionKey and a RowKey. This is perhaps the most important decision you'll make when using Windows Azure Tables. PartitionKeys and RowKeys determine how your data is automatically partitioned by the storage service and the way your queries will perform. It's recommended that you understand how your data will be queried before finalizing your decisions on PartitionKey and RowKey. Later, we'll delve into the mechanics of transactional consistency and their relationship to PartitionKeys. For now, let's walk through a simple example of how the WAS system partitions table data.

EmailAddressTable [Table]				
Enter a WCF Data Services filter to limit the entities returned				
PartitionKey	RowKey	Timestamp	EEmailAddress	PhoneNumber
microsoft.com	bterkaly	1/13/2013 4:35:...	bterkaly@micro...	555-555-5555



# 1&1 WEB HOSTING

## YOUR HOSTING PARTNER

Whether you use Windows or Linux, 1&1 is always the right choice. We offer the latest in programming languages with PHP 5.4 for Linux and ASP.NET 4.5 for Windows.

We also now offer flexible prepaid billing plans of 1, 12, 24, or 36 month terms to allow you to match your hosting needs with the billing option best suited for you.



### Dual Hosting for Maximum Reliability

Your website hosted across multiple servers in two different data centers, and in two geographic locations.



### Unlimited Bandwidth (Traffic)



### IPv6 Ready



### NEW: Choose your prepaid billing plan



## 1&1 Starter Windows



## 1&1 Starter Linux

50 GB Webspace

Unlimited Bandwidth (Traffic)

250 E-Mail Accounts (2 GB each)

24/7 Phone and E-mail Support

NEW! ASP.NET/  
.NET Framework 4, 4.5

Microsoft  
**ASP.NET**

NEW! PHP 5.4  
and Host multiple websites

NEW! 1 MSSQL 2012  
Database (1 GB)

Microsoft  
**SQL Server 2012**

10 MySQL  
Databases (1 GB each)

NEW! ASP.NET MVC

NEW! Webspaces Recovery and  
daily server backups

NEW! Dedicated App Pools

2 Click & Build Applications,  
like Wordpress, Joomla!, TYPO3

Starting at **\$3.49** per month  
Save up to \$90



1and1.com



\* Offers valid for a limited time only. Starter hosting package price of \$3.49 per month is valid for a 36 month prepaid package; total purchase amount of \$125.64. Visit [www.1and1.com](http://www.1and1.com) for billing information and 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. © 2013 1&1 Internet. All rights reserved.

## A Quick Tutorial

Imagine you wish to store and retrieve e-mails from various domains, such as the following: bterkaly@microsoft.com, ricardo.villalobos@microsoft.com, brunoterkaly@hotmail.com and ricardovillalobos@hotmail.com. In these e-mail addresses, the domain names are microsoft.com and hotmail.com, while the e-mail names are bterkaly and ricardo.villalobos. Typical queries search first by domain name, then by e-mail name.

In this simple example, the choice of PartitionKey and RowKey are fairly straightforward. We'll map the domain name to the PartitionKey and the e-mail name to the RowKey.

The code in **Figure 2** should make things a bit clearer. It illustrates four simple capabilities:

- Defining an entity (EmailAddressEntity)
- Defining the table that will store the entities (EmailAddressTable)
- Inserting the entity into the table (insert EmailAddressEntity into EmailAddressTable)
- Querying the table to search for a specific entity (search for bterkaly@microsoft.com)

First, we define the entity structure itself, EmailAddressEntity, as shown in **Figure 2**. The actual table (a container for entities) will be defined later, when we insert EmailAddressEntity into the table. An entity can be thought of as an individual object; it's the smallest unit of data that can be stored in a Windows Azure Table. As mentioned previously, an entity is a collection of typed name-value pairs, often referred to as properties. Tables are collections of entities, and each entity belongs to a table just as a row does in a relational database table. But tables in Windows Azure Table Storage don't have a

fixed schema. There's no requirement that all entities in a table be structurally identical, as is the case for a relational database table.

There are four main pieces of information in **Figure 2**. The first two, EMailAddress and PhoneNumber, are simply two strings we want to store. The other two are the properties PartitionKey and RowKey, which we discussed previously. A third property required of all entities is Timestamp, which is used internally by the system to facilitate optimistic concurrency.

The Timestamp column differs from the PartitionKey and RowKey columns because it's automatically populated by the WAS system. In contrast, developers are required to insert into the PartitionKey and RowKey properties.

To summarize, the importance of PartitionKey and RowKey is mostly about query performance and transactional consistency. We explained query performance previously and it's largely dependent on the way the data is partitioned across storage nodes. But PartitionKeys also allow you to make changes to multiple entities as part of the same operation, allowing developers to roll back changes should any single operation fail. The requirement is that entities are part of the same entity group, which really means that entities share the same PartitionKey. Transactions are supported within a single PartitionKey.

The code in **Figure 3** illustrates instantiating an entity of type EmailAddressEntity (from **Figure 2**) and then inserting that entity into EmailAddressTable. Note that we're using the local storage emulator. This lets us run and test our code and data locally without connecting to a datacenter.

You can view your data in the Server Explorer pane in Visual Studio 2012, as shown in **Figure 4**, which makes the process of writing and testing code much easier. You can also attach Server Explorer to a real instance of your Windows Azure Tables in a datacenter.

The code in **Figure 5** illustrates how to query the data.

The code performs a simple query using PartitionKey and RowKey. Note that you can construct fairly complex queries using these filters because you can join them together in an ad hoc fashion. We build a query object using the combined filter. The final step is to simply execute the query and do whatever is needed with the EmailAddressEntity. The WAS Client Library greatly simplifies both the Create/Read/Update/Delete (CRUD) operations as well as the needed queries.

## What's Inside

We thought it might be helpful to take a slightly deeper look at the internal architecture of the WAS system, shown in **Figure 6**. Much of the following narrative is based on Brad Calder's paper referenced later in this article.

WAS is composed of a series of storage stamps across its eight datacenters. A storage stamp is a cluster of about 10 to 20 racks of storage nodes. Each rack sits in a separate fault domain. Every rack comes with redundant networking and power. Each storage stamp contains approximately 30PBs of raw storage.

To keep costs low, it's important to keep these storage stamps running above 70 percent utilization, which is measured in terms of capacity, transactions and bandwidth. Going above 90 percent is considered too high, though, as it leaves little headroom in the event of rack failures, when the system needs to do more with less.

Figure 5 Querying Windows Azure Tables

```
// Use the local storage emulator
var storageAccount = CloudStorageAccount.DevelopmentStorageAccount;
try
{
    // Create the table client
    CloudTableClient tableClient = storageAccount.CreateCloudTableClient();
    CloudTable emailAddressTable =
        tableClient.GetTableReference("EmailAddressTable");

    // Retrieve the entity with partition key of "microsoft.com"
    // and row key of "bterkaly"
    TableOperation retrieveBrunoEmail =
        TableOperation.Retrieve<EmailAddressEntity>(
            "microsoft.com", "bterkaly");

    // Retrieve entity
    EmailAddressEntity specificEntity =
        (EmailAddressEntity)emailAddressTable.Execute(retrieveBrunoEmail).Result;
    TableResult result =
        emailAddressTable.Execute(TableOperation.Retrieve<EmailAddressEntity>(
            "microsoft.com", "bterkaly"));

    // Pull the data out that you searched for
    // Do something with emailAddress and phoneNumber
    string emailAddress = specificEntity.EMailAddress;
    string phoneNumber = specificEntity.PhoneNumber;
}
catch (Exception ex)
{
    // Put the message in the Web page title (for testing purposes)
    // Real error messages should go to a proper log file
    this.Title = ex.Message.ToString();
    throw;
}
```





 Visual Studio  
2012 Ready

Sophisticated reports with fixed page layout  
Support for all .NET platforms  
Designers to empower end users  
Easy customization  
Flexible licensing

# ActiveReports 7

**ComponentOne**  
a division of GrapeCity®

Download your free trial @  
[componentone.com/ar7](http://componentone.com/ar7)

© 2013 GrapeCity, inc. All rights reserved. All other product and brand names are trademarks and/or registered trademarks of their respective holders.



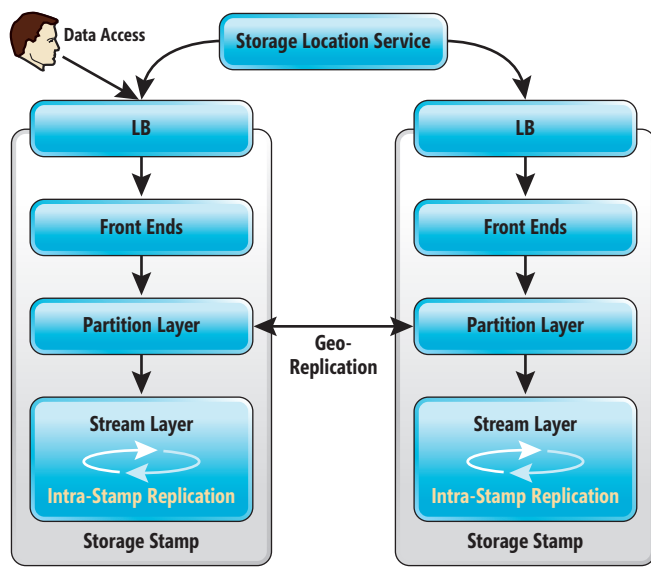


Figure 6 Windows Azure Storage Internals

## Storage Location Service

The developer has no direct control over the Storage Location Service (SLS). At the account level, not only does the SLS map account namespaces across all stamps, it's also responsible for disaster recovery, storage account allocation and load balancing. The SLS greatly simplifies the ability to add new storage in a datacenter. It can allocate new storage accounts to the new stamps for customers as well as load balance existing storage accounts from older stamps to the new stamps. All of these operations by the SLS are done automatically.

Let's look a little closer at the three layers that make up a storage stamp—stream, partition and front end (FE)—starting from the bottom.

The stream layer provides an internal interface the partition layer uses to read and write large files, and is responsible for core replication functionality. The stream layer also handles opening,

closing, deleting, renaming, reading, appending to and concatenating these large files. It doesn't concern itself with the semantics of objects that are in the stream of data.

The partition layer provides the data model for the different types of objects stored (tables, blobs, queues); the logic and semantics to process the different types of objects; a massively scalable namespace for the objects; load balancing to access objects across the available partition servers; transaction ordering and strong consistency for access to objects; and the geo-replication of data objects from the primary to the secondary region.

The partition layer also encapsulates an important internal data structure called an Object Table. There are several versions of the Object Table, including the Entity Table, which stores all entity rows for all accounts in the stamp. It's used to publicly expose Windows Azure Table data abstraction. The object table also interacts with the partition layer to ensure data consistency by ordering transactions with blobs, tables and queues.

The FE layer is composed of a set of stateless servers that take incoming requests. Upon receiving a request, an FE looks up the AccountName, authenticates and authorizes the request, then routes the request to the appropriate partition server in the partition layer (based on the PartitionName). To enhance performance, the FE maintains and caches a Partition Map, so that routing to the appropriate partition server is expedited on frequently accessed data.

## Wrapping Up

In this article, we've provided some high-level, actionable guidelines as well as some of the architectural details on how the WAS system is designed, and in particular how Windows Azure Tables can help you manage your data. We'd like to thank Brad Calder for some of his insights shared in "Windows Azure Storage: A Highly Available Cloud Storage Service with Strong Consistency," a recently published paper for the 23rd ACM Symposium on Operating Systems Principles (SOSP). You can download his paper at [bit.ly/tMIPus](http://bit.ly/tMIPus). ■

## Windows Azure Storage Client Library 2.0

**Back in late October 2012**, Microsoft released a new client-side storage library—Windows Azure Storage (WAS) Client Library 2.0—which dramatically improves usability, extensibility and performance when interacting with Windows Azure Tables. You can install the WAS Client Library 2.0 with NuGet from [bit.ly/YFeHuw](http://bit.ly/YFeHuw). This can be done within Visual Studio 2012. For a detailed look at some of the great new features, visit [bit.ly/VQSaUv](http://bit.ly/VQSaUv).

The new library includes some new approaches that improve functionality with respect to usability, extensibility and performance. One nice feature saves you from the hassle of worrying about serialization and deserialization logic when working with Plain Old C# Objects (POCO). Another cool feature is the EntityResolver, which allows you to perform client-side projections, so you can create objects on the fly based on only the information you're interested in. In short, you can convert directly from table entity data to a client object type without a separate table entity class type that deserializes every property individually. Another powerful technology is the IQueryable interface, which gives you an expressive way to define complex LINQ queries.

**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. Terkaly is also the author of two Windows Store apps, *Teach Kids Car Colors* and *Teach Kids Music*. You can read his blog at [blogs.msdn.com/brunoterkaly](http://blogs.msdn.com/brunoterkaly).

**RICARDO VILLOBO** 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 a master's degree in business administration from the University of Dallas, he works as a cloud architect in the Windows Azure CSV incubation group for Microsoft.

Terkaly and Villalobos jointly present at large industry conferences. They encourage readers to contact them for availability. Terkaly can be reached at [bterkaly@microsoft.com](mailto:bterkaly@microsoft.com) and Villalobos can be reached at [Ricardo.Villalobos@microsoft.com](mailto:Ricardo.Villalobos@microsoft.com).

**THANKS** to the following technical experts for reviewing this article:  
Brad Calder (Microsoft) and Jai Haridas (Microsoft)



# Spread

Microsoft Excel® compatibility in .NET  
 Easy and fast data binding  
 Dashboards in a cinch with charts & data visualizations  
 Info sharing across the enterprise, including Windows 8  
 Spreadsheet controls for COM, Windows Forms & ASP.NET,  
 Silverlight & WPF, and WinRT

**ComponentOne®**  
 a division of GrapeCity®

Download your free trial @  
[componentone.com/sp](http://componentone.com/sp)

© 2013 GrapeCity, inc. All rights reserved. All other product and brand names are trademarks and/or registered trademarks of their respective holders.





YOUR BACKSTAGE PASS TO THE MICROSOFT PLATFORM



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

# Sweet 127.0.0.1 chicago!

Visual Studio Live! is thrilled to be back in Chicago! Register for your backstage pass to the Microsoft Platform and 4 days of unbiased .NET training led by industry experts and Microsoft insiders.



**CHICAGO** | **MAY**  
**13-16, 2013**  
HILTON CHICAGO

**REGISTER  
TODAY!**

USE PROMO CODE CHAPR4



TURN THE PAGE FOR  
MORE EVENT DETAILS

[vslive.com/chicago](http://vslive.com/chicago)

## TOPICS WILL INCLUDE:

- ASP.NET
- Azure / Cloud Computing
- Cross-Platform Mobile
- Data Management
- HTML5 / JavaScript
- SharePoint / Office
- Windows 8 / WinRT
- WPF / Silverlight
- Visual Studio 2012 / .NET 4.5



## CONNECT WITH VISUAL STUDIO LIVE!



twitter.com/vslive – @VSLive



facebook.com – Search “VSLive”



linkedin.com – Join the “VSLive” group!

## AGENDA AT-A-GLANCE

Windows 8 / WinRT	WPF / Silverlight	ASP.NET	Visual Studio 2012 / .NET	SharePoint / Office
START TIME	END TIME	Visual Studio Live! Pre-Conference Workshops: Monday, May 13,		
7:30 AM	9:00 AM	Pre-Conference Workshop Registration		
9:00 AM	6:00 PM	MW01 - Workshop: Build a Windows 8 Application in a Day - Rockford Lhotka		
START TIME	END TIME	Visual Studio Live! Day 1: Tuesday, May 14, 2013		
7:00 AM	8:00 AM	Registration		
8:00 AM	9:00 AM	Keynote: To Be Announced		
9:15 AM	10:30 AM	T01 - A Primer in Windows 8 Development with WinJS - Philip Japikse	T02 - jQuery Fundamentals - Robert Boedigheimer	
10:45 AM	12:00 PM	T05 - Windows 8 Style Apps - Design Essentials - Billy Hollis	T06 - Hate JavaScript? Try TypeScript - Ben Hoelting	
12:00 PM	1:30 PM	Lunch		
1:30 PM	2:45 PM	T09 - MVVM in Practice aka ""Code Behind"" - Free XAML - Tiberiu Covaci	T10 - Tips for Building Multi-Touch Enabled Web Sites - Ben Hoelting	
3:00 PM	4:15 PM	T13 - New XAML Controls in Windows 8 - Billy Hollis	T14 - Beyond Hello World: A Practical Introduction to Node.js - Rick Garibay	
4:15 PM	4:45 PM	Networking Break		
4:45 PM	6:00 PM	T17 - Make Your App Alive with Tiles and Notifications - Ben Dewey	T18 - Build Speedy Azure Applications with HTML5 and Web Sockets Today - Rick Garibay	
6:00 PM	7:30 PM	Exhibitor Reception		
START TIME	END TIME	Visual Studio Live! Day 2: Wednesday, May 15, 2013		
7:00 AM	8:00 AM	Registration		
8:00 AM	9:00 AM	Keynote: To Be Announced		
9:15 AM	10:30 AM	W01 - Building Your First Windows Phone 8 Application - Brian Peek	W02 - What's New in Azure for Developers - Vishwas Lele	
10:45 AM	12:00 PM	W05 - Sharing Code Between Windows 8 and Windows Phone 8 Apps - Ben Dewey	W06 - In Depth Azure IaaS - Vishwas Lele	
12:00 PM	1:30 PM	Round Table Lunch		
1:30 PM	2:45 PM	W09 - Connecting to Data from Windows Phone 8 - Christopher Woodruff	W10 - Moving Web Apps to the Cloud - Eric D. Boyd	
3:00 PM	4:15 PM	W13 - Building a Windows Runtime Component with C# - Brian Peek	W14 - IaaS in Windows Azure with Virtual Machines - Eric D. Boyd	
4:15 PM	4:45 PM	Networking Break		
4:45 PM	6:00 PM	W17 - Demystifying the Microsoft UI Technology Roadmap - Brian Noyes	W18 - Bringing Open Source to Windows Azure: A Match Made in Heaven - Jesus Rodriguez	
6:30 PM	8:30 PM	Blues after Dark at Buddy Guy's Legends		
START TIME	END TIME	Visual Studio Live! Day 3: Thursday, May 16, 2013		
7:30 AM	8:00 AM	Registration		
8:00 AM	9:15 AM	TH01 - Building Extensible XAML Client Apps - Brian Noyes	TH02 - JavaScript, Meet Cloud: Node.js on Windows Azure - Sasha Goldshtein	
9:30 AM	10:45 AM	TH05 - Migrating from WPF or Silverlight to WinRT - Rockford Lhotka	TH06 - Using Windows Azure to Build the Next Generation of Mobile Applications - Jesus Rodriguez	
11:00 AM	12:15 PM	TH09 - Managing the .NET Compiler - Jason Bock	TH10 - Cloud Backends for Your Mobile Apps: Windows Azure Mobile Services and Parse - Sasha Goldshtein	
12:15 PM	1:30 PM	Lunch		
1:30 PM	2:45 PM	TH13 - Understanding Dependency Injection and Those Pesky Containers - Miguel Castro	TH14 - Using Windows Azure for Solving Identity Management Challenges - Michael Collier	
3:00 PM	4:15 PM	TH17 - Static Analysis in .NET - Jason Bock	TH18 - Elevating Windows Azure Deployments - Michael Collier	
4:30 PM	5:30 PM	Conference Wrap-Up - Andrew Brust, Moderator, Rockford Lhotka, Miguel Castro,		

\*Speakers and sessions subject to change





# CHICAGO | MAY 13-16, 2013

HILTON CHICAGO

Azure / Cloud Computing

Data Management

HTML5 / JavaScript

Cross-Platform Mobile

2013 (Separate entry fee required)

MW02 - Workshop: SQL Server 2012 - Andrew Brust & Leonard Lobel

MW03 - Workshop: Happy ALM with Visual Studio 2012 and Team Foundation Server 2012 - Brian Randell

T03 - Big Data-BI Fusion: Microsoft HDInsight & MS BI - Andrew Brust

T04 - Microsoft Session To Be Announced

T07 - Getting to Know the BI Semantic Model - Andrew Brust

T08 - IntelliTrace, What is it and How Can I Use it to My Benefit? - Marcel de Vries

T11 - Session To Be Announced

T12 - Team Foundation Server 2012 Builds: Understand, Configure, and Customize - Benjamin Day

T15 - Busy Developer's Guide to MongoDB - Ted Neward

T16 - Modern ALM and the DevOps Story - Brian Randell

T19 - Busy Developer's Guide to Cassandra - Ted Neward

T20 - Patterns for Parallel Programming - Tiberiu Covaci

W03 - SQL Server Data Tools - Leonard Lobel

W04 - Design for Testability: Mocks, Stubs, Refactoring, and User Interfaces - Benjamin Day

W07 - Working with Client-Side HTML5 Storage Technologies - Gil Fink

W08 - Microsoft Session To Be Announced

W11 - LINQ Performance and Scalability - Jim Wooley

W12 - Microsoft Session To Be Announced

W15 - OData - Oh Yeah - Gil Fink

W16 - Build Modern Collaborative Solutions with Office 2013, "Napa" Office 365 Development Tools, and SharePoint 2013 - Brian Randell

W19 - Not Just a Designer: Code First and Entity Framework - Gil Fink

W20 - Unit Testing in SharePoint - Jim Wooley

TH03 - Improving Web Performance - Robert Boedigheimer

TH04 - Sharing Up to 80% of Code Building Mobile Apps for iOS, Android, WP 8 and Windows 8 - Marcel de Vries

TH07 - Controlling ASP.NET MVC4 - Philip Japikse

TH08 - iOS Development Survival Guide for the .NET Guy - Nick Landry

TH11 - MVC for WebForms Developers: Comparing and Contrasting - Miguel Castro

TH12 - Session To Be Announced

TH15 - Creating Web Sites Using Visual Studio LightSwitch - Michael Washington

TH16 - Building Multi-Platform Mobile Apps with Push Notifications - Nick Landry

TH19 - Building Single Page Web Applications with HTML5, ASP.NET MVC4 and Web API - Marcel de Vries

TH20 - Create HTML5 Mobile Websites with Visual Studio LightSwitch - Michael Washington

Marcel de Vries, Jason Bock



Visual Studio Live! Chicago  
Blues After Dark Reception at  
Buddy Guy's Legends

**"SEVERAL OF THE PRESENTATIONS WERE CUTTING EDGE – THEY WOULD HAVE INSIDER TIPS THAT YOU CAN'T EASILY SEARCH FOR OR WOULDN'T KNOW TO LOOK FOR."**

John Kilic  
Web Application Developer  
Grand Canyon University

Register at  
[vslive.com/chicago](http://vslive.com/chicago)

Use Promo Code CHAPR4



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



# A Tour of C++/CX

Thomas Petchel

**Ready to write** your first Windows Store app? Or have you already been writing Windows Store apps using HTML/JavaScript, C# or Visual Basic, and you're curious about the C++ story?

With Visual C++ component extensions (C++/CX), you can take your existing skills to new heights by combining C++ code with the rich set of controls and libraries provided by the Windows Runtime (WinRT). And if you're using Direct3D, you can really make your apps stand out in the Windows Store.

When some people hear about C++/CX they think they have to learn a whole new language, but the fact is that for the vast majority of cases you'll be dealing with just a handful of nonstandard language elements such as the `^` modifier or the *ref new* keywords. Furthermore, you'll only use these elements at the *boundary* of your app, that is, only when you need to interact with the Windows Runtime.

## This article discusses:

- When to use C++/CX
- How C++/CX works
- A sample tic-tac-toe game
- Creating the game UI
- Creating a game library
- Using interfaces
- Using the Windows Runtime C++ Template Library

## Technologies discussed:

C++/CX, Windows Runtime

## Code download available at:

[archive.msdn.microsoft.com/mag201304/C++CX](http://archive.msdn.microsoft.com/mag201304/C++CX)

Your portable ISO C++ will still act as the workhorse of your app. Perhaps best of all, C++/CX is 100 percent native code. Although its syntax resembles C++/Common Language Infrastructure (CLI), your app won't bring in the CLR unless you want it to.

Whether you have existing C++ code that was already tested or just prefer the flexibility and performance of C++, rest assured that with C++/CX you don't have to learn a whole new language. In this article you'll learn what makes the C++/CX language extensions for building Windows Store apps unique, and when to use C++/CX to build your Windows Store app.

## Why Choose C++/CX?

Every app has its own unique set of requirements, just as every developer has his own unique skills and abilities. You can successfully create a Windows Store app using C++, HTML/JavaScript or the Microsoft .NET Framework, but here are some reasons why you might choose C++:

- You prefer C++ and have existing skills.
- You want to take advantage of code that you've already written and tested.
- You want to use libraries such as Direct3D and C++ AMP to fully unleash the hardware's potential.

The answer doesn't have to be one or the other—you can also mix and match languages. For example, when I wrote the Bing Maps Trip Optimizer sample ([bit.ly/13hkJhA](http://bit.ly/13hkJhA)), I used HTML and JavaScript to define the UI and C++ to perform the background processing. The background process essentially solves the “traveling salesman” problem. I used the Parallel Patterns Library (PPL) ([bit.ly/155DPtQ](http://bit.ly/155DPtQ)) in a WinRT C++ component to run my algorithm in parallel on all available CPUs to improve overall performance. This would have been difficult to do from just JavaScript alone!



# 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](http://www.aspose.com)**

EU Sales: +44 (0) 141 416 1112  
[sales.europe@aspose.com](mailto:sales.europe@aspose.com)  
AU Sales: +61 2 8003 5926  
[sales.asiapacific@aspose.com](mailto:sales.asiapacific@aspose.com)

US Sales: 1.888.277.6734  
[sales@aspose.com](mailto:sales@aspose.com)



## How Does C++/CX Work?

At the heart of any Windows Store app is the Windows Runtime. At the heart of the Windows Runtime is the application binary interface (ABI). WinRT libraries define metadata through Windows metadata (.winmd) files. A .winmd file describes the public types that are available, and its format resembles the format that's used in .NET Framework assemblies. In a C++ component, the .winmd file contains only metadata; the executable code resides in a separate file. This is the case for the WinRT components that are included with Windows. (For .NET Framework languages, the .winmd file contains both the code and the metadata, just like a .NET Framework assembly.) You can view this metadata from the MSIL Disassembler (ILDASM) or any CLR metadata reader. **Figure 1** shows what Windows.Foundation.winmd, which contains many of the fundamental WinRT types, looks like in ILDASM.

The ABI is built using a subset of COM to enable the Windows Runtime to interact with multiple languages. In order to call WinRT APIs, the .NET Framework and JavaScript require projections that are specific to each language environment. For example, the underlying WinRT string type, HSTRING, is represented as System.String in .NET, a String object in JavaScript and the Platform::String ref class in C++/CX.

Although C++ can directly interact with COM, C++/CX aims to simplify this task through:

- Automatic reference counting. WinRT objects are reference-counted and typically heap-allocated (no matter which language uses them). Objects are destroyed when their reference count reaches zero. The benefit that C++/CX offers is that the reference counting is both automatic and uniform. The ^ syntax enables both of these.

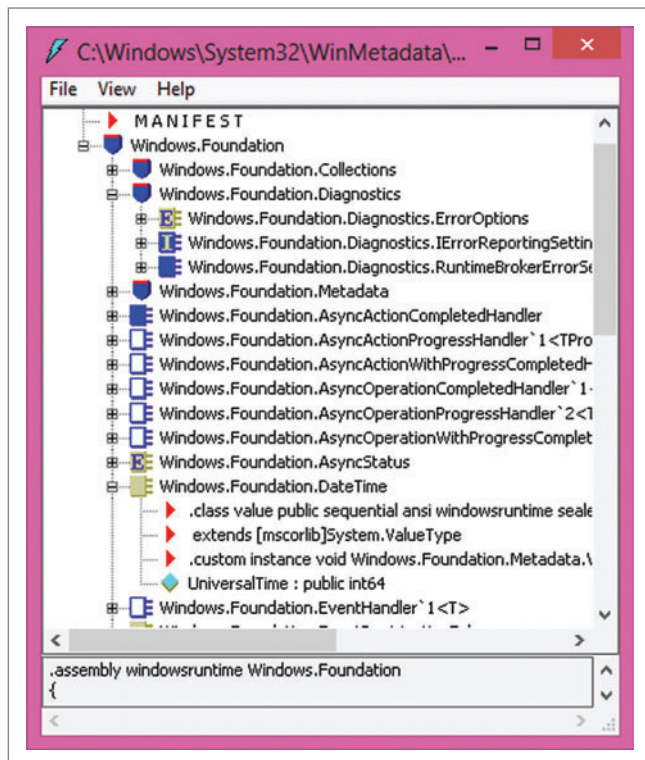


Figure 1 Inspecting Windows.Foundation.winmd with ILDASM

- Exception handling. C++/CX relies on exceptions, and not error codes, to indicate failures. Underlying COM HRESULT values are translated to WinRT exception types.
- An easy-to-use syntax for consuming the WinRT APIs, while still maintaining high performance.
- An easy-to-use syntax for creating new WinRT types.
- An easy-to-use syntax for performing type conversion, working with events and other tasks.

And remember, although C++/CX borrows the C++/CLI syntax, it produces pure native code. You can also interact with the Windows Runtime by using the Windows Runtime C++ Template Library (WRL), which I'll introduce later. However, I hope that after using C++/CX you'll agree it makes sense. You get the performance and control of native code, you don't have to learn COM and your code that interacts with the Windows Runtime will be as succinct as possible—letting you focus on the core logic that makes your app unique.

Although C++/CX borrows the C++/CLI syntax, it produces pure native code.

C++/CX is enabled through the /ZW compiler option. This switch is set automatically when you use Visual Studio to create a Windows Store project.

## A Tic-Tac-Toe Game

I think the best way to learn a new language is to actually build something with it. To demonstrate the most common parts of C++/CX, I wrote a Windows Store app that plays tic-tac-toe (or depending on where you grew up, you might call it "noughts and crosses" or "Xs and Os").

For this app, I used the Visual Studio Blank App (XAML) template. I named the project TicTacToe. This project uses XAML to define the app's UI. I won't focus much on the XAML. To learn more about that side of things, see Andy Rich's article, "Introducing C++/CX and XAML" (msdn.microsoft.com/magazine/jj651573), in the 2012 Windows 8 Special Issue.

I also used the Windows Runtime Component template to create a WinRT component that defines the logic of the app. I love code reuse, so I created a separate component project so that anyone can use the core game logic in any Windows Store app using XAML and C#, Visual Basic, or C++.

**Figure 2** shows what the app looks like.

When I worked on the Hilo C++ project (bit.ly/Wy5E92), I fell in love with the Model-View-ViewModel (MVVM) pattern. MVVM is an architectural pattern that helps you separate the appearance, or view, of your app, from its underlying data, or model. The view model connects the view to the model. Although I didn't use full-on MVVM for my tic-tac-toe game, I found that using data binding to separate the UI from app logic made the app easier to write, more readable and easier to maintain in the future. To learn more about how we used MVVM in the Hilo C++ project, see bit.ly/XigPg.



To connect the app to the WinRT component, I added a reference to the TicTacToeLibrary project from the TicTacToe project's Property Pages dialog.

By simply setting the reference, the TicTacToe project has access to all of the public C++/CX types in the TicTacToeLibrary project. You don't have to specify any `#include` directives or do anything else.

## Creating the TicTacToe UI

As I said earlier, I won't go much into the XAML, but in my vertical layout, I set up one area to display the score, one for the main play area and one to set up the next game (you can see the XAML in the file `MainPage.xaml` in the accompanying code download). Again, I used data binding pretty extensively here.

The definition of the `MainPage` class (`MainPage.h`) is shown in **Figure 3**.

So what's in `MainPage.g.h`? A `.g.h` file contains a compiler-generated partial class definition for XAML pages. Basically, these partial definitions define the required base classes and member variables for any XAML element that has the `x:Name` attribute. Here's `MainPage.g.h`:

```
namespace TicTacToe
{
    partial ref class MainPage :
    public ::Windows::UI::Xaml::Controls::Page,
    public ::Windows::UI::Xaml::Markup::IComponentConnector
    {
    public:
        void InitializeComponent();
        virtual void Connect(int connectionId, ::Platform::Object^ target);

    private:
        bool _contentLoaded;

    };
}
```

The *partial* keyword is important because it enables a type declaration to span files. In this case, `MainPage.g.h` contains compiler-generated parts, and `MainPage.h` contains the additional parts that I define.

Notice the `public` and `ref class` keywords in the `MainPage` declaration. One difference between C++/CX and C++ is the concept of class accessibility. If you're a .NET programmer, you'll be familiar with this. Class accessibility means whether a type or method is visible in metadata, and therefore accessible from external components. A C++/CX type can be `public` or `private`. `Public` means that the `MainPage` class can be accessed outside of the module (for example, by the Windows Runtime or by another WinRT component). A `private` type can be accessed only inside the module. `Private` types give you more freedom to use C++ types in public methods, which isn't possible with `public` types. In this case, the `MainPage` class is `public` so that it's accessible to XAML. I'll look at some examples of `private` types later.

The `ref class` keywords tell the compiler that this is a WinRT type and not a C++ type. A `ref class` is allocated on the heap and its lifetime is reference-counted. Because `ref types` are reference-counted, their lifetimes are deterministic. When the last reference to a `ref`

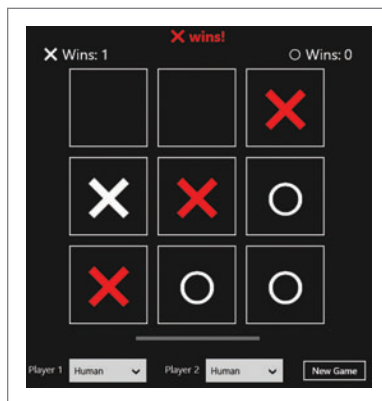


Figure 2 The TicTacToe App

type object is released, its destructor is called and the memory for that object is released. Compare this to .NET, where lifetimes are less deterministic and garbage collection is used to free memory.

When you instantiate a `ref type`, you typically use the `^` (pronounced “hat”) modifier. The `^` modifier is similar to a C++ pointer (`*`), but it tells the compiler to insert code to manage the object's reference count automatically and delete the object when its reference count reaches zero.

To create a plain old data (POD) structure, use a value class or value struct. Value types have a fixed size and consist of fields only. Unlike

`ref types`, they have no properties. `Windows::Foundation::DateTime` and `Windows::Foundation::Rect` are two examples of WinRT value types. When you instantiate value types, you don't use the `^` modifier:

```
Windows::Foundation::Rect bounds(0, 0, 100, 100);
```

Also notice that `MainPage` is declared as `sealed`. The `sealed` keyword, which is similar to the C++11 `final` keyword, prevents further derivation of that type. `MainPage` is `sealed` because any `public ref type` that has a `public` constructor must also be declared as `sealed`. This is because the runtime is language-agnostic and not all languages (for example, JavaScript) understand inheritance.

Now direct your attention to the `MainPage` members. The `m_processor` member variable (the `GameProcessor` class is defined in the WinRT component project—I'll talk about that type later) is `private` simply because the `MainPage` class is `sealed` and there's no possibility that a derived class can use it (and, in general, data members should be `private` when possible to enforce encapsulation). The `OnNavigatedTo` method is protected because the `Windows::UI::Xaml::Controls::Page` class, from which `MainPage` derives, declares this method as protected. The constructor and the `Processor` property must be accessed by XAML, and therefore both are `public`.

You're already familiar with `public`, `protected` and `private` access specifiers; their meanings in C++/CX are the same as in C++. To learn about internal and other C++/CX specifiers, see [bit.ly/Xqb5Xe](http://bit.ly/Xqb5Xe). You'll see an example of internal later on.

A `ref class` may have only publicly accessible types in its `public` and `protected` sections—that is, only primitive types, `public ref` or `public value types`. Conversely, a C++ type can contain `ref types` as member variables, in method signatures and in local function variables. Here's an example from the Hilo C++ project:

```
std::vector<Windows::Storage::IStorageItem^> m_createdFiles;
```

The Hilo team uses `std::vector` and not `Platform::Collections::Vector` for this `private` member variable because we don't expose the collection outside of the class. Using `std::vector` helps us use C++ code as much as possible and makes its intention clear.

Moving on to the `MainPage` constructor:

```
MainPage::MainPage() : m_processor(ref new TicTacToeLibrary::GameProcessor())
{
    InitializeComponent();
    DataContext = m_processor;
}
```

I use the `ref new` keywords to instantiate the `GameProcessor` object. Use *ref new* instead of *new* to construct WinRT reference

type objects. When you're creating objects in functions, you can use the C++ auto keyword to reduce the need for specifying the type name or use of ^:

```
auto processor = ref new TicTacToeLibrary::GameProcessor();
```

## Creating the TicTacToe Library

The library code for the TicTacToe game contains a mixture of C++ and C++/CX. For this app, I pretended that I had some existing C++ code that I'd already written and tested. I incorporated this code directly, adding C++/CX code only to connect the internal implementation to XAML. In other words, I used C++/CX only to bridge the two worlds together. Let's walk through some of the important parts of the library and highlight any C++/CX features not already discussed.

The GameProcessor class serves as the data context for the UI (think view model if you're familiar with MVVM). I used two attributes, BindableAttribute and WebHostHiddenAttribute, when declaring this class (like .NET, you can omit the "Attribute" part when you declare attributes):

```
[Windows::UI::Xaml::Data::Bindable]
[Windows::Foundation::Metadata::WebHostHidden]
public ref class GameProcessor sealed : public Common::BindableBase
```

The BindableAttribute produces metadata that tells the Windows Runtime that the type supports data binding. This ensures that all the public properties of the type are visible to the XAML components. I derive from BindableBase to implement the functionality required to make binding work. Because BindableBase is intended for use by XAML and not JavaScript, it uses the WebHostHiddenAttribute ([bit.ly/ZsAOV3](http://bit.ly/ZsAOV3)) attribute. Per convention, I also marked the GameProcessor class with this attribute to essentially hide it from JavaScript.

I separated GameProcessor's properties into public and internal sections. The public properties are exposed to XAML; the internal properties are exposed only to other types and functions in the library. I felt that making this distinction helps make the intent of the code more obvious.

One common property usage pattern is binding collections to XAML:

```
property Windows::Foundation::Collections::IObservableVector<Cell>^ Cells
{
    Windows::Foundation::Collections::IObservableVector<Cell>^ get()
    { return m_cells; }
}
```

This property defines the model data for the cells that appear on the grid. When the value of Cells changes, the XAML is updated automatically. The type of the property is IObservableVector, which is one of several types defined specifically for C++/CX to enable full interoperability with the Windows Runtime. The Windows Runtime defines language-independent collection interfaces, and each language implements those interfaces in its own way. In C++/CX, the Platform::Collections namespace provides types such as Vector and Map that provide concrete implementations for these collections interfaces. Therefore, I can declare the Cells property as IObservableVector but back that property by a Vector object, which is specific to C++/CX:

```
Platform::Collections::Vector<Cell>^ m_cells;
```

So when do you use Platform::String and Platform::Collections versus the standard types and collections? For example, should you use std::vector or Platform::Collections::Vector to store

your data? As a rule of thumb, I use Platform functionality when I plan to work primarily with the Windows Runtime, and standard types such as std::wstring and std::vector for my internal or computationally intensive code. You can also easily convert between Vector and std::vector when you need to. You can create a Vector from a std::vector or you can use to\_vector to create a std::vector from a Vector:

```
std::vector<int> more_numbers =
    Windows::Foundation::Collections::to_vector(result);
```

There's a copy cost associated when marshaling between the two vector types, so again, consider which type is appropriate in your code.

Another common task is converting between std::wstring and Platform::String. Here's how:

```
// Convert std::wstring to Platform::String.
std::wstring sl(L"Hello");
auto s2 = ref new Platform::String(sl.c_str());
// Convert back from Platform::String to std::wstring.
// String::Data returns a C-style string, so you don't need
// to create a std::wstring if you don't need it.
std::wstring s3(s2->Data());
// Here's another way to convert back.
std::wstring s4(begin(s2), end(s2));
```

There are two interesting points to note in the GameProcessor class implementation (GameProcessor.cpp). First, I use only standard C++ to implement the checkEndOfGame function. This is one place where I wanted to illustrate how to incorporate existing C++ code that I'd already written and tested.

The second point is the use of asynchronous programming. When it's time to switch turns, I use the PPL task class to process the computer players in the background, as shown in **Figure 4**.

If you're a .NET programmer, think of task and its then method as the C++ version of async and await in C#. Tasks are available from any C++ program, but you'll use them throughout your C++/CX code to keep your Windows Store app fast and fluid. To learn more about async programming in Windows Store apps, read Artur Laksberg's February 2012 article, "Asynchronous Programming in C++ Using PPL" ([msdn.microsoft.com/magazine/hh781020](http://msdn.microsoft.com/magazine/hh781020)), and the MSDN Library article at [msdn.microsoft.com/library/hh750082](http://msdn.microsoft.com/library/hh750082).

The Cell class models a cell on the game board. Two new things that this class demonstrates are events and weak references.

Figure 3 The Definition of the MainPage Class

```
#pragma once

#include "MainPage.g.h"

namespace TicTacToe
{
    public ref class MainPage sealed
    {
    public:
        MainPage();

        property TicTacToeLibrary::GameProcessor^ Processor
        {
            TicTacToeLibrary::GameProcessor^ get() { return m_processor; }
        }
    protected:
        virtual void OnNavigatedTo(
            Windows::UI::Xaml::Navigation::NavigationEventArgs^ e) override;
    private:
        TicTacToeLibrary::GameProcessor^ m_processor;
    };
}
```

# WPF lives!



➔ **XCEED Business Suite for WPF**

The essential set of WPF controls for all your line-of-business solutions. Includes the industry-leading **Xceed DataGrid for WPF**.  
A total of 85 tools!



**Figure 4 Using the PPL Task Class to Process Computer Players in the Background**

```
void GameProcessor::SwitchPlayers()
{
    // Switch player by toggling pointer.
    m_currentPlayer = (m_currentPlayer == m_player1) ? m_player2 : m_player1;

    // If the current player is computer-controlled, call the ThinkAsync
    // method in the background, and then process the computer's move.
    if (m_currentPlayer->Player == TicTacToeLibrary::PlayerType::Computer)
    {
        m_currentThinkOp =
            m_currentPlayer->ThinkAsync(ref new Vector<wchar_t>(m_gameBoard));
        m_currentThinkOp->Progress =
            ref new AsyncOperationProgressHandler<uint32, double>([this](
                IAsyncOperationWithProgress<uint32, double>^ asyncInfo, double value)
            {
                (void) asyncInfo; // Unused parameter

                // Update progress bar.
                m_backgroundProgress = value;
                OnPropertyChanged("BackgroundProgress");
            });

        // Create a task that wraps the async operation. After the task
        // completes, select the cell that the computer chose.
        create_task(m_currentThinkOp).then([this](task<uint32> previousTask)
        {
            m_currentThinkOp = nullptr;

            // I use a task-based continuation here to ensure this continuation
            // runs to guarantee the UI is updated. You should consider putting
            // a try/catch block around calls to task::get to handle any errors.
            uint32 move = previousTask.get();

            // Choose the cell.
            m_cells->GetAt(move)->Select(nullptr);

            // Reset the progress bar.
            m_backgroundProgress = 0.0;
            OnPropertyChanged("BackgroundProgress");

        }, task_continuation_context::use_current());
    }
}
```

The grid for the TicTacToe play area consists of `Windows::UI::Xaml::Controls::Button` controls. A `Button` control raises a `Click` event, but you can also respond to user input by defining an  `ICommand` object that defines the contract for commanding. I use the  `ICommand` interface instead of the `Click` event so that `Cell` objects can respond directly. In the XAML for the buttons that define the cells, the `Command` property binds to the `Cell::SelectCommand` property:

```
<Button Width="133" Height="133" Command="{Binding SelectCommand}"
        Content="{Binding Text}" Foreground="{Binding ForegroundBrush}"
        BorderThickness="2" BorderBrush="White" FontSize="72"/>
```

I used the `Hilo DelegateCommand` class to implement the  `ICommand` interface. `DelegateCommand` holds the function to call when the command is issued, and an optional function that determines whether the command can be issued. Here's how I set up the command for each cell:

```
m_selectCommand = ref new DelegateCommand(
    ref new ExecutedDelegate(this, &Cell::Select), nullptr);
```

You'll commonly use predefined events when doing XAML programming, but you can also define your own events. I created an event that's raised when a `Cell` object is selected. The `GameProcessor` class handles this event by checking whether the game is over and switching the current player if needed.

To create an event, you must first create a delegate type. Think of a delegate type as a function pointer or a function object:

```
delegate void CellSelectedHandler(Cell^ sender);
```

I then create an event for each `Cell` object:

```
event CellSelectedHandler^ CellSelected;
```

Here's how the `GameProcessor` class subscribes to the event for each cell:

```
for (auto cell : m_cells)
{
    cell->CellSelected += ref new CellSelectedHandler(
        this, &GameProcessor::CellSelected);
}
```

A delegate that's constructed from a `^` and a pointer-to-member function (PMF) holds only a weak reference to the `^` object, so this construct won't cause circular references.

Here's how `Cell` objects raise the event when they're selected:

```
void Cell::Select(Platform::Object^ parameter)
{
    (void)parameter;

    auto gameProcessor = m_gameProcessor.Resolve<GameProcessor>();
    if (m_mark == L'\0' && gameProcessor != nullptr &&
        !gameProcessor->IsThinking && !gameProcessor->CanCreateNewGame)
    {
        m_mark = gameProcessor->CurrentPlayer->Symbol;
        OnPropertyChanged("Text");

        CellSelected(this);
    }
}
```

What's the purpose of the `Resolve` call in the preceding code? Well, the `GameProcessor` class holds a collection of `Cell` objects, but I want each `Cell` object to be able to access its parent `GameProcessor`. If `Cell` held a strong reference to its parent—in other words, a `GameProcessor^`—I'd create a circular reference. Circular references can cause objects to never be freed because the mutual association causes both objects to always have at least one reference. To avoid this, I create a `Platform::WeakReference` member variable and set it from the `Cell` constructor (think very carefully about lifetime management and what objects own what!):

```
Platform::WeakReference m_gameProcessor;
```

When I call `WeakReference::Resolve`, `nullptr` is returned if the object no longer exists. Because `GameProcessor` owns `Cell` objects, I expect the `GameProcessor` object to always be valid.

I try to avoid the need to break circular references because it can make code less maintainable.

In the case of my TicTacToe game, I can break the circular reference each time a new game board is created, but in general, I try to avoid the need to break circular references because it can make code less maintainable. Therefore, when I have a parent-child relationship and children need to access their parent, I use weak references.

## Working with Interfaces

To distinguish between human and computer players, I created an `IPlayer` interface with concrete implementations `HumanPlayer` and `ComputerPlayer`. The `GameProcessor` class holds two `IPlayer` objects—one for each player—and an additional reference to the current player:

Figure 5 The IPlayer Interface

```
private interface class IPlayer
{
    property PlayerType Player
    {
        PlayerType get();
    }

    property wchar_t Symbol
    {
        wchar_t get();
    }

    virtual Windows::Foundation::IAsyncOperationWithProgress<uint32, double>^
        ThinkAsync(Windows::Foundation::Collections::IVector<wchar_t>^ gameBoard);
};
```

```
IPlayer^ m_player1;
IPlayer^ m_player2;
IPlayer^ m_currentPlayer;
```

Figure 5 shows the IPlayer interface.

Because the IPlayer interface is private, why didn't I just use C++ classes? To be honest, I did it to show how to create an interface and how to create a private type that isn't published to metadata. If I were creating a reusable library, I might declare IPlayer as a public interface so other apps could use it. Otherwise, I might choose to stick with C++ and not use a C++/CX interface.

The ComputerPlayer class implements ThinkAsync by performing the minimax algorithm in the background (see the file ComputerPlayer.cpp in the accompanying code download to explore this implementation).

Minimax is a common algorithm when creating artificial intelligence components for games such as tic-tac-toe. You can learn more about minimax in the book, "Artificial Intelligence: A Modern Approach" (Prentice Hall, 2010), by Stuart Russell and Peter Norvig.

I adapted Russell and Norvig's minimax algorithm to run in parallel by using the PPL (see minimax.h in the code download). This was a great opportunity to use pure C++11 to write the processor-intensive part of my app. I've yet to beat the computer and have never seen the computer beat itself in a computer-versus-computer game. I admit this doesn't make for the most exciting game, so here's your call to action: Add additional logic to make the game winnable. A basic way to do this would be to have the computer make random selections at random times. A more sophisticated way would be to have the computer purposely choose a less-optimal move at random times. For bonus points, add a slider control to the UI that adjusts the game's difficulty (the less difficult, the more the computer either chooses a less-optimal move or at least a random one).

For the HumanPlayer class, ThinkAsync has nothing to do, so I throw Platform::NotImplementedException. This requires that I test the IPlayer::Player property first, but it saves me a task:

```
IAsyncOperationWithProgress<uint32, double>^
HumanPlayer::ThinkAsync(IVector<wchar_t>^ gameBoard)
{
    (void) gameBoard;

    throw ref new NotImplementedException();
}
```

## The WRL

There's a great sledgehammer available in your toolbox for when C++/CX doesn't do what you need or when you prefer to work

directly with COM: the WRL. For example, when you create a media extension for Microsoft Media Foundation, you must create a component that implements both COM and WinRT interfaces. Because C++/CX ref classes can only implement WinRT interfaces, to create a media extension you must use the WRL because it supports the implementation of both COM and WinRT interfaces. To learn more about WRL programming, see [bit.ly/YE8Dxu](http://bit.ly/YE8Dxu).

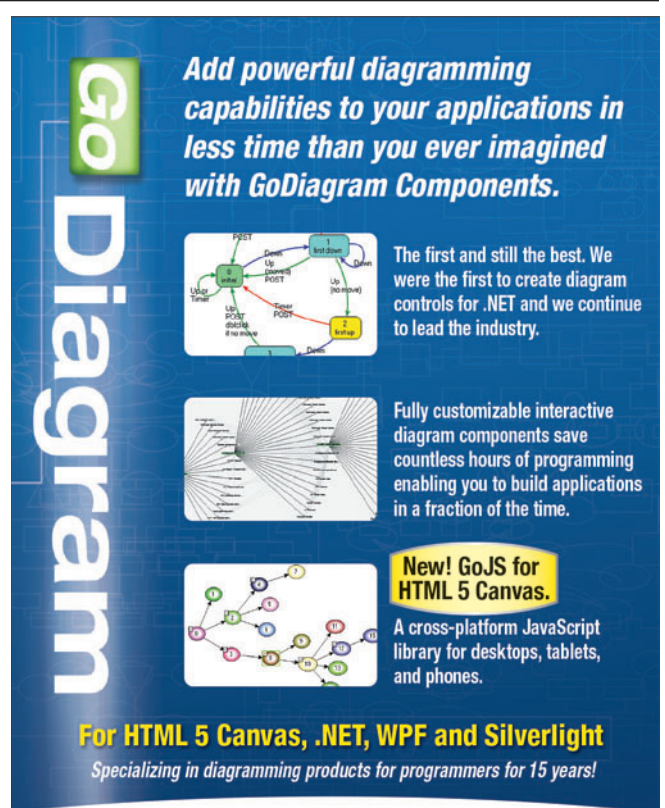
## Going Deeper

At first I had misgivings about C++/CX extensions, but they soon became second nature, and I like them because they enable me to write Windows Store apps quickly and use modern C++ idioms. If you're a C++ developer, I highly recommend you at least give them a shot.

I reviewed just some of the common patterns you'll encounter when writing C++/CX code. Hilo, a photo app using C++ and XAML, goes deeper and is much more complete. I had a great time working on the Hilo C++ project, and I actually refer back to it often as I write new apps. I recommend that you check it out at [bit.ly/15xZ5JL](http://bit.ly/15xZ5JL). ■

**THOMAS PETCHEL** works as a senior programming writer in the Microsoft Developer Division. He has spent the past eight years with the Visual Studio team creating documentation and code samples for the developer audience.

**THANKS** to the following technical experts for reviewing this article: Michael Blome (Microsoft) and James McNellis (Microsoft)



**GoDiagram**

**Add powerful diagramming capabilities to your applications in less time than you ever imagined with GoDiagram Components.**

The first and still the best. We were the first to create diagram controls for .NET and we continue to lead the industry.

Fully customizable interactive diagram components save countless hours of programming enabling you to build applications in a fraction of the time.

**New! GoJS for HTML 5 Canvas.**

A cross-platform JavaScript library for desktops, tablets, and phones.

**For HTML 5 Canvas, .NET, WPF and Silverlight**

Specializing in diagramming products for programmers for 15 years!

**Powerful, flexible, and easy to use.**

Find out for yourself with our **FREE** Trial Download  
with full support at: [www.godiagram.com](http://www.godiagram.com)





**TECHMENTOR**  
CONFERENCES

YEARS OF IT

**15**

EDUCATION

**LAS VEGAS**

**Sept 30-Oct 4, 2013**

The Tropicana, Las Vegas

# GET INSIDE THE IT CLASSROOM

*In-Depth Training for IT Pros*

TechMentor is returning to Las Vegas for 5 days of information-packed sessions and workshops! Surrounded by your fellow IT professionals, you will receive immediately usable education that will keep you relevant in the workforce.







Register today and get inside the IT classroom! There's a lot of knowledge out there just waiting for you – TechMentor is an experience you won't want to miss.

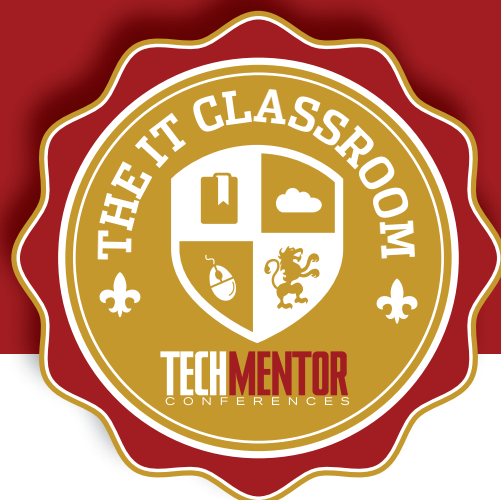
**Save  
\$300!**



Register before July 31  
Use Promo Code TMLVAPR

**TechMentor session topics include:**

- ✦ Windows PowerShell and Automation
- ✦ Cisco and Networking Infrastructure
- ✦ Windows Server Management
- ✦ Windows Client Management
- ✦ Cloud and Virtualization
- ✦ Identity, Access Management and Security
- ✦ Performance Tuning and Troubleshooting
- ✦ Mobility and BYOD
- ✦ Messaging and Collaboration
- ✦ Microsoft Certification Training



# Exploring the JavaScript API for Office: Data Binding and Custom XML Parts

Stephen Oliver and Eric Schmidt

This article is **Part 3** in a series of in-depth walkthroughs of the JavaScript API for Office. This article continues the examination of key aspects of the API, focusing on data binding and support for working with custom XML parts. Part 1, “Exploring the New JavaScript API for Office” ([msdn.microsoft.com/magazine/jj891051](https://msdn.microsoft.com/magazine/jj891051)), provides a broad overview of the object model. Part 2, “Exploring the JavaScript API for Office: Data Access and Events” ([msdn.microsoft.com/magazine/jj991976](https://msdn.microsoft.com/magazine/jj991976)), looks closely at the important concept of how to get file content and

conducts a thorough review of the Event model. Following this article, Part 4 will focus solely on the third type of app for Office: mail apps.

Throughout this series, we often make reference to the JavaScript API for Office reference documentation. You can find the official documentation, code samples and community resources at the Apps for Office and SharePoint Developer Preview page on MSDN ([dev.office.com](https://dev.office.com)).

## Data Binding in an App for Office

Data binding provides close integration between a specific region of data in the document and the app. The data in the region is bound to a named object in the app so that the app can access the data in the named region, even if the user has selected something else.

Once created, the binding persists even if the region is moved on the page (in Word) or copied to another worksheet (in Excel). For example, a binding to a table persists even if it's renamed by the user.

When the data in the region is changed, the binding raises an event into which the app can hook. From this event, the app can get to the data that has changed and react appropriately.

**Bindings and the “View” of an App** Certainly, the data binding in an app for Office gives an app direct access to a set of data within the Office file, thus making it easier for the app to analyze that data without relying on direct action from the user. Yet data binding does more than just allow for targeted data access—it allows the developer to include the Office file itself as a customizable and integral component of the app.

### This article discusses:

- Scenarios for using data binding in an app for Office
- Using the Binding object and Bindings collection
- The Office Open XML file format
- Using content controls with custom XML parts
- Mapping content controls to an element in the XML
- Using JavaScript with custom XML parts
- Basics of the CustomXmlParts object
- Using CustomXmlParts methods

### Technologies discussed:

JavaScript API for Office

### Code download available at:

[bit.ly/ZaXvaG](https://bit.ly/ZaXvaG)

Many apps for Office present their users with an interface contained solely within the confines of the task pane or content app UI—and there's nothing wrong with that. Yet, in a very simple sense, the data and its presentation within the Office file is itself a “view” of the app. Users interact with their data within the Office file. They enter new data, change existing data and delete unnecessary data within the content of the document. Office applications present a view of the data that users know and understand.

The data-binding capabilities in the JavaScript API for Office allow you to leverage the view of the data that the Office application provides in an app. You, the developer, can develop an “interface” for your app using what's already provided to you in Office. In this way, you can style the view of your app using the out-of-the-box features of the Office application. The data binding then provides the sinews that connect the view of the app to the business logic “model” contained in the JavaScript files.

Of course, the reverse is true, as well. You can use the Office file as your data source, storing the content of the data model. You can then use the app to provide a view of the data. With the flexibility of bindings, you can apply the Model-View-Controller (MVC) pattern to an app and Office file as fits your needs.

**Scenarios for Using Bindings** Without setting a hard limit on the creativity of developers, an app can use bindings in any combination of three generalized ways:

- The app reacts when the user changes the data in the region.
- The app picks up the data in the region, analyzes it, and presents the user with options for modeling or submitting the data.
- The app pushes data from an external data source into the bound region.

Take, for example, a simple stock ticker app inserted into an Excel workbook, where one column in the workbook contains stock symbols and another contains current stock values. With data binding, an app could bind to the column with the stock symbols, picking

up the stock symbols in the column. The app could then subscribe to changes in the price of those stocks via a Web service and parse the results sent from the service. Finally, the app could bind to the price column in the worksheet and update the values in real time.

We'll do just that—create a stock ticker workbook—in the next section as we examine the Binding object.

**Using the Binding Object** The underlying magic of data binding is contained in the Bindings collection and the Binding object.

- The Bindings collection represents all of the bindings created between the Office file and the app for Office. An app doesn't have access to any bindings created by other apps.
- The Binding object represents one named binding between a region in the Office file and the app. It exposes several members for getting, reading, and setting data, and reacting to changes in the bound region.

We'll look more closely at these objects as we build the stock ticker app.

Before we go any further, let's take a quick look at the data. **Figure 1** shows how the view of this app looks. As you can see, we're using fictitious stock symbols for demonstration purposes.

Also, we've already added some “intelligence” to this workbook. The region of data to which we want to bind has been formatted as a table and named “Stocks.” A custom formula has been added to the values in the right-hand column to compare the other values in the table. We also applied conditional formatting to the table to make the icon sets appear in the right-hand column.

It's worth noting that we've added this workbook to our solution in Visual Studio 2012 so that we don't have to re-create our table each time we debug the app. To add the workbook to the solution, right-click the app project in the solution (the first project listed in the Solution Explorer when using the default template), click Add Existing Item and then select your workbook. Then, in the properties for the app project, set the Start Action to your workbook file. When debugging, you'll need to insert the app manually into

your workbook (Insert tab | Apps for Office button).

When it's initialized, the business logic of the app needs to both set the binding and then add an event handler to the event of the binding, `Office.EventType.BindingDataChanged`. **Figure 2** shows the code. Notice that we've encapsulated our code within a self-executing anonymous function stored in the `StockTicker` variable. The name of the table on the spreadsheet, the binding name and the binding are all stored as class fields within the `StockTicker` “class.” The `StockTicker` “class” exposes only a single member: `initializeBinding`.

To establish a binding between the app and the table in the worksheet, we can use one of several

Stock Symbol	Purchase	Shares	Value	Pos / Neg
CONTOSO	\$ 45.73	4	\$ 74.02	▲ \$ 113.16
WINGTIP	\$ 22.10	5	\$ 79.61	▲ \$ 287.55
WOODGROVE	\$ 107.31	3	\$ 53.52	▼ \$(161.37)

**Figure 1 A Table Named “Stocks” in an Excel Workbook with Formulas and Conditional Formatting Applied**



different methods of the Document class in the JavaScript API, including `addFromNamedItemAsync`, `addFromPromptAsync` and `addFromSelectionAsync`. (Note that `addFromPromptAsync` is available only in Excel and the Excel Web App.)

Because we know the name of the region to which we want to bind—it's the table titled "Stocks" on Sheet1—we used the `addFromNamedItemAsync` method to establish the binding. We passed in the name of the table using Excel range notation (`Sheet1!Stocks`). The results of this method call include a reference to the binding itself, allowing us to store a reference to the binding in our binding variable (class field).

In our code, we've passed in the `Office.BindingType.Table` value for the `bindingType` parameter of the method. This specifies that we want to create a "Table" type of binding with our data, although we also could've specified a text or matrix type of binding. Binding to the region as a table provides us with several benefits. For example, if the user adds a new column or row to the table, the scope of the bound region increases, too. That works the other way, as well. The `TableBinding` object, which underlies the binding, exposes properties for adding columns, adding rows and even deleting all of the data in the table.

(See the section titled "Accessing Office File Content from an App for Office" in the second article in this series for details about the text and matrix data types in the JavaScript API for Office.)

**Figure 2 Creating the Binding to the Excel Workbook and Adding a Handler to the Data Changed Event in the Binding**

```
var StockTicker = (function () {

    var tableName = "Sheet1!Stocks",
        bindingName = "Stocks",
        binding;

    // Create the binding to the table on the spreadsheet.
    function initializeBinding() {
        Office.context.document.bindings.addFromNamedItemAsync(
            tableName,
            Office.BindingType.Table,
            { id: bindingName },
            function (results) {
                binding = results.value;
                addBindingsHandler(function () { refreshData(); });
            });
    }

    // Event handler to refresh the table when the
    // data in the table changes.
    var onBindingDataChanged = function (result) {
        refreshData();
    }

    // Add the handler to the BindingDataChanged event of the binding.
    function addBindingsHandler(callback) {
        Office.select("bindings#" + bindingName).addHandlerAsync(
            Office.EventType.BindingDataChanged,
            onBindingDataChanged,
            function () {
                if (callback) { callback(); }
            });
    }

    // Other member methods of this "class" ...

    return {
        initializeBinding: initializeBinding
    };

})();
```

Our code then adds a handler to the `BindingDataChanged` event of the binding. When the data changes in the bound region—that is, when the user changes the data in the region—we want to call a locally defined `refreshData` function to start the process that updates the table. Also, because the table hasn't yet been updated with data from the data source, we'll want to call `refreshData` after the event handler has been added.

You'll note that the `addBindingsHandler` function uses the `Office.select` method to get the binding, although we could've used the `Bindings.getByIdAsync` method instead. The major difference between the two methods is the level of access to the data returned in the results. The `Office.select` method returns a `Binding` object promise to the calling code. If the method succeeds, the `Binding` object returned has only a limited number of members available for use. By selecting the binding using `Office.select`, we can call members from the `Binding` object immediately. This way, we don't have to add a callback to a function that gets the binding in order to add a handler to the binding.

(You might be thinking that we could've just used the local "binding" variable that captures the reference to the binding—and you're right, we could have. We've written this code as it is for demonstration purposes.)

**Figure 3** displays the `refreshData` and `getBindingData` functions. The `refreshData` function simply begins the chain of asynchronous calls that gets the table data from the worksheet by calling `getBindingData`. The `getBindingData` function contains a call to the `Binding.getDataAsync` method and returns the data as a `TableData` object.

In the call to `getDataAsync` shown in **Figure 3**, we could've specified the data type to retrieve (or changed the data type) explicitly by passing in an anonymous object, `{coercionType: Office.CoercionType.Table}`, for the options parameter. Because we haven't specified a data type to retrieve, the `getDataAsync` call returns the binding data in its original data type (a `TableData` object).

The `TableData` object, as we discussed in the second article, provides more structure to the data that we're working with—namely, a header and a `rows` property that we can use to select data from the table. In this example, we just need to get the stock symbols from the first column in the table. As you might recall, the `rows` property stores the data in the table as an array of arrays, where each item in the first array corresponds to a row in the table.

When we work with a binding to a `TableData` object, we can specify a subset of the rows and columns to get from the binding, using the `startRow` and `startColumn` parameters. Both parameters specify zero-based starting points for the data to extract from the table, where the upper-left corner of the table is the point of origin. (Note that you must use the `startRow` and `startColumn` parameters together or else you'll raise an exception.) Because we only need the first column of data from the table, we also pass in the `columnCount` parameter, set to 1.

Once we have that column of data, we push each value into a one-dimensional array. In **Figure 3**, you see that we call a `getStockQuotes` function that accepts the array of stock symbols as an argument. In **Figure 4**, we use the `getStockQuotes` function to retrieve data from a stock quote Web service. (For demonstration purposes, we've left out the code for the Web service.) After we've parsed the results from the Web service, we call the locally defined `removeHandler` method.



## You used to think "Impossible" Your Apps, Any Device

Now you think—game on!! The new tools in 12.2 help you envision and create engaging applications for the Web that can be accessed by mobile users on the go. And, with our Windows 8 XAML and JS tools you will begin to create highly interactive applications that address your customer needs today and build next generation touch enabled solutions for tomorrow.



Download your 30-day trial at  
[www.DevExpress.com](http://www.DevExpress.com)

## DXv2

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



Copyright 1998-2013 Developer Express Inc. All rights reserved. All trademarks are property of their respective owners.

**Figure 3 Getting the Data from the Table Binding and Calling the Web Service**

```
var StockTicker = (function () {

    // Other members of this "class"...

    // Refresh the data displayed in the bound table of the workbook.
    // This function begins a chain of asynchronous calls that
    // updates the bound table.
    function refreshData() {
        getBindingData();
    }

    // Get the stock symbol data from the bound table and
    // then call the stock quote information service.
    function getBindingData() {
        binding.getDataAsync(
            {
                startRow: 0,
                startColumn: 0,
                columnCount: 1
            },
            function (results) {
                var bindingData = results.value,
                    stockSymbols = [];

                for (var i = 0; i < bindingData.rows.length; i++) {
                    stockSymbols.push(bindingData.rows[i][0]);
                }

                getStockQuotes(stockSymbols);
            });
    }

    return {
        // Exposed members of the "class."
    };
})();
```

**Figure 4 Calling the Web Service and Removing the BindingDataChanged Event Handler**

```
var StockTicker = (function () {

    // Other members of this "class"...

    // Call a Web service to get new stock quotes.
    function getStockQuotes(stockSymbols) {

        var stockValues = [];

        // Make a call to the Web service and parse the results.
        // The results are stored in the stockValues variable, which
        // contains an array of arrays that include the stock symbol
        // with the current value.

        removeHandler(function () {
            updateTable(stockValues);
        });

        // Disables the BindingDataChanged event handler
        // while the table is being updated.
        function removeHandler(callback) {

            binding.removeHandlerAsync(
                Office.EventType.BindingDataChanged,
                { handler: onBindingDataChanged },
                function (results) {
                    if (results.status === Office.AsyncResultStatus.Succeeded) {
                        if (callback) { callback(); }
                    }
                });
        }

        return {
            // Exposed members of the "class."
        };
    }
})();
```

The `removeHandler` function calls the `binding.removeHandlerAsync` method, which removes the event handler to the `BindingDataChanged` event. Now, if we had left that handler attached to the event, then the event would be raised when we updated the table. The event handler would then be called again and would update the table, thereby causing an infinite loop. After we've updated the table with the new data, we'll add the event handler back to the event.

(Of course, we also could've created different bindings to separate columns in the table, using the matrix coercion type. Then we could've hooked up events only to the columns that users can edit.)

The `removeHandlerAsync` method takes a parameter, `handler`, which specifies the name of the handler to be removed. It's a best practice to use the handler parameter to remove handlers from binding events.

In **Figure 5**, we're going to update the table with the new stock values by calling the locally defined `updateTable` function.

The `updateTable` function takes the data passed in from the Web service and then writes it back to the bound table. In this example, the `stockValues` parameter contains another array of arrays, where each item in the first array is an array containing a stock symbol and its current price. To set this data back into the bound table, we create a new `TableData` object and insert the stock value data into it.

We need to be careful that the data we set in the `TableData.rows` property matches the shape of the data that we're inserting into the binding. If we blindly set a brand-new `TableData` object into the bound table, we run the risk of losing some of the data in our table—like the formulas, for example. In **Figure 5**, we added the data to the `TableData` object as a single column of data (an array of arrays, where each subarray contains a single item). When we insert this data back into the bound table, we need to insert this updated column of data into the appropriate column.

Here again we use the `startRow` and `startColumn` properties. The `updateTable` function contains a call to `binding.setDataAsync` that pushes the `TableData` back into the table in the worksheet, specifying the `startColumn` and `startRow` parameters. The `startColumn` parameter is set to 3, meaning that the inserted `TableData` object will insert its data starting at the fourth column in the table. In the callback for the `setDataAsync` method, we call the `addBindingsHandler` function again to reapply the event handler to the event.

When the `binding.setDataAsync` method completes successfully, the new table data is pushed into the bound region and immediately displayed. From the user's perspective the experience is seamless. The user types data into a cell in the table, presses Enter and then the Value column of the table automatically updates.

## Custom XML Parts

A particularly noteworthy feature supported by the JavaScript API for Office is the ability to create and manipulate custom XML parts in Word. In order to appreciate the deep potential of the JavaScript API for Office for custom XML parts, some background is helpful. Specifically, you need to understand how the Office Open XML (OOXML or OpenXML) file format, custom XML parts, content controls and XML mapping can be combined to create really powerful solutions—namely, solutions that involve the creation of dynamic Word documents.



# 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](http://www.DynamicPDF.com)



**TRY OUR PDF SOLUTIONS FREE TODAY!**

[www.DynamicPDF.com/eval](http://www.DynamicPDF.com/eval) or call 800.631.5006 | +1 410.772.8620

**ceTe software**

**Figure 5 Getting the Data from the Table Binding and Calling the Web Service**

```
var StockTicker = (function () {

    // Other members of this "class"...

    // Update the TableData object referenced by the binding
    // and then update the data in the table on the worksheet.
    function updateTable(stockValues) {

        var stockData = new Office.TableData(),
            newValues = [];

        for (var i = 0; i < stockValues.length; i++) {
            var stockSymbol = stockValues[i],
                newValue = [stockSymbol[1]];

            newValues.push(newValue);
        }

        stockData.rows = newValues;

        binding.setDataAsync(
            stockData,
            {
                coercionType: Office.CoercionType.Table,
                startColumn: 3,
                startRow: 0
            },
            function (results) {
                if (results.status === Office.AsyncResultStatus.Succeeded) {
                    addBindingsHandler();
                }
            }
        );
    }

    return {
        // Exposed members of the "class."
    };
})();
```

**OOXML Formats** Office 2007 introduced the new OOXML file format for Office documents, now the default file format for Office 2010 and Office 2013. (You can tell which Office documents are in the OOXML file format because the extensions for those documents are now four-letter extensions, many of which end in “x,” for example “.docx” for a Word document, “.xlsx” for an Excel spreadsheet or “.pptx” for a PowerPoint document.)

Office documents in the OOXML format are essentially .zip files. Each .zip file contains a collection of XML files, called “parts,” which together make up the Office document. If you rename an Office document, such as a Word .docx document, to .zip, and then examine the files inside, you can see that the document is really just a collection of separate XML files, organized into folders, inside a .zip package, as shown in **Figure 6**.

**Custom XML Parts Basics** While there are standard XML parts that the Office applications always create for each new Office document in the OOXML format (for example, there’s a built-in XML part that describes core document properties), the interesting thing is that you can also add your own “custom XML” parts to a Word document, Excel workbook or PowerPoint presentation. The custom XML

parts are added to the collection of XML files inside the .zip package that forms the Office document. A custom XML part is stored within the file structure of the document but isn’t displayed to the end user. This allows you to insert business data that travels with a specific instance of an Office document that’s hidden inside the file structure. You can then work with that custom XML in your app, and that’s exactly what the JavaScript API for Office supports.

**Content Controls** Along with the OOOXML format and its file structure that allows for the inclusion of custom XML into a document, Word 2007 added content controls, a feature that richly complements custom XML parts.

Content controls are a way to define fixed regions in a Word document that hold certain kinds of data, such as plain text, rich text, pictures, dates and even repeating data. The key aspect of content controls that complements custom XML parts is data binding using XML mapping.

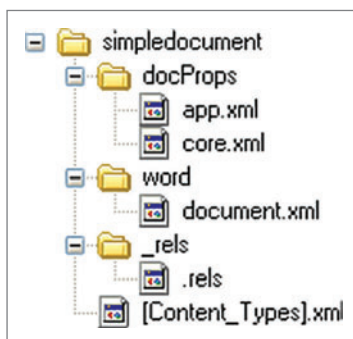
**XML Mapping** A content control can be bound or “mapped” to an element in the XML in an XML part that’s contained in the document. For example, a business could inject business data from a back-end system as a custom XML part into a Word document that has content controls mapped to the custom XML part. The content controls are bound to specific nodes in the custom XML part so when the end user opens the document, the XML-mapped content controls are automatically populated with data from the custom XML part. Or, reversing the scenario, a business could use the same Word document with mapped content controls but have the end user enter data into the content controls. When the document is saved, the data in the mapped content controls is saved back to the XML file. An application could then scrape the data from the custom XML part in the saved document and push it into a back-end system. The JavaScript API for Office provides rich support for developing applications exactly like those just described.

**Using the JavaScript API for Office to Work with Custom XML Parts** The best way to walk through some of the more significant parts of the custom XML parts API in the apps for Office JavaScript Object Model is through an example. In this section, we use the “invoice manager” sample ([bit.ly/YRdlwt](http://bit.ly/YRdlwt)) from the Samples area of the apps for Office and SharePoint developer portal so that you can follow along. The invoice manager sample is an example of a dynamic document scenario where a business wants to generate documents that draw data from a back-end system to produce

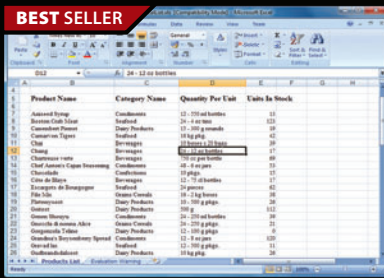
invoices. In this case, the data is a customer’s name and shipping address, and an associated list of the customer’s purchases.

The sample includes a template document used to create new invoices. The template document has a layout with the customer name, address and a table of customer purchases. The customer name, address and purchases sections of the document are each content controls. Each content control is mapped to a node in the schema that was created to hold customer invoice data, as shown in **Figure 7**.

The UI for the invoice manager sample app is straightforward, as shown in **Figure 8**.

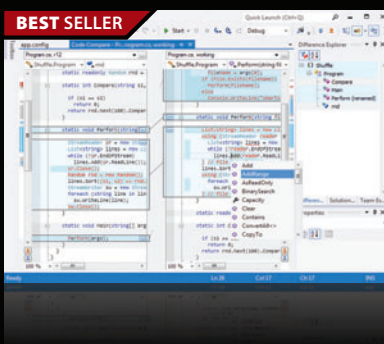


**Figure 6 File Structure of an Office Open XML Format Document**

**Aspose.Total for .NET** from \$2,449.02

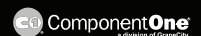
Every Aspose .NET component in one package.

- 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

**Code Compare Pro** from \$48.95

An advanced visual file comparison tool with Visual Studio integration.

- Code oriented comparison, including syntax highlighting, unique structure and lexical comparison algorithms, for the most popular programming languages
- Smooth Visual Studio integration to develop and merge within one environment in the context of current solution, using native IDE editors
- Three-way file merge, folder comparison and synchronization

**ComponentOne Studio Enterprise** from \$1,315.60

.NET Tools for the Smart Developer: Windows, Web, and XAML.

- Hundreds of UI controls for all .NET platforms including grids, charts, reports and schedulers
- Supports Visual Studio 2012 and Windows 8
- Now includes Windows 8 Studios for WinRT XAML and WinJS
- New Cosmopolitan (Windows 8 UI) theme provides a modern look and feel
- Royalty-free deployment and distribution

**Help & Manual Professional** from \$583.10

Easily create documentation for Windows, the Web and iPad.

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



**Packing Slip**

**Shipping Information**

Ship From:  
Litware, Inc.  
1234 NE Campus Pkwy  
Seattle, WA 98888

Ship To:

Click here to enter text.

Customer Name Click here to enter text.

Address Line 1 Click here to enter text.

text Address Line 1

Address Line 2 Click here to enter text.

text Address Line 2

Click here to enter text.

Click here to enter text.

Figure 7 Content Controls on the Document Surface Mapped to a Custom XML Part

The end user chooses an invoice number from the dropdown box in the app UI and the customer data associated with the invoice number is shown in the body of the app, as shown in Figure 9.

When the user chooses the Populate button, the app pushes the displayed data as a custom XML part into the document. Because the content controls are mapped to nodes in the custom XML part, as soon as the custom XML part is pushed into the document the content controls immediately show the data for each XML node to which they're mapped. You can replace the custom XML part on the fly (as we did here), but as long as the part conforms to the schema to which the content controls are mapped, the content controls will show the mapped data. Figure 10 shows the invoice manager Packing Slip form when the content controls are mapped to a custom XML part in the document.

## The CustomXmlParts Object

**CustomXmlParts.addAsync** The first step in working with custom XML parts is learning how to add them to a document using the JavaScript API for Office. The only way to do this is by using the customXmlParts.addAsync method. As its name suggests, the customXmlParts.addAsync method adds a custom XML part asynchronously and has the following signature:

```
Office.context.document.customXmlParts.addAsync(xml [, options], callback);
```

Note that the required first parameter for the function is a string of XML. This is the XML for the custom XML part. As we mentioned earlier, the invoice manager uses custom XML parts that are mapped to the content controls on the document surface, but first it has to get customer data to insert as custom XML. In the InvoiceManager.js file, which holds the logic for the entire app, the app simulates getting customer data from a back-end system using the user-defined

function setupMyOrders. This function creates an array of three objects that represent three customer orders. You can, of course, imagine any number of ways a business might store and get a customer's purchase history—for example, a SQL database—but for simplicity's sake, the app creates three "hardwired" customer orders right within the app.

Once the orders objects have been created, the data they represent must be rendered in XML so that they can be used in the call to customXmlParts.addAsync. That's what happens in the initializeOrders function, which also sets up the app UI and wires up event handlers to the controls on the UI. The important piece to note is in the jQuery code that wires up an event handler for

the Populate button click event, as shown in Figure 11.

Essentially, the anonymous function that acts as the event handler for the Populate button click event converts an order object (which was created with the setupMyOrders function) into a string of XML and then calls the customXmlParts.addAsync method and passes the string of XML that contains the order info as the required first parameter.

The other parameter for customXmlParts.addAsync is a callback. Of course, this can be a reference to a method defined elsewhere in your code, or it can be an anonymous function. The invoice manager sample uses an inline anonymous function:

```
_document.customXmlParts.addAsync(xml, function (result) { });
```

As is the case for all callbacks in the JavaScript API for Office, an AsyncResult object is passed in as the only argument for the callback. For customXmlParts.addAsync and all the customXmlParts functions, you can use the AsyncResult object to:

- get a reference to the newly created custom XML part using the AsyncResult.value property
- get the result of the request using the AsyncResult.status property
- get information about an error (if one occurred) using the AsyncResult.error property
- get your own state data (if you included any in the call to customXmlParts.addAsync) using the AsyncResult.asyncContext property

For that last item, remember that the other parameter in the customXmlParts.addAsync method was an optional options object:

```
Office.context.document.customXmlParts.addAsync(xml [, options], callback);
```

The options object is provided as a way for you to pass your own user-defined object into the call for your callback.

**InvoiceManager**

Order ID: Help me!

Date

Customer Name

Address

Items

Populate

Figure 8 The UI for the Invoice Manager Sample App

**InvoiceManager**

Order ID: 918291

Date  
5/24/2011

Customer Name  
Lisa Andrews

Address  
678 Elm St.  
Redwood City, CA 12202

Items  
Diary of a Wussy Kid: Cabin Fever [\$20.00]  
1Q95 [\$16.05]  
A Wild Goose Chase: A Novel [\$12.35]  
Kafka in the Woods [\$7.86]  
My Isadora [\$16.05]  
Sputnik Darling [\$11.20]  
Swedish Wood [\$11.99]

Populate

Figure 9 The Invoice Manager UI Populated with Data from a Custom XML Part

Visual Studio **LIVE!**  
EXPERT SOLUTIONS FOR .NET DEVELOPERS

YOUR BACKSTAGE PASS TO THE MICROSOFT PLATFORM

# Sweet 127.0.0.1 Chicago!

4 Days of 60+ Sessions and Workshops

**VISUAL STUDIO LIVE! CHICAGO | MAY 13-16, 2013**  
HILTON CHICAGO | [vslive.com/chicago](http://vslive.com/chicago)



Flip over for more details



**PRACTICAL & UNBIASED  
EDUCATION FOR DEVELOPERS:**

- ASP.NET
- Azure / Cloud Computing
- Cross-Platform Mobile
- Data Management
- HTML5 / JavaScript
- Windows 8 / WinRT
- WPF / Silverlight
- Visual Studio 2012 / .NET 4.5

**REGISTER TODAY  
AND SAVE \$200**

USE PROMO CODE CHTIP1

[vslive.com/chicago](http://vslive.com/chicago)

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



GOLD SPONSOR



SUPPORTED BY

**Microsoft**

 **Visual Studio**

**msdn**  
magazine

PRODUCED BY

 **1105 MEDIA**



## Packing Slip

**Shipping Information**

Ship From:  
Litware, Inc.  
1234 NE Campus Pkwy  
Seattle, WA 98888

Ship To:  

Lisa

Customer Name

Andrews

Address Line 1

678 Elm St.

Address Line 1

Address Line 2

Redwood City, CA

12202

Address Line 2

Diary of a Wussy Kid: Cabin Fever	\$20.00
1Q95	\$16.05
A Wild Goose Chase: A Novel	\$12.35
Kafka in the Woods	\$7.86
My Isadora	\$16.05
Sputnik Darling	\$11.20
Swedish Wood	\$11.99

Figure 10 Content Controls Mapped to Nodes in a Custom XML Part Showing Bound Data

As you can see in the invoice manager sample, the anonymous function in the call to `customXmlParts.addAsync` does nothing, but in a production environment you'd probably want to do error checking to handle an instance gracefully if for some reason the custom XML part isn't successfully added.

**CustomXmlParts.getByNamespaceAsync** Another key part of the JavaScript API for Office for working with custom XML parts that's demonstrated in the invoice manager sample is the use of the `customXmlParts.getByNamespaceAsync` method, which you can

Figure 11 Wiring Up an Event Handler for the Populate Button Click Event

```
$("#populate").click(function () {
    var selectedOrderID = parseInt($("#orders option:selected").val());
    _document.customXmlParts.getByNamespaceAsync("", function (result) {
        if (result.value.length > 0) {
            for (var i = 0; i < result.value.length; i++) {
                result.value[i].deleteAsync(function () {
                });
            }
        }
    });
    var xml = $.json2xml(findOrder(myOrders, selectedOrderID));
    _document.customXmlParts.addAsync(xml, function (result) { });
});
```

Figure 12 Using the Method CustomXmlParts.getByNamespaceAsync

```
$("#populate").click(function () {
    var selectedOrderID = parseInt($("#orders option:selected").val());
    _document.customXmlParts.getByNamespaceAsync("", function (result) {
        if (result.value.length > 0) {
            for (var i = 0; i < result.value.length; i++) {
                result.value[i].deleteAsync(function () {
                });
            }
        }
    });
    var xml = $.json2xml(findOrder(myOrders, selectedOrderID));
    _document.customXmlParts.addAsync(xml, function (result) { });
});
var selOrder = $("#orders option:selected");
popOrder(selOrder.val());
```

see in the click event handler code for the Populate button. The signature for `customXmlParts.getByNamespaceAsync` is:

```
Office.context.document.customXmlParts.getByNamespaceAsync(
    ns [, options], callback);
```

The required first parameter, *ns*, is a string that specifies the namespace of the custom XML parts that you want to get. So `customXmlParts.getByNamespaceAsync` returns an array of custom XML parts in the document that have the namespace you specified. Because the custom XML parts created in the invoice manager sample don't use namespaces, the call to `customXmlParts.getByNamespaceAsync` passes an empty string as the argument for the namespace parameter, as shown in Figure 12.

Like all asynchronous functions in the API, `customXmlParts.getByNamespaceAsync` has optional options and callback parameters.

**CustomXmlParts.getByIdAsync** The last programmatic way to get a custom XML part in a document is the `customXmlParts.getByIdAsync`. The signature is:

```
Office.context.document.customXmlParts.getByIdAsync(id [, options], callback);
```

This function gets a single custom XML part using the GUID of the part. You find the GUID for the custom XML part in the `itemProps.xml` file inside the document package. You can also get the GUID using the `id` property of the `customXmlPart`. A key thing to note here is that the string for the GUID must contain the curly braces ("{}") around the GUID.

The invoice manager sample doesn't use the `customXmlParts.getByIdAsync` function, but the following code demonstrates it clearly enough:

```
function showXMLPartById() {
    Office.context.document.customXmlParts.getByIdAsync(
        "{3BC85265-09D6-4205-B665-8EB239A8B9A1}", function (result) {
        var xmlPart = result.value;
        write(xmlPart.id);
    });
}
// Function that writes to a div with id='message' on the page.
function write(message){
    document.getElementById('message').innerText += message;
}
```

In addition to the `id` parameter, like `customXmlParts.addAsync` and `customXmlParts.getByNamespaceAsync`, the `customXmlParts.getByIdAsync` method also has the optional parameter, *options*, and the required parameter callback, and they're used just as in the other functions.

**The CustomXmlPart Object** The `customXmlPart` object represents a single custom XML part. Once you get a reference to a `customXmlPart` using the methods from the `customXmlParts` object, you have several properties available, as shown in Figure 13.

`CustomXmlPart` also has events associated with it, which are shown in Figure 14.

But for the purposes of this article, we want to focus on a few key methods of the `customXmlPart` object that will often be used by developers. These are shown in Figure 15.

**CustomXMLPart.addHandlerAsync** The `customXmlPart.addHandlerAsync` method is key to wiring up event handlers that

Figure 13 CustomXmlPart Properties

Name	Description
builtIn	Gets a value that indicates whether the customXmlPart is built in.
id	Gets the GUID of the customXmlPart.
namespaceManager	Gets the set of namespace prefix mappings (customXmlPrefixMappings) used against the current customXmlPart.

Figure 14 CustomXmlPart Events

Name	Description
nodeDeleted	Occurs when a node is deleted.
nodeInserted	Occurs when a node is inserted.
nodeReplaced	Occurs when a node is replaced.

Figure 15 CustomXmlPart Methods

Name	Description
addHandlerAsync	Asynchronously adds an event handler for a customXmlPart object event.
deleteAsync	Asynchronously deletes this custom XML part from the collection.
getNodesAsync	Asynchronously gets any customXmlNodes in this custom XML part that match the specified XPath.
getXmlAsync	Asynchronously gets the XML inside this custom XML part.

respond to changes to the custom XML part. The signature for the customXmlPart.addHandlerAsync method is as follows:

```
customXmlPart.addHandlerAsync(eventType, handler [, options], callback);
```

Note that the first required parameter is an Office.EventType enumeration, which specifies what kind of event in the apps for Office object model you want to handle. The next required parameter is the handler for the event. The important thing here is that when the handler is invoked, the JavaScript API for Office will pass in an event arguments parameter specific to the kind of event being handled (NodeDeletedEventArgs, NodeInsertedEventArgs or NodeReplacedEventArgs). Then, as in all asynchronous functions in the API, you have, optionally, options and callback parameters.

Consider the scenario where a document is being used like a data-entry form. The user inputs data into the form and then the

Figure 16 Wiring Up an Event Handler for the CustomXmlPart.NodeInserted Event

```
function addNodeInsertedEvent() {
    Office.context.document.customXmlParts.getByIdAsync(
        "{3BC85265-09D6-4205-B665-8EB239A8B9A1}", function (result) {
        var xmlPart = result.value;
        xmlPart.addHandlerAsync(Office.EventType.NodeInserted,
            function (eventArgs) {
                write("A node has been inserted.");
            });
    });
}
// Function that writes to a div with id='message' on the page.
function write(message){
    document.getElementById('message').innerText += message;
}
```

form is scraped for the data. The form contains a Repeating Section content control so that each time the user enters a repeated item, a new node is added to the underlying custom XML part. Every time a node is added, or inserted, the NodeInserted event is fired and you can react to the event (and all customXmlPart events) using customXmlPart.addHandlerAsync.

Figure 16 shows how you could respond to the NodeInserted event.

**CustomXMLPart.deleteAsync** Of course, along with knowing how to add a custom XML part, it's important to know how to delete one. The customXmlPart.deleteAsync method provides that functionality. CustomXmlPart.deleteAsync is an asynchronous function with the following signature:

```
customXmlPart.deleteAsync([options], callback);
```

Going back to the invoice manager sample, you can see a demonstration of customXMLPart.deleteAsync:

```
$("#populate").click(function () {
    var selectedOrderID = parseInt($("#orders option:selected").val());
    _document.customXmlParts.getByNamespaceAsync("", function (result) {
        if (result.value.length > 0) {
            for (var i = 0; i < result.value.length; i++) {
                result.value[i].deleteAsync(function () {
                });
            }
        }
    });
});
```

Within the click event handler for the Populate button, the program logic checks to see if any custom XML parts with "blank" namespaces exist. If they do, it deletes each one using the customXmlPart.deleteAsync method.

There's much more to working with custom XML parts, but what we've walked through in this article should be enough to give you a sense of the rich support the JavaScript API for Office provides for custom XML parts.

## Next Up: Mail Apps

In this third article of the series, we reviewed some advanced techniques for working with data in apps for Office. We showed how to add additional intelligence to a table in Excel by using data bindings. We also explored how to leverage custom XML parts in an app for Word to facilitate automated document creation.

In the next and final article in this series, we'll examine the JavaScript API for Office as it applies to mail apps. Mail apps represent a unique set of capabilities within the JavaScript API for Office, allowing app developers and Exchange administrators to build powerful tools for working with e-mail items. ■

**STEPHEN OLIVER** is a programming writer in the Office Division and a Microsoft Certified Professional Developer (SharePoint 2010). He writes the developer documentation for the Excel Services and Word Automation Services, along with PowerPoint Automation Services developer documentation. He helped curate and design the Excel Mashup site at ExcelMashup.com.

**ERIC SCHMIDT** is a programming writer in the Office Division. He has created several code samples for apps for Office, including the popular Persist custom settings code sample. In addition, he has written articles and created videos about other products and technologies within Office programmability.

**THANKS** to the following technical experts for reviewing this article: Mark Brewster (Microsoft), Shilpa Kothari (Microsoft) and Juan Balmori Labra (Microsoft)

# 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 ...



## 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](http://www.nsoftware.com)



# Building and Validating Windows Store Apps with Team Foundation Service

Thomas Lebrun

**Available in final release** since October 2012, Team Foundation Service, the cloud version of Visual Studio Team Foundation Server (TFS), offers a number of features that can help you deliver quality software. And it's no great leap to imagine the software you're thinking about developing will likely target Windows 8.

You might be wondering whether you can mix the power of the two products—use Team Foundation Service to build Windows Store applications. Unfortunately, this isn't possible out of the box, due to some limitations I'll cover later in this article. However, I'll also show you the steps needed to bypass this issue and help you validate Windows Store apps during the build process.

First, let's take a quick look at Team Foundation Service.

## Overview of Team Foundation Service

As a cloud-based service, Team Foundation Service allows developers to access the features offered by TFS without the hassle of

having to install and manage it. Just sign up (for free!) at [bit.ly/ZusqUY](http://bit.ly/ZusqUY) and you're ready to go.

Among the services offered by the product, you'll likely focus on three major features to develop and deliver high-quality software:

**Source Code Management** You can use the source control feature with your current tools and preferred language—virtually any kind of file (C#, C++, HTML, PHP, Java and more) can be handled by the source controller. If you use an IDE such as Visual Studio or Eclipse, you can continue to use it to develop your applications and to check your files into the source control.

The source controller architecture provides a local workspace that stores a local copy of the source code. In disconnected mode, all modifications are performed in this workspace. When you reconnect, you simply check in your code to push it to the server, keeping the full history of all versions so you can track and roll back changes.

**Collaboration** Team Foundation Service lets developers work better together, primarily via a tool called the task board. The task board is accessible from any modern browser that lets you create custom dashboards. You use the task board to manage information about work items, build status, tests results and so forth, as shown in **Figure 1**.

**Build Service** Still in preview at the time of this writing, the Build Service is based on Team Build, which is part of TFS 2010, and provides automated build in the cloud.

All the standard features of Team Build can be used by the Build Service, including continuous integration, nightly builds and gated check-in builds. Moreover, the build template it uses is fully customizable. There's even a "ready to go" template for doing continuous

The Team Foundation Service Build Service is still in preview. All information is subject to change.

### This article discusses:

- Major features of Team Foundation Service
- Implementing a workaround for building Windows Store apps with Team Foundation Service
- Validating Windows Store apps and customizing the build report

### Technologies discussed:

Team Foundation Service, Windows 8, Visual Studio 2012, Visual Studio Team Foundation Server 2012

deployment on Windows Azure (you can check in code in the source control and see its updates on Windows Azure Web sites).

The Build Service is hosted on a build server deployed with Windows Server 2008 R2, Team Build, Visual Studio 2010 or later, and more (see the full list of required software and options at the bottom of this page: [bit.ly/12Sf99Z](http://bit.ly/12Sf99Z)). The default configuration is good for most applications—except Windows Store apps. As you can see in **Figure 2**, Windows Store apps need to be built on Windows 8 (or Windows Server 2012) and these are not installed on the build server.

So, as noted earlier, building Windows Store apps with Team Foundation Service isn't possible out of the box. But, as you'll see, there's a workaround. In essence, the workaround consists of installing a Windows 8 computer that will become a new build agent, dedicated to Windows Store apps, for Team Foundation Service. I'll show you how to implement it.

## Building Windows Store Apps with Team Foundation Service

Here's a look at the steps needed to build Windows Store apps using Team Foundation Service.

**Installing the Build Service** First, you need a machine running Windows 8. This can be either a physical or a virtual machine (VM); it's not important as long as the machine is accessible from the Internet. Next, install TFS 2012 on that machine. Note that installing TFS doesn't mean it'll be configured and ready to use. The only reason you're installing it is to be able to configure the Build Service. You don't need to get a Team Foundation Application Server, as you have the one from Team Foundation Service.

Once the Build Service is installed, you can configure it using a dedicated team project collection. In this case, because you won't be setting up other TFS components and you do want to use the Team Foundation Service, specify the team projects collection available with your Team Foundation Service account and finish the configuration, as shown in **Figure 3**.

**Configuring the Build Service** For the next part, you need to understand some basics about TFS and its build architecture.

Each TFS has a set of dedicated build controllers. A build controller is the endpoint that will receive the build request and execute it using dedicated build agents. The build agent does the most important

work of the build: It gets files from source control, compiles the code, executes unit tests and so on.

Team Foundation Service comes with a dedicated build controller—the hosted build controller—so you might think you just have to create a new agent to run with this controller. Unfortunately, you can't attach an on-premises build agent to a hosted build controller. You need to either choose another build controller or create a new build controller.

To keep it simple, let's create a new one, as shown in **Figure 4**.

Once the controller is up and running, the next step is to set up a new agent dedicated to the building of Windows Store apps. Creating a new build agent is as simple as clicking on the New Agent link and then filling in the fields. In a real production environment, you might have more agents for your build controller, so to be sure that Windows Store apps will be built only by agents running on Windows 8, add a dedicated tag as shown in **Figure 5**. This isn't required, but when you create the build definition later, you'll be able to specify this tag to ensure that only this agent will be used for the build process.

Before moving to the next part, you need to go to the Build Service properties and set it to run interactively. This step isn't mandatory if you just want to build Windows Store apps. But if you want to validate your applications, Team Foundation Service

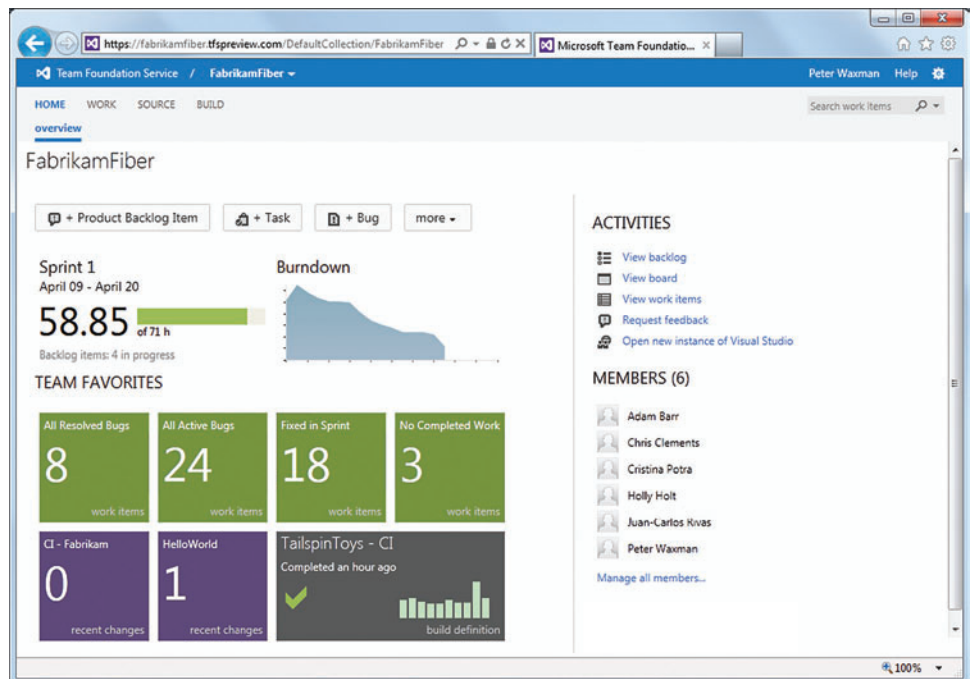


Figure 1 Sample Dashboard for Team Foundation Service



Figure 2 Building Windows Store Apps on Team Foundation Service Is Not Possible out of the Box

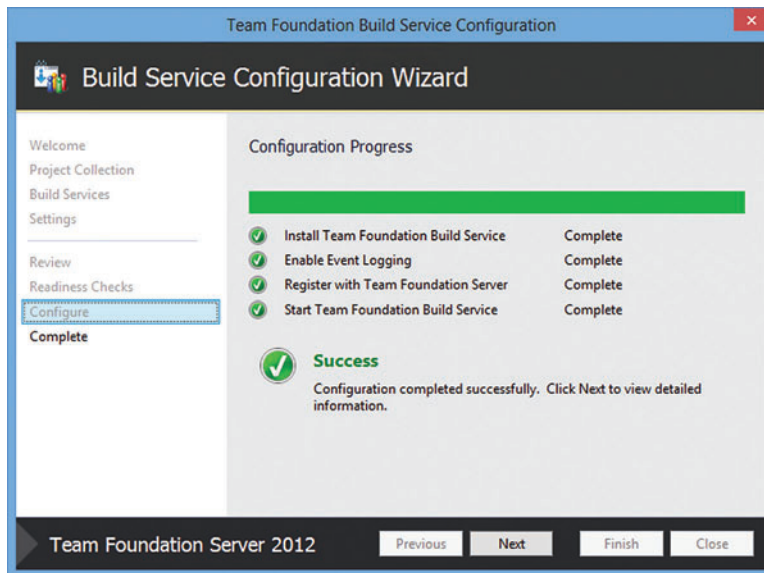


Figure 3 Installing the Team Foundation Build Service in a Dedicated Team Projects Collection

and the Build Service will need to install and launch them on the build machine, and this can't be done if the Build Service isn't configured to run interactively.

On the Builds page in Visual Studio Team Explorer, click on Actions, then on Manage Build Controllers to open a window showing a list of all the build controllers (with their dedicated agents) installed. If the configuration was successful, you should see your new controller and agent.

**Preparing the Build Agent to Run Unit Tests** If the computer that hosts the build agent will be used to perform unit tests, there are two other steps that need to be performed. First, a Windows 8 developer license must be installed on the computer. Developer licenses are free, they need to be renewed every 30 days (or 90 days, if you have a Windows Store account), and you can get as many as you need if you already have a Microsoft account. There are two ways to acquire a developer license. On your build machine, you can simply create a Windows Store app, which opens a dialog box from which you can get a valid license. If you don't want to create a fake application on the build machine, you can run the following Windows PowerShell command to get the same dialog box:

```
C:\PS> Show-WindowsDeveloperLicenseRegistration
```

After you get the developer license, you need to generate and install a unit test certificate (from the code project that contains the unit tests you want to run) on the build agent. For this step, generate an application package on the developer machine. In Visual Studio, click on Store | Create App Package. This creates a folder containing the Windows Store app (in a file with the extension .appx) and its certificate.

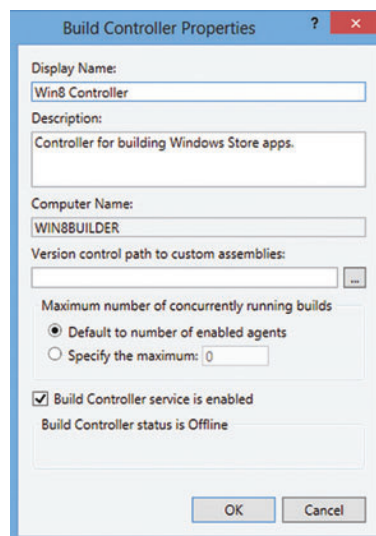


Figure 4 Creating a New Build Controller

To install the certificate on the build machine, open a command prompt as an administrator and enter the following command:

```
certutil -addstore root certificate_file
```

Note that *certificate\_file* is the path to the certification file.

**Building Windows Store Apps** Once the controller and agent are running, building Windows Store apps is the same as building other kinds of applications. You just need to set up a new build definition and specify that you want to use the new build controller you just set up. To be sure that the build process will use the build agent running on Windows 8, in the Process tab of the build definition, select the tag you indicated when you created the build agent (see Figure 6).

Once this is done, queuing a new build using the build definition you just created launches it and, thanks to the specified tag, you can be sure the build is performed using the right agent, so it won't fail.

As you can see, building a Windows Store app using Team Foundation Service is pretty easy and extremely powerful, and you can fully customize the build process.

However, there's still a problem. Even if the build succeeds, it doesn't mean the application will run correctly or even that it will pass all the basics steps for validation. Next, I'll explain how you can validate the application and how to indicate to users (through the build report) whether the validation passed or failed.

## Validating Windows Store Apps During Team Build

As you probably know, in order to be published to the Windows Store, an app must be certified. That is, it must pass the required validation steps. You can validate your applications during the build process. This can be performed easily by simply adding a post-build event, which will launch the Windows App Certification Kit (ACK) to validate the application. But, out of the box, the validation doesn't notify users of the results. I'll show you how to extend the build process to include this step.

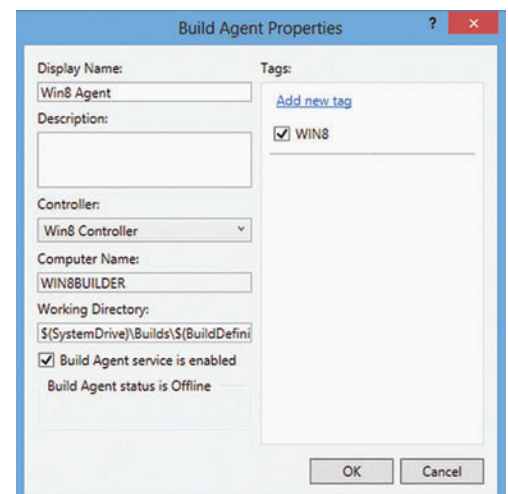


Figure 5 Creating a New Build Agent



# NOSQL IS JUST THE BEGINNING

## INTRODUCING FATDB

The only NoSQL database built on and for .NET that includes a distributed file management system, work queue, and high-speed cache. For more information or to download FatDB visit **[FATCLOUD.COM/DOWNLOAD](http://FATCLOUD.COM/DOWNLOAD)**.



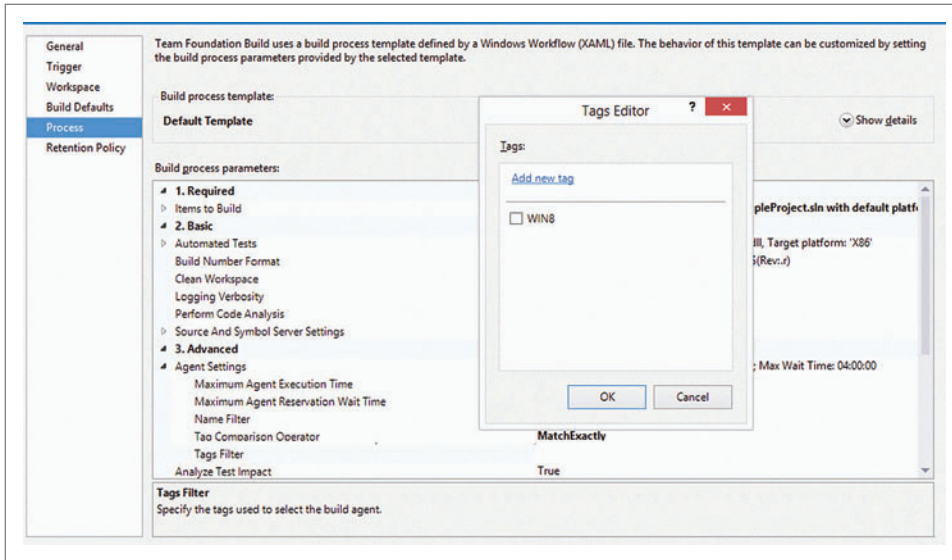


Figure 6 Creating the Build Process Using the Specified Tag

To integrate ACK execution during the build process, you just need to modify your project file to add the following PostPackageEvent:

```
<Target Name="PostPackageEvent" AfterTargets="_GenerateAppxPackage">
  <Exec Command="&quot;$(TargetPlatformSdkPath)\App Certification Kit\
  appcert.exe&quot; reset"/>
  <Exec Command="&quot;$(TargetPlatformSdkPath)\App Certification
  Kit\appcert.exe&quot; test -apptype windowsstoreapp -AppxPackagePath
  &quot;$(FinalAppxPackage)&quot; -reportoutputpath &quot;$(outdir)\
  ValidationResult.xml&quot;"/>
  <Exec Command="copy &quot;$(userprofile)\appdata\Local\Microsoft\
  appcertkit\ValidationResult.htm&quot; &quot;$(outdir)\ValidationResult.
  htm&quot;"/>
</Target>
```

Figure 7 CheckWackResultsActivity

```
[BuildActivity(HostEnvironmentOption.All)]
public sealed class CheckWackResultsActivity : CodeActivity<bool>
{
    [RequiredArgument]
    public InArgument<string> DropLocation { get; set; }

    [RequiredArgument]
    public InArgument<string> WackResultsFilename { get; set; }

    [RequiredArgument]
    public InArgument<string> WackReportFilename { get; set; }

    public OutArgument<string> WackReportFilePath { get; set; }

    // If your activity returns a value, derive from CodeActivity<TResult>
    // and return the value from the Execute method.
    protected override bool Execute(CodeActivityContext context)
    {
        string dropLocation = context.GetValue(this.DropLocation);
        string wackResultsFilename =
            context.GetValue(this.WackResultsFilename);
        string wackReportFilename = context.GetValue(this.WackReportFilename);

        var dropLocationFiles = Directory.GetFiles(dropLocation, "*",
            SearchOption.AllDirectories);
        if (dropLocationFiles.Any())
        {
            var resultFile = dropLocationFiles.FirstOrDefault(
                f => Path.GetFileName(f).ToLowerInvariant() ==
                wackResultsFilename.ToLowerInvariant());
            if (!string.IsNullOrEmpty(resultFile))
            {
                var xDocument = XDocument.Load(resultFile);
                var reportElement = xDocument.Element("REPORT");
```

```
if (reportElement != null)
{
    var resultAttribute = reportElement.Attribute("OVERALL_RESULT");
    if (resultAttribute != null)
    {
        context.SetValue(this.WackReportFilePath,
            Path.GetDirectoryName(resultFile));

        var validationResult = resultAttribute.Value;
        // Fail or Pass
        if (validationResult.ToLowerInvariant() == "fail")
        {
            return false;
        }

        return true;
    }
}

throw new InvalidOperationException(
    "Unable to find the Windows App Certification Kit results file!");
}
else
{
    throw new InvalidOperationException(
        "There are no files in the drop location!");
}

throw new InvalidOperationException(
    "Unknown error while checking the content of the Windows App
    Certification Kit results file!");
}
```

When executed, the code will create the file ValidationResult.html, which contains the results of the validation performed by ACK. If you're connected to the build server when the build is executed, you'll see that the application is launched to be validated by ACK. This is normal; the app is installed, validated and then removed automatically when the test is finished. Remember that you configured the Build Service to run interactively, which is what allows the application to be installed and executed. If you hadn't done this, an error would have occurred during the build.

The build process itself isn't affected by the results of the validation, so users need to be able to

check the test results to know whether the application has validation errors. Fortunately, you can enhance the build report to let users know if the validation encountered any errors. Let's see how to customize the build report to integrate the validation results of the ACK tool.

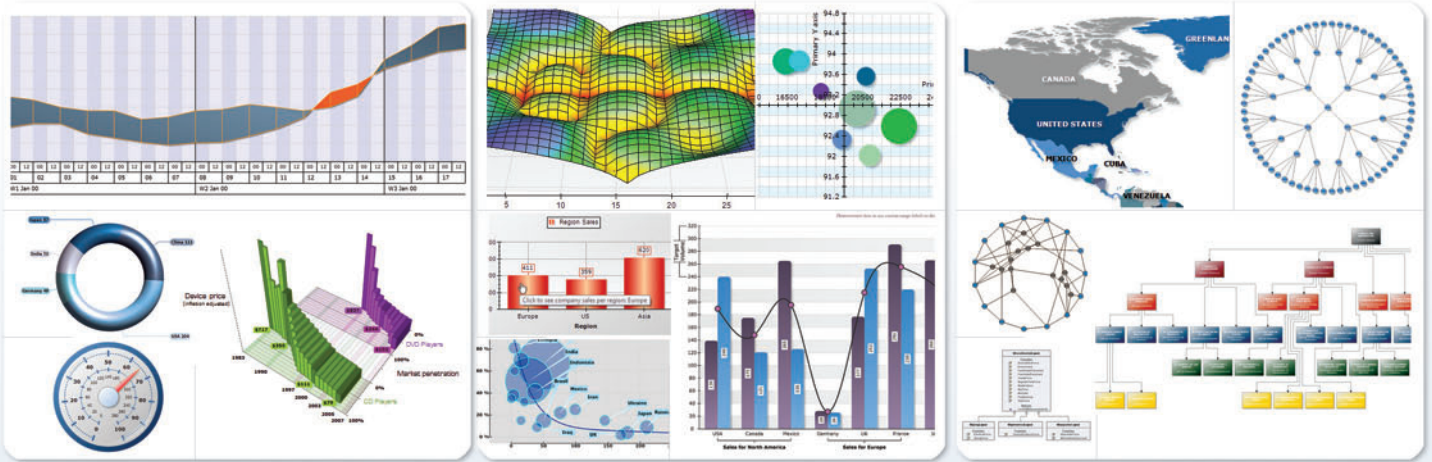
**Customizing the Build Report** ACK creates both an HTML and an XML file and saves them in the folder you choose. You can use this XML file to create a custom workflow activity that will modify the build report to notify users of the validation results.

The code to create this activity is quite simple. It finds the XML file (which contains the validation results), reads the file to find the value

# Nevron Data Visualization

The leading data visualization components for a wide range of .NET platforms.

14+ years of refinement, complete feature sets, highly customizable design and great support.



## Nevron Vision for .NET

Incorporates components that help you create enterprise grade digital dashboards, scorecards, diagrams, maps, MMI interfaces and much more.



## Nevron Vision for SharePoint

The leading data visualization web parts for SharePoint 2007 and 2010. Helps you convert your SharePoint pages into interactive dashboards and reports.



## Nevron Vision for SSRS

The leading data visualization report items for SSRS 2005, 2008 and 2012. Helps you deliver deeper data insights with more engaging looks.



Nevron components integrate seamlessly in Web and Desktop .NET applications, SQL Server Reporting Services 2005/2008/2012 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.

Make sure that your data is making the visual statement it deserves by downloading your free evaluation copy from [www.nevron.com](http://www.nevron.com) today.



Figure 8 The Modified Build Process

```
<Sequence DisplayName="Windows 8" sap2010:WorkflowViewState.IdRef="Sequence_4">
  <Sequence.Variables>
    <Variable x:TypeArguments="x:Boolean" Name="WackToolRanSuccessfully" />
    <Variable x:TypeArguments="x:String" Name="WackReportFilePath" />
  </Sequence.Variables>
  <c:CheckWackResultsActivity DropLocation="[DropLocation]"
    sap2010:WorkflowViewState.IdRef="CheckWackResultsActivity_3"
    Result="[WackToolRanSuccessfully]"
    WackReportFilePath="[WackReportFilePath]"
    WackReportFilename="ValidationResult.html"
    WackResultsFilename="ValidationResult.xml" />
  <If Condition="[Not WackToolRanSuccessfully]"
    sap2010:WorkflowViewState.IdRef="If_4">
    <If.Then>
      <Sequence sap2010:WorkflowViewState.IdRef="Sequence_6">
        <mtbwa:WriteCustomSummaryInformation
          sap2010:WorkflowViewState.IdRef=
            "WriteCustomSummaryInformation_2">
          Message="[&quot;Windows App Certification Kit ran with errors.
            Click [here]&quot; &amp; WackReportFilePath &amp; &quot;)&quot; to
            access the folder containing the report.&quot;]"
          SectionDisplayName="Windows 8" SectionKey="Windows8"
          SectionPriority="75"
          mva:VisualBasic.Settings=
            "Assembly references and imported namespaces
              serialized as XML namespaces" />
        <mtbwa:WriteBuildError
          sap2010:WorkflowViewState.IdRef="WriteBuildError_1">
          Message="Windows App Certification Kit ran with errors." />
        <mtbwa:SetBuildProperties
          CompilationStatus=
            "[Microsoft.TeamFoundation.Build.Client.BuildPhaseStatus.Failed]"
          DisplayName="Set Status and CompilationStatus to Failed"
          sap2010:WorkflowViewState.IdRef="SetBuildProperties_1">
          mtbwt:BuildTrackingParticipant.Importance=
            "Low" PropertiesToSet="CompilationStatus" />
      </Sequence>
    </If.Then>
    <If.Else>
      <mtbwa:WriteCustomSummaryInformation
        sap2010:WorkflowViewState.IdRef="WriteCustomSummaryInformation_1">
        Message="[&quot;Windows App Certification Kit ran with success.
          Click [here]&quot; &amp; WackReportFilePath &amp; &quot;)&quot; to
          access the folder containing the report.&quot;]"
        SectionDisplayName="Windows 8" SectionKey="Windows8"
        SectionPriority="75"
        mva:VisualBasic.Settings=
          "Assembly references and imported namespaces
            serialized as XML namespaces" />
    </If.Else>
  </If>
</Sequence>
```

of the “OVERALL\_RESULT” attribute and returns that value. **Figure 7** shows the code that creates the activity CheckWackResultsActivity.

By default, the build activities run on build agents. But there might be some scenarios where you want the activity to execute as early as the first step, even before the build starts, or as the last step before the build is finalized. For that kind of flexibility, you need to have the activity run on the controller, not on the agent. You can do this using the attribute BuildActivityAttribute, which takes as an argument the enumeration value HostEnvironmentOption. All (as you can see in **Figure 7**). Note that if you don’t use the correct Host-EnvironmentOption option, you’ll get an error during the build process.

The class CheckWackResultsActivity inherits from Code-Activity<bool> so that its result value can be used to display the correct message in the build report. To display this message, you can use a new activity available in TFS 2012: WriteCustomSummaryInfo. This activity is very useful if you want to add a message to the build report because, instead of adding simple text, it allows you to add a dedicated category in the build report.

You have to specify the following properties:

- Message, which is the text to display in the report
- SectionDisplayName, which corresponds to the header of the section
- SectionKey, the unique value of the section
- SectionPriority, which defines the position of the new section in the report (0 is the highest priority and the standard sections start at 100)

So, using the new activity and WriteCustomSummaryInfo, I’m able to modify the build process to check for the validation results and add a new section in the build report. **Figure 8** shows the XAML code of the modified build process.

To be published to the Windows Store, an app must be certified.

Notice in **Figure 8** that if the validation fails, the compilation status is set to “Failed” to prevent the build process from continuing. This isn’t mandatory and you can remove it if you prefer to always finish the build process, regardless of the validation results.

Now, each time a build is triggered, the build report displays a new section, dedicated to Windows 8, which shows the result of the validation (see **Figure 9**).

Using WriteCustomSummaryInfo, the build report can be enhanced with text and links only. If a more complex modification is needed (for example, adding an image), you can still apply the techniques used with TFS 2010.

There are a lot of possibilities for customizing the build process template for Windows Store apps, and the good news is that such customizations are pretty much the same for both Team Foundation Service and on-premises TFS.

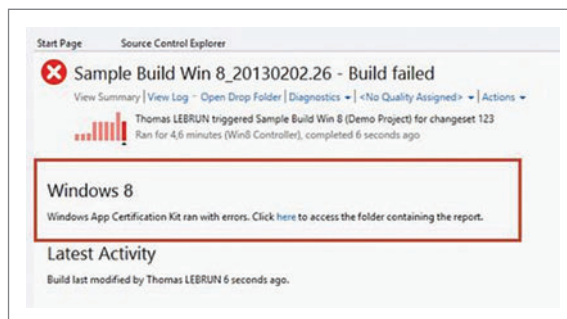


Figure 9 The New Section in the Build Report Showing Validation Results

**THOMAS LEBRUN** is technical leader at Infinite Square, a French Microsoft partner working on technologies including Windows 8/ Windows Phone, Team Foundation Server, SharePoint and more. The author of two French books about Windows Presentation Foundation and the Model-View-ViewModel pattern, Lebrun is also a regular speaker during French events. Follow his blog at [blog.thomaslebrun.net](http://blog.thomaslebrun.net) and follow him on Twitter at [twitter.com/thomas\\_lebrun](https://twitter.com/thomas_lebrun).

**THANKS** to the following technical expert for reviewing this article: Chris Patterson (Microsoft)

# Does your Team do more than just track bugs?

Free Trial and Single User FreePack™ available at [www.alexcorp.com](http://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.



**Alexsys Team**

## 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

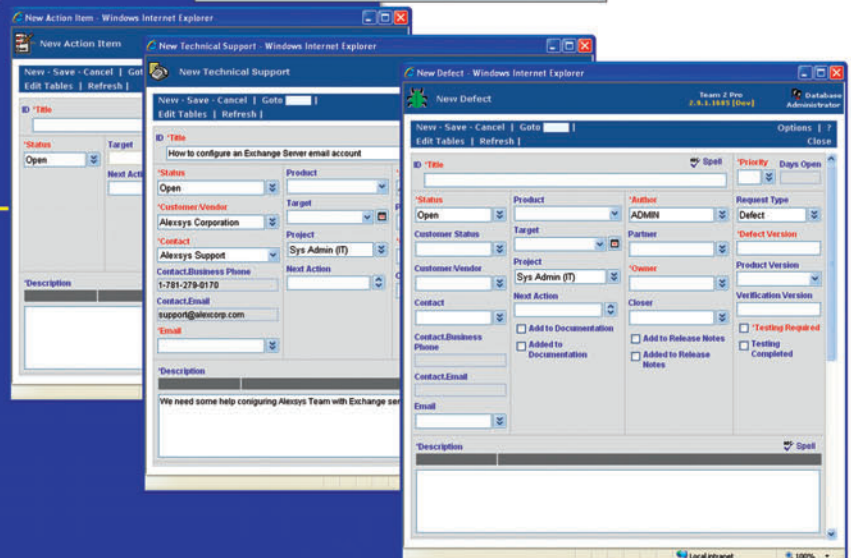
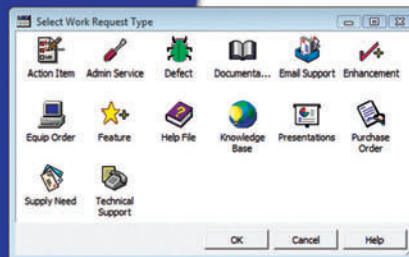
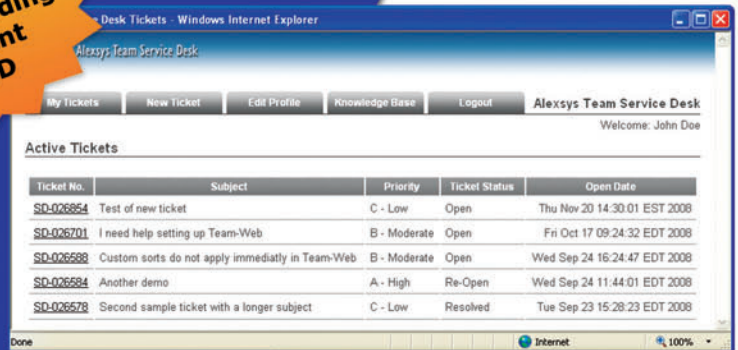
- Full Windows 7 Support
- Windows Single Sign-on
- System Audit Log
- Trend Analysis
- Alternate Display Fields for Data Normalization
- Lookup Table Filters
- XML Export
- Network Optimized for Enterprise Deployment

### 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



**Free Trial and Single User FreePack™ available at [www.alexcorp.com](http://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.

# A Treasure Hunt Through ALM Readiness

## Visual Studio ALM Rangers

**In this article**, we introduce the sample Windows Store app, ALM Readiness Treasure Map, and share the design, coding and testing experiences of the Visual Studio ALM Rangers in building the app. It's designed to provide a master catalog of the content available to help developers become proficient in Application Lifecycle Management (ALM) practices. The ALM Rangers are a group of experts who promote collaboration among the Visual Studio product group, Microsoft Services and the Microsoft MVP community by addressing missing functionality, removing adoption blockers, and publishing best practices and guidance based on real-world experiences.

### What Did We Do and Why?

We have a confession to make: We love Visual Studio and Visual Studio Team Foundation Server (TFS). Microsoft produces some of

the best software development tools available. We're not just saying that because we work for Microsoft—we felt this way long before joining the company. These suites provide an incredible number of features but can prove daunting to new users. How do developers start learning to use these tools? This question presented itself to us in a slightly different fashion.

ALM Rangers present at TechReady conferences that typically have a number of sessions describing how to improve knowledge of Microsoft development tools. The conferences even hold interactive sessions where participants can provide feedback to the internal groups building these products. We found these to be exciting opportunities for developers and consultants. After joining the ALM Rangers team, we started exploring the published guidance and quickly realized there was a significant amount of content to digest; we were unsure of where to begin studying.

What we really wanted was a resource to help us become proficient in ALM practices. We began building the ALM Readiness Treasure Map Windows Store app to lead users through the materials on their journey to becoming experts. **Figure 1** shows you the results of our work. It contains five categories, each with multiple topics of study:

1. Prepare
2. Quick intro
3. Guidance
4. Tooling
5. Workshops

These areas contain guides, hands-on labs, and videos to make it as easy as possible for users to pick up the skills they need quickly

#### This article discusses:

- Why the app was built
- UX design considerations
- Coding goals
- Testing and quality assurance

#### Technologies discussed:

Visual Studio, Team Foundation Server, C#, XAML, Windows Store Apps

#### Code download and map available at:

[aka.ms/almtreasure](http://aka.ms/almtreasure)



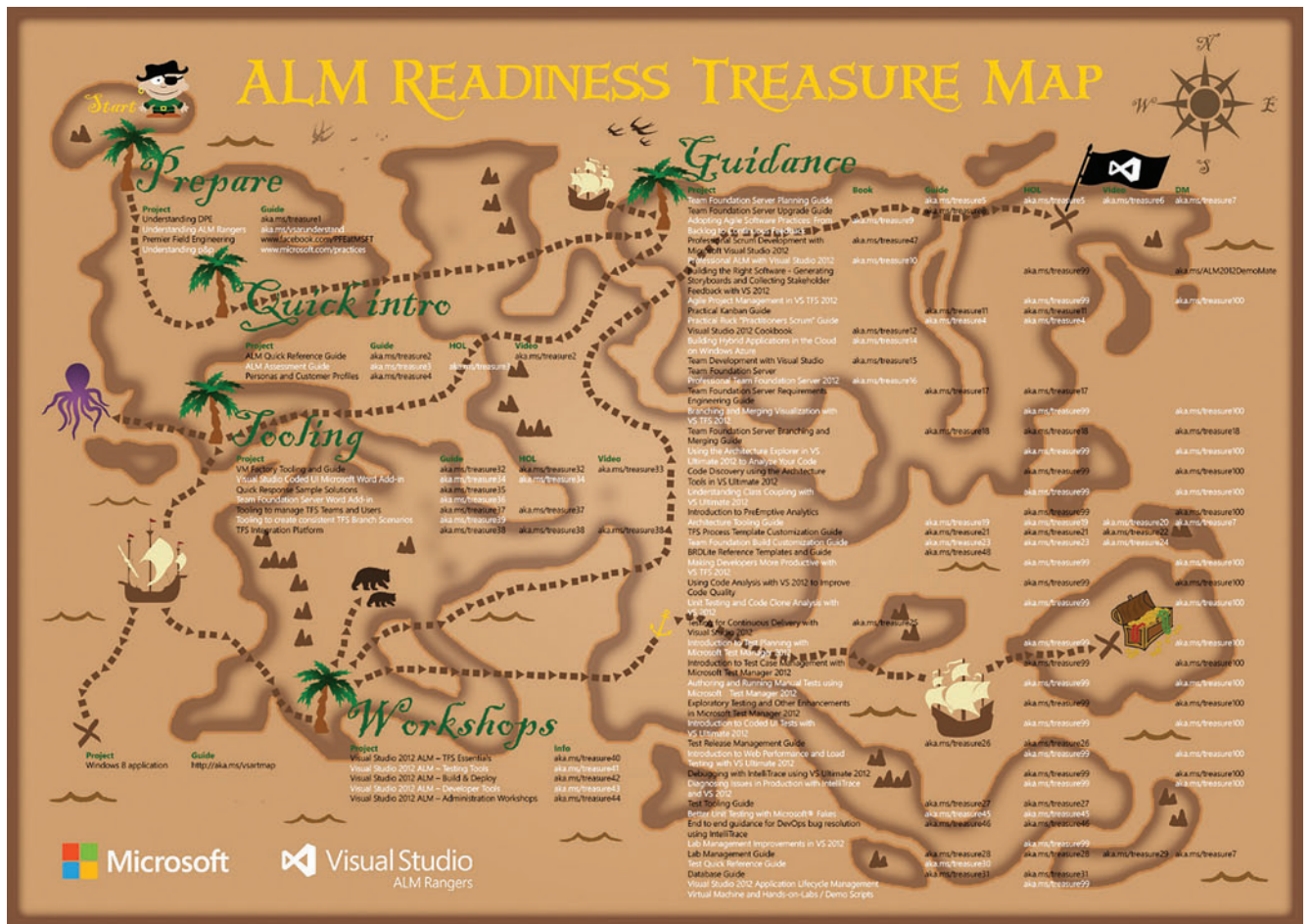


Figure 1 The Treasure Map Windows Store App

and effectively. Navigating the materials works particularly well on new touch-enabled devices, such as the Microsoft Surface.

To provide the optimal experience, we enlisted the help of an expert UX designer and senior developers to build the app.

## The ALM Readiness Treasure Map Solution: UX Design Nuggets

**Making a Good First Impression** The treasure map tile (see Figure 2) is eye-catching and bright, with an orange background used to symbolize sand. Immediately, the user sees the theme of the app—this is clear because of the palm tree and the path leading to where “X” marks the spot of the treasure. The app title is clearly visible on the tile. Making sure that the tile depicts what your app is actually about creates a good first impression. The last thing you want is for users to open your app and be confused about why they’re using it and how it can help them.

We’ve leveraged the splash screen (see Figure 3) to show a bit more of the app’s personality and to help ensure a good launch experience. The treasure map’s splash screen renders an extended path to the treasure, which is a smooth, polished loading experience. It’s uncluttered and straightforward by design, reflecting how the UX will be. In addition, a unique screen can help reinforce the brand.

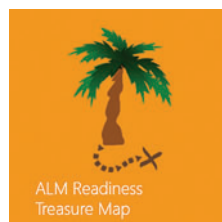


Figure 2 The Treasure Map Windows Store App Tile

The homepage—the treasure map itself—appears once the app has loaded. Again, our clear, content-focused design immediately confirms the purpose of the app. We wanted to make this page fantastic so the user would want to explore the rest of the app. The homepage is where the journey begins and, at a glance, the user knows this is going to be a journey. The titles are the source for navigation, and because of our “content over chrome” design, they stand out. The app’s content is emphasized by removing any non-functional elements. Anyone looking to find information quickly can find all the links on the screen without having to navigate through the app, which provides a great experience for all types of users.

**Sometimes Overlooked, but Crucial** The Windows 8 UI uses the principle of a grid system across all its apps. This principle promotes a clean, uncluttered design.

The Treasure Map app employs the grid system everywhere except on the homepage and the result is that content is highly visible—more content, less chrome. This content over chrome principle is one of the more unique principles of the Windows Store app style, where visual unity contributes to a great UX. The homepage is the only page that’s an exception to this rule. We’re portraying a journey and a pirate theme that we wouldn’t have been able to achieve without visual representations.



Figure 3 The Treasure Map Windows Store App Splash Screen

Typography is sometimes overlooked and many people don't realize how much it can strengthen the brand of the app. Using fonts correctly and consistently helps achieve clarity and gives a clean, uncluttered look that makes the app easier to read and therefore use. The recommended fonts are Segoe UI (primarily used for UI elements like buttons), Calibri (used for reading and writing, such as in e-mails) and Cambria (for larger blocks of text). We used Segoe UI for all text other than the headers. For those, we used Blackadder ITC to establish a stronger theme (see Figure 4). We do break the font rule here, because we wanted the app's appearance to be consistent with a paper-based map, as this helps to reinforce the pirate theme. So, in this case, it does work well.

Seamless and fluid navigation is crucial to provide that ease of use and great UX. Two forms of navigational patterns are recommended: the hierarchical system and the flat system (see Figure 5). The hierarchical system is what most apps use. It's the most common and will be the most familiar type of navigation to many users. It's also the best system to create that fluid feel yet still be easy to use. The flat system is used mainly in games, browsers, or document-creation apps, where the user can only go backward and forward at the same hierarchical level. The Treasure Map app uses the hierarchical system, and we believe that it uses it well. The homepage would be classified as the hub, and each section creates the first hierarchical branch, from where each section then creates another hierarchical branch. For example, from the homepage, the user can navigate to the Prepare section, where the user can explore other Prepare subsections.

**Usability** It's important to assess the UX of the app to improve the design so that the app is:

- Easy to use
- More valuable to users—for example, in the features it can offer
- More desirable to use

Assessing your design gives you confidence that the app has an outstanding UX and that users will find it useful, usable and desirable.

So how do we assess the UX of the app? There are many ways to do this, but two common ones are self-assessments and cognitive walkthroughs, as shown in Figure 6.

There are four success metrics that will help in both the self-assessment of the app as well as the cognitive walkthrough of the app. These are:

- **Great at:** What's the app great at? What are the focal points of the visuals?
- **Usable:** What should users be able to understand, know or do more successfully because of the app?
- **Useful:** What do you want the users to value?
- **Desirable:** What parts of the app do you want to delight users or make them love?

We used both self-assessments and cognitive walkthroughs. The self-assessments were carried out through each sprint and reviewed at our weekly stand-ups. The cognitive walkthroughs were carried out during the design process and through every sprint. Assessing the UX of our app helped us to understand the desire and emotional connection to experiences that a user may acknowledge.

To sum up UX design:

- Make sure that the tile depicts what your app is about.
- Create a unique splash screen to reinforce your brand.
- Write content with a clear focus.
- Use the recommended grid layout to create a simple and clean design.
- Don't forget about typography. Use the recommended fonts where possible, such as Segoe UI, Calibri or Cambria.
- Have a clear navigation pattern. Choose from either the hierarchical system or the flat system.
- Assess the usability of your app throughout the development cycle.

## The Coding Jewels

Before coding started, we set forth a series of coding goals for this project. These goals became the mantra for how we designed and developed the codebase.

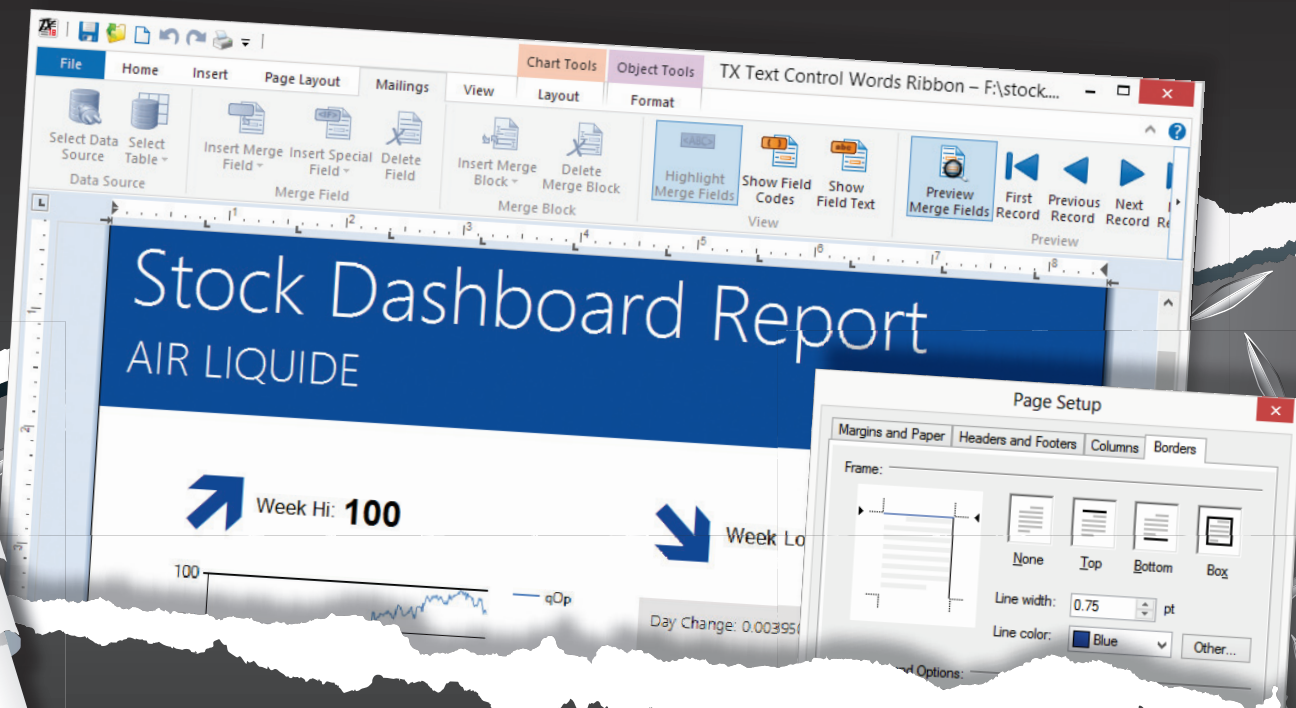
- **Adaptability:** Requirements can change, features can be added or cut, and designs can be thrown away in favor of something completely different. Adaptability is the name of the game!
- **Simplicity:** Simplicity is essential for many advantages in software design, in particular for maintainability and fixability.
- **Testability:** Quality assurance must be a high priority for every project, and the codebase must allow for comprehensive and "simple" testing.
- **Performance and fluidity:** The app's UX must be positive from the start. The app must display information in a timely manner, must always be responding to user input and mustn't lag.
- **Team environments:** Rarely is an app built by an individual contributor or even an individual team. We made sure that the app was built in a way that could be scaled to many more team members.

So now that we had our goals defined, how in the world did we achieve writing an app in such a short time—working part-time—while not only meeting the functional requirements, but also our non-functional requirements as well? Luckily for us, this wheel has been built before and the construction of our app was a matter of applying proven patterns and practices to our app:



Figure 4 The Typography We Used in the Treasure Map Windows Store App

# FLOW TYPE LAYOUT REPORTING



Reuse MS Word documents or templates as your reporting templates.



Easy database connection with master-detail nested blocks.



Powerful, programmable template designer with full sources for Visual Studio®.



Integrate dynamic 2D and 3D charting to your reports.



Create print-ready, digitally signed Adobe PDF and PDF/A documents.



Create flow type layouts with tables, columns, images, headers and footers and more.

**TX**  
**TEXTCONTROL®**  
word processing components



Visual Studio

Microsoft

Partner

US +1 855-533-8398

EU +49 421-4270671-0

[WWW.TEXTCONTROL.COM](http://WWW.TEXTCONTROL.COM)



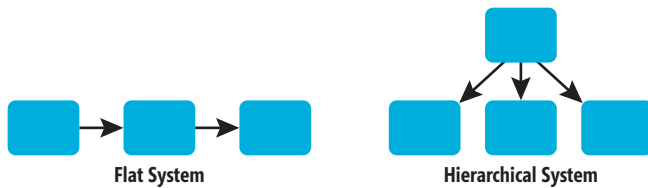


Figure 5 Recommended Navigational Patterns

- **C# and XAML:** We decided to use C# and XAML, primarily because the majority of the contributors on this project are familiar with this approach. This includes the languages themselves as well as the tooling and support for them.
- **Model-View-ViewModel (MVVM):** This is a pattern for separating your presentation layer from your business logic from your objects. We chose this particular pattern because the C# and XAML technologies lend themselves very well to it. But more important, with a single pattern, we were able to begin chiseling away at our non-functional requirements. The goals that MVVM most positively affected were adaptability, testability, team environments and simplicity. Adaptability is improved in that you can interchange any of the functional pieces of your app. Perhaps a new presentation for a particular view model has been finished, and you can instantly replace it without changing any other code. Testability is improved because each core functional piece of code is separated into its own individual responsibilities, which means tests can be written against those directly and automated. Team environments are improved because you have a defined set of contracts among the app's moving parts, allowing teams to work in parallel with one another. Simplicity is improved in that each moving part is its own defined moving part and interacts in a specific way. For more information, see "What's New in Microsoft Test Manager 2012" at [msdn.microsoft.com/magazine/jj618301](http://msdn.microsoft.com/magazine/jj618301).
- **Resources:** Following the spirit of MVVM and separation of roles and replaceable parts, we decided to add resources for the definition of our fonts, buttons and other similar design elements. This helped improve team environments, simplicity and adaptability.
- **But what did we do about performance and fluidity?** We followed the async/await pattern for long-running processes. This is one of the areas where developers might struggle in building Windows Store apps, but it doesn't have to be. Luckily, Windows Store apps using C# and XAML are powered by the Microsoft .NET Framework 4.5, which lets you easily include asynchronous workloads into your app through this pattern. If it's so easily done, why do folks struggle with it? The answer to this question usually boils down to using logic that's long-running and isn't provided out of the box by the .NET Framework. An example of this could be logic to calculate plot points for a chart based on complex mathematics. A full understanding of async and await is crucial for providing a fluid, high-performing app. For more information, see "Async Performance: Understanding the Costs of Async and Await" at [msdn.microsoft.com/magazine/hh456402](http://msdn.microsoft.com/magazine/hh456402).

Other considerations included:

- **Touch language:** From a development perspective, this couldn't be any easier. Nearly every out-of-the-box control supports touch in all of the ways that you expect. From a coding perspective, this was the easiest part of the app to build for.
- **Charms:** Interacting with the charms was simple as well. Register these in your appxmanifest and add in the event handlers to each page for the specific charms you want to register, such as Search or Share. We had no issues dealing with charms. They worked well, like a charm should work.
- **Tiles, splash screens and orientations:** All of these were handled in the appxmanifest and then through event hooks in the app at the app level. It was straightforward, and everything is detailed in the sample code.

**How It All Really Worked** Here's how things worked out in actual practice:

- **MVVM commands:** MVVM was fantastic in theory. However, in reality, it proved a bit different from the usual Windows Presentation Foundation (WPF) and Silverlight development, particularly for implementing commands, because our old samples didn't work. Luckily, `Command<T>` was fairly easy to implement in the new framework and can be seen in our sample app. But our woes didn't end with commands, because `ListViewBase` items don't have an attached command property! We decided to solve this for demonstrational purposes in two ways:
  1. First, we decided to solve this problem using an unused property:
 

```
<ListView Grid.Column="2"
  SelectedItem="{Binding Selected, Mode=TwoWay}"
```

 The property to which it's bound returns "null" and doesn't throw any exceptions (even if you turn all exceptions on), which is nice, but the key is in the set. In the set, instead of setting anything, we make a navigation call and pass as a parameter the index of the selected item.
  2. Second, we decided to create an attached dependency property of type  `ICommand`. The sample implementation is in the class "ItemClickCommand" within the folder "MVVMSupport".
- **Multiple view states lead to massive view files:** Our view files became extremely large and more difficult to

Figure 6 Assessing the UX of the App

	Self-Assessment	Cognitive Walkthrough
Why	This is based on the goals that you want the user to achieve or find. It ensures that the design is on track with your main intentions.	This is a little more structured around the specific tasks that a user might want to fulfill, for example, to find information about "VM factory tooling and guidance."
When	It's a good idea to do self-assessments every sprint, or when each goal has been reached; they last up to 30 minutes.	During the design process, and through every sprint.

# SpreadsheetGear

## Performance Spreadsheet Components

### SpreadsheetGear 2012 Now Available

**NEW!**

WPF and Silverlight controls, multithreaded recalc, 64 new Excel compatible functions, save to XPS, improved efficiency and performance, Windows 8 support, Windows Server 2012 support, Visual Studio 2012 support and more.

### Excel Reporting for ASP.NET, WinForms, WPF and Silverlight



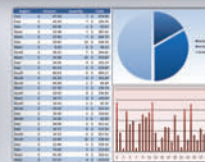
Easily create richly formatted Excel reports without Excel from any ASP.NET, Windows Forms, WPF or Silverlight application using spreadsheet technology built from the ground up for performance, scalability and reliability.

### Excel Compatible Windows Forms, WPF and Silverlight Controls



Add powerful Excel compatible viewing, editing, formatting, calculating, filtering, charting, printing and more to your Windows Forms, WPF and Silverlight applications with the easy to use WorkbookView controls.

### Excel Dashboards, Calculations, Charting and More



You and your users can design dashboards, reports, charts, and models in Excel or the SpreadsheetGear Workbook Designer rather than hard to learn developer tools and you can easily deploy them with one line of code.

**Free  
30 Day  
Trial**

Download our fully functional 30-Day evaluation and bring Excel Reporting, Excel compatible charting, Excel compatible calculations and much more to your ASP.NET, Windows Forms, WPF, Silverlight and other Microsoft .NET Framework solutions.

[www.SpreadsheetGear.com](http://www.SpreadsheetGear.com)



 SpreadsheetGear

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



Figure 7 Exploratory Testing

manage, primarily due to multiple view states. A different layout per view state is generally required, and you could even have different views for different display dimensions or changing dimensions, and so on. Our approach was to split them up into multiple .xaml files, using one .xaml file per view per state. For example, if we had a view called “HomePageView,” we’d have “HomePageView.xaml” inside a full, snapped and filled folder, each one in the folder for its state.

- **Adapting in the real world:** This was a good story. After we had developed the large parts of our app—switching data providers, switching UIs and adding new charm interactions to all of the appropriate places—adapting it to changing requirements became a piece of cake. Issues were easy to track down, and much of the planning paid off because the designers could work in parallel with the developers and the sponsors.

To summarize this section, from a coding standpoint, Windows Store apps are straightforward to develop properly. Following a few predefined rules, mentalities and patterns lets you create a good app quickly. The primary concerns are app view state, app lifecycle state, charm interaction, fluidity and ensuring that you code in a way that allows your design team to iterate designs at will. For more information, see [dev.windows.com](http://dev.windows.com).

## Testing and Verifying Solution Quality

One of the core acceptance requirements we defined from the start was to raise the sample code quality and be part of our overall quality dog-fooding program (see [aka.ms/df\\_tooling](http://aka.ms/df_tooling)). Stringent quality gates for geopolitical, namespace, code analysis and StyleCop (see [stylecop.codeplex.com](http://stylecop.codeplex.com)) compliance helped us produce a better solution, albeit only a sample.

Testing should *not* be an afterthought and is as important with samples as it is with mission-critical solutions. It’s easier to enforce code quality, and to manage user expectations and requirements from

the start, rather than be faced with hundreds of compliance bugs and irate testers and have to deal with feature and code churn during a belated quality-improvement cycle.

Because the intention was a quick sample solution, we primarily adopted a “black box” testing strategy focusing on behavior as part of the system and user acceptance testing (UAT). The former was manually performed by the team, focusing on expected features and non-functional requirements, and exploring edge-cases where the internals were understood. The community was invited to assist with the UAT validation, performing exploratory testing based on real-world scenarios and expectations, and

unearthing bugs and missing, impractical, and unclear features.

As shown in **Figure 7** (with the numerals here corresponding to the red-circled numbers in the figure), we typically used the Microsoft Test Manager exploratory testing feature (1) and the simulator (2) to evaluate the sample solution on a Surface-type device, captured detailed comments (3) and recorded the test session (4) for future reference.

In the future, we’ll seriously consider the definition of more-structured test cases and the use of Microsoft Fakes to help us implement the unit and smoke testing. This will allow us to automatically and continuously validate feature changes and associated code changes.

## What’s Next?

We will evolve the ALM Readiness Treasure Map app, and we’re considering an online update feature for the readiness reference assets. For more information, see “Understanding the Visual Studio ALM Rangers” at [aka.ms/vsarunderstand](http://aka.ms/vsarunderstand); “Visual Studio ALM Ranger Solutions” at [aka.ms/vsarsolutions](http://aka.ms/vsarsolutions); the ALM Readiness Treasure Map sample code at [aka.ms/almtreasure](http://aka.ms/almtreasure); and the app itself in the Windows Store at [aka.ms/vsartmapapp](http://aka.ms/vsartmapapp). We welcome your candid feedback and ideas! ■

**ANISHA PINDORIA** is a UX developer consultant with Microsoft Consulting Services in the United Kingdom.

**DAVE CROOK** is a developer consultant with Microsoft Consulting Services East Region, where his focus is Visual Studio and Team Foundation Server.

**ROBERT BERNSTEIN** is a senior developer with the Microsoft Consulting Services Worldwide Public Sector Cyber Security Team.

**ROBERT MACLEAN** is a software developer nestled in a typical open-plan development office at BBD ([bbd.co.za](http://bbd.co.za)).

**WILLY-PETER SCHAUB** is a senior program manager with the Visual Studio ALM Rangers at the Microsoft Canada Development Center.

**THANKS** to the following technical expert for reviewing this article:  
Patricia Wagner (Microsoft)





# Extreme Performance & Linear Scalability

Remove data storage and database performance bottlenecks and scale your applications to extreme transaction processing (XTP). NCache lets you cache data in memory and reduce expensive database trips. It also scales linearly by letting you add inexpensive cache servers at runtime.

## Enterprise Distributed Cache

- Extremely fast & linearly scalable with 100% uptime
- Mirrored, Replicated, Partitioned, and Client Cache
- NHibernate & Entity Framework Level-2 Cache

## ASP.NET Optimization in Web Farms

- ASP.NET Session State storage
- ASP.NET View State cache
- ASP.NET Output Cache provider
- ASP.NET JavaScript & image merge/minify

## Runtime Data Sharing

- Powerful event notifications for pub/sub data sharing

Download a 60-day FREE trial today!



Distributed Cache for .NET & Java

[www.alachisoft.com](http://www.alachisoft.com)

1-800-253-8195



# Classification and Prediction Using Adaptive Boosting

James McCaffrey

**Classification is a machine-learning** technique that uses training data to generate a model (usually a single complex rule or mathematical equation) that assigns data items to one of several distinct categories. The model can then be used to make predictions about new data items whose category is unknown. Examples include predicting whether a patient has cancer (yes, no) based on various medical test data, and predicting the risk category (low, medium, high) of a loan application based on an applicant's financial history.

There are many different classification algorithms and techniques, including Naive Bayes, neural network and logistic regression. In this article I explain a fascinating technique called adaptive boosting classification in which, instead of attempting to determine a single complex prediction rule, training data is used to generate a large collection of very simple crude rules of thumb. A weight for each rule of thumb is then computed. A prediction about new input is

made by combining the rules of thumb, taking into account each simple rule's weight and arriving at a consensus outcome. The term "boosting" comes from the fact that the predictive quality of the simple rules is boosted (improved) by combining them.

Adaptive boosting is a meta-heuristic. By that I mean adaptive boosting is a set of guidelines that can be used to create a specific classification algorithm. There are many variations of adaptive boosting algorithms and there are many existing standalone tools that implement some form of adaptive boosting, so why bother to code adaptive boosting classification from scratch? Existing adaptive boosting classification tools can be difficult or impossible to customize, they might be difficult to integrate into a software system, and they may have copyright or intellectual property issues.

A concrete example is the best way to understand what adaptive boosting is. Take a look at **Figure 1**. The ultimate problem of the demo program is to predict whether some sports team from Seattle will win or lose an upcoming contest against a team from Detroit, when Seattle will be playing on its home field and the betting consensus (the spread) is that Seattle has a slight advantage (small). The top part of **Figure 1** shows 10 hypothetical training data items with known results. The first training tuple (Detroit, Home, Large, Win) means that in a previous game against Detroit, when Seattle played at home and the point spread was large, Seattle won the game.

The next part of the demo program shows that the 10 training data items were converted from string data into zero-based integer form for more efficient processing and then stored into machine memory as a matrix. For example, (Detroit, Home, Large, Win) is stored as (3, 0, 2, 1). Notice that in this example, the outcome to be

## This article discusses:

- Adaptive boosting classification
- A demo program based on adaptive boosting
- The adaptive boosting algorithm
- The overall program structure

## Technologies discussed:

Visual Studio 2010, C#

## Code download available at:

[archive.msdn.microsoft.com/mag201304AdaptiveBoost](http://archive.msdn.microsoft.com/mag201304AdaptiveBoost)

predicted has just two values, win or lose. This is called a binary classification problem. Adaptive boosting can also be used in situations where the dependent variable has three or more values (multinomial classification). Most binary classification techniques encode the dependent variable to be predicted using a (0, 1) scheme, but adaptive boosting almost always uses a (-1, +1) scheme because that encoding slightly simplifies some of the algorithm implementation. Notice that all of the independent variable predictor values are categorical (“Home,” “Medium” and so on) rather than numerical. Adaptive boosting can also be used when training data has numerical values, as I’ll explain later.

The third part of **Figure 1** shows that eight rules of thumb were generated from the training data. Rules of thumb are often called weak learners or weak classifiers in adaptive boosting terminology. The first weak learner, expressed in human-friendly form, is “IF Opponent IS Buffalo THEN Result IS Win” with a 0.00 raw error rate. The second weak learner, which is more illustrative, is “IF Opponent IS Chicago THEN Result IS Lose” with a 0.33 error rate. Where did this weak learner come from? If you examine the training data, you’ll see that there are three instances where the opponent is Chicago. In two of those training instances the result was Lose, so the weak learner is correct two out of three times (0.67) and wrong one time out of three (0.33).

Notice that not all training item predictor values generated a weak learner—there’s no rule for when the opponent is Atlanta. Because there are two training tuples where opponent is Atlanta, and one outcome is Win and the other outcome is Lose, the error rate for the learner would be 0.50, which doesn’t provide any useful information. The version of adaptive boosting classification presented in this article assumes that all weak learners have a raw error rate that is less than 0.50.

The next section of **Figure 1** indicates that behind the scenes the adaptive boosting algorithm processes the weak learners to find a weight for each learner. These weights are measures of the importance of each weak classifier, and are called alpha values in adaptive boosting terminology. Determining the alpha values is the key part of adaptive boosting. The set of weak learners and their alpha weights make up the classification model.

The last part of the demo program shows the classification model being used to make a prediction for team Seattle when the opponent team is Detroit, the field is Home and the point spread is Small. If you refer to the set of generated weak learners, you’ll see that three learners are applicable: [2] IF Opponent IS Detroit THEN Result IS Win (+1), [3] IF Field IS Home THEN Result IS

Win (+1), and [5] IF Spread IS Small THEN Result IS Lose (-1). The computed alpha values for weak learners 2, 3, and 5 are 0.63, 3.15, and 4.49, respectively. Therefore, the consensus prediction = (0.63)(+1) + (3.15)(+1) + (4.49)(-1) = -0.72 (rounded), which, because the value is negative, means Lose. Notice that even though two of three weak learners (2 and 3) predict Win, the large alpha of weak learner 5 outweighs those Win predictions to yield an overall prediction of Lose.

In the sections that follow I’ll carefully explain how the demo program works so that you can add prediction features to a .NET system or application. This article assumes you have advanced programming skills with a C-family language but does not assume you know anything about classification with adaptive boosting. I coded the demo using C# but the explanation should give you enough information to refactor the code to other languages such as Visual Basic .NET or Python. The code for the demo program is too long to fully list in this article, but the complete source code is available at [archive.msdn.microsoft.com/mag201304AdaptiveBoost](http://archive.msdn.microsoft.com/mag201304AdaptiveBoost).

```

file:///C:/AdaptiveBoosting/bin/Debug/AdaptiveBoosting.EXE
Begin adaptive boosting classification demo
Raw <string> training data for team Seattle:
Opponent Field Spread Result
=====
Detroit Home Large Win
Detroit Away Medium Win
Buffalo Home Small Win
Buffalo Home Medium Win
Atlanta Away Large Win
Chicago Home Medium Win
Chicago Away Small Lose
Chicago Home Small Lose
Atlanta Away Medium Lose
Detroit Away Large Lose

Converting and storing training data.
Training data in int form:
3 0 2 -> +1
3 1 1 -> +1
1 0 0 -> +1
1 0 1 -> +1
0 1 2 -> +1
2 0 1 -> +1
2 1 0 -> -1
2 0 0 -> -1
0 1 1 -> -1
3 1 2 -> -1

Creating weak categorical stump learners from training data.
Completed. Weak learners are:
[0] IF Opponent IS Buffalo THEN Result IS Win <raw error = 0.00>
[1] IF Opponent IS Chicago THEN Result IS Lose <raw error = 0.33>
[2] IF Opponent IS Detroit THEN Result IS Win <raw error = 0.33>
[3] IF Field IS Home THEN Result IS Win <raw error = 0.20>
[4] IF Field IS Away THEN Result IS Lose <raw error = 0.40>
[5] IF Spread IS Small THEN Result IS Lose <raw error = 0.33>
[6] IF Spread IS Medium THEN Result IS Win <raw error = 0.25>
[7] IF Spread IS Large THEN Result IS Win <raw error = 0.33>

Initializing list of best learner indexes.
Using adaptive boosting to find best learners and their alpha values.
Initializing training tuple weights <'D'> array.
Entering main algorithm loop.
Model completed.
Algorithm found 8 good learners and associated alpha values.
The good learners and their alpha value are:
[0] 6.91 [1] 2.31 [2] 0.63 [3] 3.15 [4] 0.59 [5] 4.49 [6] 0.68 [7] 0.63
Predicting outcome when Opponent = Detroit, Field = Home, Spread = Small.
Using learner 2 with alpha = 0.63 prediction is +1 so vote = 0.63
Using learner 3 with alpha = 3.15 prediction is +1 so vote = 3.15
Using learner 5 with alpha = 4.49 prediction is -1 so vote = -4.49
Final accumulated vote is -0.72
Predicted Y = -1 => Seattle will Lose
End

```

Figure 1 Adaptive Boosting Classification and Prediction



## The Adaptive Boosting Algorithm

The heart of adaptive boosting classification is a routine that examines each weak learner and assigns an alpha weight to each. The algorithm is quite tricky and best explained by a concrete example. Suppose there are 10 training tuples and eight weak learners, as shown in **Figure 1**. Each training tuple is assigned a weight, usually called D in adaptive boosting literature. The sum of the D weights is 1.0, making the D values a distribution. Initially all training data items are assigned equal D weights, in this case 0.10 because there are 10 items:

```
[0] (D = 0.10) Detroit   Home   Large   Win
[1] (D = 0.10) Detroit   Away   Medium  Win
[2] (D = 0.10) Buffalo   Home   Small  Win
[3] (D = 0.10) Buffalo   Home   Medium Win
[4] (D = 0.10) Atlanta   Away   Large  Win
[5] (D = 0.10) Chicago   Home   Medium Win
[6] (D = 0.10) Chicago   Away   Small  Lose
[7] (D = 0.10) Chicago   Home   Small  Lose
[8] (D = 0.10) Atlanta   Away   Medium Lose
[9] (D = 0.10) Detroit   Away   Large  Lose
```

Each learner has an epsilon value and an alpha value. Epsilon values are weighted error rates used to compute alpha values. Initially all learners have unknown alpha values (say  $a = 0.00$ ) and unknown epsilon values (say  $e = -1.0$ ):

```
[0] (a = 0.00) (e = -1.0) IF Opponent IS Buffalo THEN Result IS Win
[1] (a = 0.00) (e = -1.0) IF Opponent IS Chicago THEN Result IS Lose
[2] (a = 0.00) (e = -1.0) IF Opponent IS Detroit THEN Result IS Win
[3] (a = 0.00) (e = -1.0) IF Field IS Home THEN Result IS Win
[4] (a = 0.00) (e = -1.0) IF Field IS Away THEN Result IS Lose
[5] (a = 0.00) (e = -1.0) IF Spread IS Small THEN Result IS Lose
[6] (a = 0.00) (e = -1.0) IF Spread IS Medium THEN Result IS Win
[7] (a = 0.00) (e = -1.0) IF Spread IS Large THEN Result IS Win
```

In pseudo-code, the algorithm to find the alpha weights for each learner is:

```
set t=0
while not done loop
    update all learners' epsilons (weighted errors)
    find best (smallest epsilon) unused learner
    compute and save the alpha of best learner using its epsilon
    update the D weights for each training item using the best learner
    normalize the D weights so they sum to 1.0
    ++t
end loop
```

The main processing loop terminates when all weak learners have been processed and assigned an alpha weight; or when the loop counter variable  $t$  exceeds some maximum value; or when the weighted error rate, epsilon, for the best unused weak learner is some value, such as 0.45 or 0.49, indicating there aren't any relatively good unused learners left to process.

Figure 2 Overall Program Structure

```
using System;
using System.Collections.Generic;
namespace AdaptiveBoosting
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                Console.WriteLine("\nBegin adaptive boosting classification demo\n");

                string[] features = new string[] { "Opponent", "Field", "Spread", "Result" };

                string[][] values = new string[4][];
                values[0] = new string[] { "Atlanta", "Buffalo", "Chicago", "Detroit" }; // opponent
                values[1] = new string[] { "Home", "Away" }; // Field
                values[2] = new string[] { "Small", "Medium", "Large" };
                // Note: Spaces added
                values[3] = new string[] { "Lose", "Win" };
                // The dependent/predicted variable

                string[][] rawTrain = new string[10][];
                rawTrain[0] = new string[] { "Detroit", "Home", "Large", "Win" };
                rawTrain[1] = new string[] { "Detroit", "Away", "Medium", "Win" };
                rawTrain[2] = new string[] { "Buffalo", "Home", "Small", "Win" };
                rawTrain[3] = new string[] { "Buffalo", "Home", "Medium", "Win" };
                rawTrain[4] = new string[] { "Atlanta", "Away", "Large", "Win" };
                rawTrain[5] = new string[] { "Chicago", "Home", "Medium", "Win" };
                rawTrain[6] = new string[] { "Chicago", "Away", "Small", "Lose" };
                rawTrain[7] = new string[] { "Chicago", "Home", "Small", "Lose" };
                rawTrain[8] = new string[] { "Atlanta", "Away", "Medium", "Lose" };
                rawTrain[9] = new string[] { "Detroit", "Away", "Large", "Lose" };

                Console.WriteLine("Raw (string) training data for team seattle:\n");
                Console.WriteLine("Opponent Field Spread Result");
                Console.WriteLine("-----");
                ShowMatrix(rawTrain);

                Console.WriteLine("\nConverting and storing training data");
                int[][] train = RawTrainToInt(rawTrain, values);
                Console.WriteLine("Training data in int form:\n");
                ShowMatrix(train, true);

                Console.WriteLine(
                    "\nCreating weak categorical stump learners from training data");
                List<Learner> learners = MakeLearners(values, train);

                Console.WriteLine("Completed. Weak learners are:\n");
                for (int i = 0; i < learners.Count; ++i)
                {
                    Console.WriteLine("[ " + i + " ] " + Description(learners[i],
                        features, values));
                }

                Console.WriteLine("\nInitializing list of best learner indexes");
                List<int> bestLearners = new List<int>(); // Indexes of good weak learners

                Console.WriteLine(
                    "\nUsing adaptive boosting to find best learners and alphas");
                MakeModel(train, values, learners, bestLearners);

                Console.WriteLine("\nModel completed");
                int numGood = bestLearners.Count;
                Console.WriteLine("Algorithm found " + numGood + " good learners ");
                Console.WriteLine("and associated alpha values");

                Console.WriteLine("\nThe good learners and their alpha value are:");
                for (int i = 0; i < bestLearners.Count; ++i)
                {
                    int lrn = bestLearners[i];
                    Console.WriteLine("[ " + lrn + " ] " +
                        learners[lrn].alpha.ToString("F2") + " ");
                }

                Console.WriteLine("\nPredicting outcome when Opponent = Detroit, ");
                Console.WriteLine("Field = Home, Spread = Small\n");
                int[] unknownTuple = new int[] { 3, 0, 0 }; // Detroit, Home, Small

                int Y = Classify(unknownTuple, learners, bestLearners);
                Console.WriteLine("Predicted Y = " + Y + " => ");
                Console.WriteLine("seattle will " + YValueToString(Y, values));

                Console.WriteLine("\nEnd\n");
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        } // Main

        // (Many) static methods here
    } // Class Program

    public class Learner // Weak learner
    {
        // Definition code here
    } // ns
```

The first step inside the loop is to update all epsilons. An epsilon value is the sum of the D weights of incorrectly classified training tuples. For learner [0] (IF Opponent IS Buffalo THEN Result IS Win) there are two applicable training tuples, [2] and [3], and the rule is correct in both instances, so epsilon is 0.00. For learner [1] (IF Opponent IS Chicago THEN Result IS Lose) there are three applicable training tuples, [5], [6] and [7]. Of these, tuple [5] is incorrect, so epsilon is just the D weight for tuple [5] = 0.10.

Although it's not immediately obvious, if you carefully review how epsilons are computed, you'll notice that epsilon values will always be between 0.0 and 0.5. After all updates, the learner epsilons are:

```
[0] (a = 0.00) (e = 0.00) IF Opponent IS Buffalo THEN Result IS Win
[1] (a = 0.00) (e = 0.10) IF Opponent IS Chicago THEN Result IS Lose
[2] (a = 0.00) (e = 0.10) IF Opponent IS Detroit THEN Result IS Win
[3] (a = 0.00) (e = 0.10) IF Field IS Home THEN Result IS Win
[4] (a = 0.00) (e = 0.20) IF Field IS Away THEN Result IS Lose
[5] (a = 0.00) (e = 0.10) IF Spread IS Small THEN Result IS Lose
[6] (a = 0.00) (e = 0.10) IF Spread IS Medium THEN Result IS Win
[7] (a = 0.00) (e = 0.10) IF Spread IS Large THEN Result IS Win
```

At this point the best learner is selected. It is learner [0] because its epsilon is smallest at 0.00. The associated alpha is computed as:

```
alpha = 0.5 * log((1.0 - epsilon) / epsilon)
```

This is essentially a magic equation from adaptive boosting theory. Here, log is the natural (base e) logarithm. Recall that the alpha value is a weight that assigns an importance to a learner. The previous equation is designed so that smaller values of epsilon (learner error) yield larger values of alpha (learner importance).

In this particular situation there's a problem because epsilon is 0, so there'd be a division by 0 error. To avoid this, the demo program arbitrarily converts any epsilon with a 0 value to 0.000001. So the alpha for learner [0] is computed as  $0.5 * \log(0.999999 / 0.000001) = 6.91$  and that value is assigned to learner [0], and learner [0] is flagged as completed.

The next step inside the algorithm loop is to update the D training tuple weights based on the just-computed best learner. The idea is to increase the D weights for those training tuples that are incorrectly classified by the best learner and decrease the D weights for training tuples that are correctly classified by the best learner. The D update equation is a bit tricky at first glance:

```
D(new) = D(old) * exp(-alpha * actualY * predictedY)
```

The best learner was learner [0] (IF Opponent IS Buffalo THEN Result IS Win) with an alpha of 6.91. Of the 10 training tuples, learner [0] applies to tuples [2] and [3], so those D values are updated. For training tuple [2]:

```
D(new) = 0.10 * exp(-6.91 * (+1) * (+1))
        = 0.10 * exp(-6.91)
        = 0.0000997758
```

The new D value for training tuple [3] has the same computation and same value as tuple [2].

In this case, we get a new D weight that's very small because the training tuple was correctly classified by the best learner. Notice that when the actual Y value and predicted Y value are the same (both are -1 or both are +1), when multiplied together the result is +1 and the argument to exp will be a negative number (because -alpha will always be negative), which will yield a result less than 1.0. However, if the actual Y and predicted Y are different, their product will be -1, and the argument to exp will be positive, which will yield a (possibly large) number greater than 1.0. This update technique for D is why adaptive boosting classification typically uses -1 and +1 for the dependent variable values rather than 0 and +1.

At this point the preliminary approximate D values (rounded) are:

```
[0] (D = 0.1000) Detroit      Home      Large      Win
[1] (D = 0.1000) Detroit      Away      Medium     Win
[2] (D = 0.0001) Buffalo      Home      Small      Win
[3] (D = 0.0001) Buffalo      Home      Medium     Win
[4] (D = 0.1000) Atlanta      Away      Large      Win
[5] (D = 0.1000) Chicago      Home      Medium     Win
[6] (D = 0.1000) Chicago      Away      Small      Lose
[7] (D = 0.1000) Chicago      Home      Small      Lose
[8] (D = 0.1000) Atlanta      Away      Medium     Lose
[9] (D = 0.1000) Detroit      Away      Large      Lose
```

The next step in the main algorithm loop is to normalize the D values so they sum to 1.0 by dividing each preliminary D value by the sum of the D values. The sum of the 10 D values is about 0.8002, so the normalized D for training tuple [0] is approximately  $0.1000 / 0.8002 = 0.1249$ . The final updated D weights are:

```
[0] (D = 0.1249) Detroit      Home      Large      Win
[1] (D = 0.1249) Detroit      Away      Medium     Win
[2] (D = 0.0001) Buffalo      Home      Small      Win
[3] (D = 0.0001) Buffalo      Home      Medium     Win
[4] (D = 0.1249) Atlanta      Away      Large      Win
[5] (D = 0.1249) Chicago      Home      Medium     Win
[6] (D = 0.1249) Chicago      Away      Small      Lose
[7] (D = 0.1249) Chicago      Home      Small      Lose
[8] (D = 0.1249) Atlanta      Away      Medium     Lose
[9] (D = 0.1249) Detroit      Away      Large      Lose
```

The idea here is that we want the algorithm to now focus on training tuples other than [2] and [3] because those tuples have been accounted for by learner [0]. At this point the algorithm jumps back to the top of the loop and updates all learners' epsilon values based on the newly computed D weights, determines the best unused learner (learner [5]), computes the alpha for the best learner (4.49), updates the D values for applicable training tuples ([2], [6] and [7]) and computes normalized D values for all training tuples.

In this example, the process continues until alpha values for all eight weak learners have been computed, but in general, not all weak learners will necessarily make the cut as good learners and be assigned an alpha value.

## Overall Program Structure

The demo program shown in **Figure 1** is a single C# console application. I used Visual Studio 2010, but any version that supports the Microsoft .NET Framework 2.0 or higher will work. I created a new project named AdaptiveBoosting and then in the Solution Explorer window renamed Program.cs to the more descriptive AdaptiveBoostingProgram.cs, which also automatically renamed class Program. I deleted all the template-generated using statements at the top of the source code except for the references to the System and Collections.Generic namespaces. The Main method—with some WriteLine statements removed, a few other minor edits and a key program-defined class to define weak learner objects—is listed in **Figure 2**.

The Main method begins by setting up hardcoded strings for "Opponent," "Field," "Spread" and "Result" for the features. Then the code in Main sets up hard-coded values for each feature: "Atlanta," "Buffalo," "Chicago," "Detroit," "Home," "Away," "Small," "Medium," "Large," "Lose" and "Win." To keep my output tidy, I used a hack and inserted a blank space at the end of "Small" and "Large."

For simplicity, the training data is also hardcoded into the demo program. In many situations your training data will be stored in a text file or a SQL table. In those situations you might want to consider programmatically scanning the training data to determine

the feature names (presumably from a text file header line or SQL table column names) and the feature values.

Method `RawTrainToInt` converts the training data in string form to zero-based integers and stores those integers into an `int[][]` matrix named `train`. `RawTrainToInt` calls a helper named `ValueToInt`. The `train` matrix has the dependent variable values (`Result`) stored in the last column. You may want to store dependent values in a separate column array. In situations with very large training data sets, you might not be able to store the entire training data set into machine memory. In those situations you'll have to stream through the external data store instead of an internal matrix.

The demo program determines the weak learners using method `MakeLearners` and a program-defined class `Learner`. I'll describe that method and class in detail in the next section of this article. After the weak learners have been created, the demo program calls method `MakeModel`. `MakeModel` is the heart of the adaptive boosting algorithm as described in the previous section. The net result is a sorted `List`, named `bestLearners`, of the indexes of the learners that were assigned alpha values.

The `Main` method finishes by predicting the outcome for Seattle for a set of inputs with Opponent of "Detroit," Field of "Home" and Spread of "Small," using method `Classify`. In this case, the return value from `Classify` is `-0.72`, which is interpreted as "Lose."

## Making the Weak Learners

The implementation of a specific adaptive boosting algorithm depends to some extent on the specific definition of a weak learner. Program-defined class `Learner` has six fields:

```
public int feature;  
public int value;  
public int predicted;  
public double error;  
public double epsilon;  
public double alpha;
```

I declared all five fields with public scope for simplicity. The `feature` field holds an integer that indicates which independent variable is the key to the learner. For example, if `feature` is 0, the weak learner is based on the value of an opponent. The `value` field holds an integer that indicates the value of the feature. For example, if `value` is 3, the weak learner is based on the condition opponent is Detroit. The `predicted` field is `-1` or `+1`, depending on whether the actual category for the feature value is Lose or Win.

The `error` field is type `double` and is the raw error rate associated with the weak learner on the training data. For example, if a weak learner has `feature` = 0, and `value` = 3, and `predicted` = `+1` (meaning if Opponent is Detroit then result is Win), then the raw error rate for the training data in **Figure 1** is 0.33 because one out of three training data items would be incorrectly predicted. Notice that raw error treats each training item equally. It turns out that the adaptive boosting algorithm presented in this article doesn't really need the raw error field, so that field could've been omitted, but I believe the information is useful.

The `epsilon` field is a weighted error term. The `epsilon` for a weak learner is an error term that takes into account the internal `D` weights assigned to each training item. The `epsilon` values are used by the adaptive boosting algorithm to compute the alpha weights. To summarize, there are two sets of weights used in adaptive boosting classification. The alpha weights assign an importance to

each weak learner and are used to determine an overall prediction. An epsilon error is an internal error associated with a weak learner that's used to compute the alpha weights. Each training tuple has an internal weight (given the name `D` in adaptive boosting literature) that's used to compute the epsilon errors.

In pseudo-code, method `MakeLearners` works like this:

```
initialize an empty result list of learners  
for each feature loop  
    for each value of curr feature loop  
        scan training data to determine most likely -1, +1 result  
        if no most likely result, skip curr value  
        create a new learner object with feature, value, predicted  
        add learner object to result list  
    end each value  
end each feature  
for each learner in result list  
    for each training data item  
        if learner isn't applicable to data, skip curr data item  
        compute raw error rate  
        store raw error into learner  
    end each training data item  
end each learner
```

The idea is that each feature value such as "Atlanta" (opponent) or "Medium" (point spread) will generate a weak learner rule based on the training data unless the value doesn't appear in the training data (for example, an opponent of "New York") or the value doesn't produce a most-likely result because there are the same number of wins and losses associated with the value (for example, when opponent is "Atlanta" in the demo data, with one win and one loss).

## Wrapping Up

An important variation on the algorithm presented in this article is dealing with data that has numeric values. For example, suppose that the values for the point spread feature, instead of being categorical "Small," "Medium" and "Large," were numeric, such as 1.5, 3.0 and 9.5. One of the major advantages of adaptive boosting classification compared with some other classification techniques is that adaptive boosting can easily handle both categorical and numeric data directly. You could create a dedicated learner class that has a friendly description similar to "if Spread is less than or equal to 5.5 then Result is Lose," or a more-complex learner along the lines of "if Spread is between 4.0 and 6.0 then Result is Win."

In my opinion, adaptive boosting classification is best used when the dependent variable to be predicted has just two possible values. However, advanced forms of adaptive boosting can deal with multinomial classification. If you wish to investigate this or the theory behind adaptive boosting in general, I recommend searching for articles by researchers R. Schapire and Y. Freund.

Machine-learning research suggests that there's no single best data classification/prediction technique. But adaptive boosting is a very powerful approach that can form the basis of adding predictive features to a .NET software system. ■

---

**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](mailto:jammc@microsoft.com).

---

**THANKS** to the following technical expert for reviewing this article:  
Darren Gehring (Microsoft)



# All these data sources at your fingertips – and that is just a start.



## RSSBus Data Providers [ADO.NET]

Build cutting-edge .NET applications that connect to any data source with ease.

- Easily “databind” to applications, databases, and services using standard Visual Studio wizards.
- Comprehensive support for CRUD (Create, Read, Update, and Delete operations).
- Industry standard ADO.NET Data Provider, fully integrated with Visual Studio.

## Databind to the Web...

The RSSBus Data Providers give your .NET applications the power to databind (just like SQL) to Amazon, PayPal, eBay, QuickBooks, FedEx, Salesforce, MS-CRM, Twitter, SharePoint, Windows Azure, and much more! Leverage your existing knowledge to deliver cutting-edge WinForms, ASP.NET, and Windows Mobile solutions with full readwrite functionality quickly and easily.

## Databind to Local Apps...

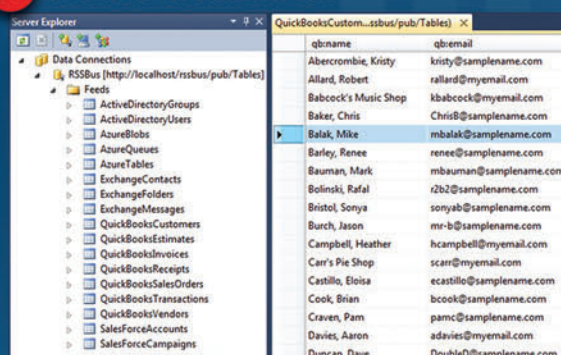
The RSSBus Data Providers make everything look like a SQL table, even local application data. Using the RSSBus Data Providers your .NET applications interact with local applications, databases, and services in the same way you work with SQL Tables and Stored Procedures. No code required. It simply doesn't get any easier!

*“Databind to anything...  
...just like you do with SQL”*

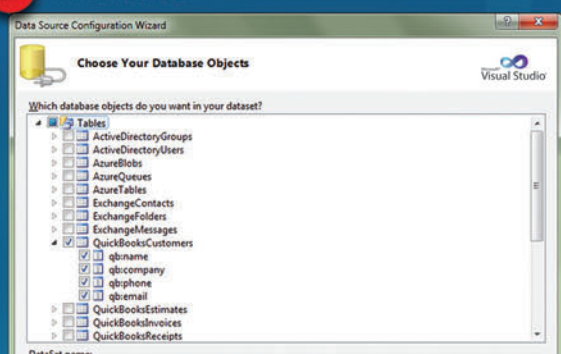
Also available for:

JDBC | ODBC | SQL SSIS | Excel | OData | SharePoint ...

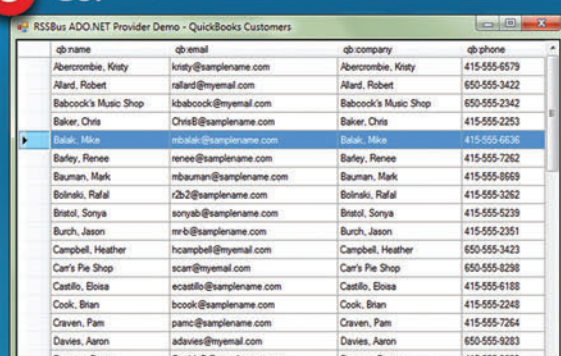
### 1 SELECT CONNECTOR



### 2 DATABIND



### 3 GO!





YOUR BACKSTAGE PASS TO THE MICROSOFT PLATFORM



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

# ROCK YOUR CODE ON CAMPUS!

Celebrating 20 years of education and training for the developer community, Visual Studio Live! is back on the Microsoft Campus – backstage passes in hand! Over 5 days and 60+ sessions and workshops, you'll get an all-access look at the Microsoft Platform and practical, unbiased, developer training at Visual Studio Live! Redmond.



# REDMOND, WA | AUGUST 19-23, 2013

MICROSOFT CAMPUS



**REGISTER  
TODAY  
AND  
SAVE \$400**

USE PROMO CODE REDAPR2



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

## TOPICS WILL INCLUDE:

- ASP.NET
- Azure / Cloud Computing
- Cross-Platform Mobile
- Data Management
- HTML5 / JavaScript
- SharePoint / Office
- Windows 8 / WinRT
- WPF / Silverlight
- Visual Studio 2012 / .NET 4.5

## CONNECT WITH VISUAL STUDIO LIVE!



[twitter.com/vslive](https://twitter.com/vslive) – @VSLive



[facebook.com](https://facebook.com) – Search "VSLive"



[linkedin.com](https://linkedin.com) – Join the "VSLive" group!

[vslive.com/redmond](https://vslive.com/redmond)





## Capturing Important Business Logic

Given that this is the month of April, that the magazine arrived on your doorstep (or mailbox or e-mail inbox or ... whatever) somewhere on or around April 1, and that said date is often referred to in the United States and other Western nations as “April Fool’s Day,” it seemed appropriate to do a column on languages in which we can capture Important Business Logic. (Note the capitals there; that’s how business people always refer to whatever ridiculous new feature they want, so it’s only appropriate that we refer to it with the same degree of veneration and respect that it deserves.) It’s important to note it’s of utmost importance that when we capture said Important Business Logic, we do so with gravitas and sincerity; this means doing so in a language that the business owners and customer representatives (and most of the marketing department) can understand.

So, it’s with a grave and sincere demeanor that I present to you an important language in your .NET programming toolbox: LOLCODE.

### LOLCODE

While it has long been the opinion of scientists and dog owners alike that cats can’t really communicate with their owners, cat owners have known for years that they communicate just fine—just ask any cat’s co-residing human whose cat has just informed him that despite the fact it’s 5 a.m., it’s time for the cat to be fed. (For the uninitiated, such activity usually involves loud meows, head butts to the nose, innocent expressions and occasionally a very slight flexing of a feline claw to a sensitive human body part.)

However, ever since cats discovered the mechanical mouse, a quiet revolution has been taking place across the Internet: Cats have steadily improved their ability to communicate by posting pictures of themselves with captions written in what many anthropologists and historians (all of whom are cat lovers—the dog-loving anthropologists and historians having been “unavailable” to comment on this) agree is the fundamental building blocks of all human language. This fundamental language is sometimes referred to imperfectly by humans by the name “LOLspeak.” (Examples of it can be found in your Facebook and Twitter feeds, so don’t pretend you don’t know what I’m referring to.)

Fortunately, well-intentioned cat-owned engineers familiar with the .NET platform have taken to extending this language into the .NET platform; two such endeavors were, in fact, in place, but sadly only one has survived to this day. The first was an implementation based on the Dynamic Language Runtime (DLR) from Martin Maly and John Lam back when the DLR was new (2007), but it seems to have disappeared. The second is a project from around the same time frame, but compiling the code into .NET assemblies rather than interpreting the source directly. That project is hosted at [bit.ly/AeM](http://bit.ly/AeM) and, being open source, is easily accessible through Subversion

via a single “svn checkout <http://lolcode-dot-net.googlecode.com/svn/trunk/>” command at your local command-line window. Once built, we can start to explore this amazing and powerful language.

For full details on the LOLCODE language, check out the LOLCODE Web site at [lolcode.com](http://lolcode.com), and in particular check out the LOLCODE language specifications, all of which are European Cat Programmers’ Association (ECPA) standards, at [lolcode.com/specs/1.2](http://lolcode.com/specs/1.2). LOLCODE.NET implements much (if not all) of LOLCODE 1.2, which is the latest LOLCODE standard.

### HAI

Of course, the first program that anyone ever writes in any new language must be that language’s version of “Hello World,” and our dive into LOLCODE will be no different:

```
HAI
BTW Greet the people
VISIBLE "Hello, world!"
KTHXBYE
```

Every LOLCODE program begins with the traditional greeting, “HAI,” and is terminated with the traditional closing, “KTHXBYE.” As with all keywords in LOLCODE, these are case-sensitive and must be typed all uppercase. “VISIBLE” is, as might be inferred, the command to write to the standard output stream, and “BTW” is the single-line comment. (Multi-line comments open with “OBTW” and end with “TLDR.”)

Using the “lolc” compiler built from the LOLCODE.NET source tree gives us a traditional .NET assembly, named (in the traditional fashion of .NET compilers) after the source file, so if the previous code is stored in “hello.lol,” then “hello.exe” contains the Intermediate Language (IL).

### I HAS YARN

LOLCODE recognizes that there are only four types in the world: strings (YARN), numbers (NUMBR), Booleans (TROOF) and arrays (BUKKIT), but because cats never commit to anything until the last possible moment, a variable declaration in LOLCODE (using the “I HAS A” syntax) is untyped until the value is assigned to it. More important, because a cat would never be held accountable to a decision even once the decision is made, variables in LOLCODE are flexibly typed, meaning that a variable can hold any value, changing its type as necessary to reflect the new value:

```
I HAS A COOLVAR ITZ THREE
LOL COOLVAR R 3
```

The first line of code declares a variable named “COOLVAR” and assigns it the string (YARN) value THREE. The second line of code sets that same variable to the value 3, of NUMBR type. Any uninitialized variable contains the null value, “NOOB.”

Like keywords, variables are case-sensitive and may use mixed-case, though idiomatic LOLCODE suggests all capitals for clarity. Like C#, variable names must begin with a letter, but can then consist of letters, numbers or underscores.

Reading from the console is done using the “GIMMEH” command, so a second program to greet the user, ask him his name and print it back to him looks like the following:

```
HAI
I HAS A NAME ITZ "Ted"
VISIBLE "d00d type in ur name"
GIMMEH NAME
VISIBLE "d00d ur name is " NAME
KTHXBYE
```

While the l33tspeak in the console isn't a required part of the language, it's considered idiomatically correct and good form.

## Flow Control

If/then constructs are created by placing “O RLY?” after an expression, using “YA RLY” to define the truth branch of the expression and “NO WAI” for the false branch of the expression, and “OIC” to indicate the end of the if/then construct entirely. Multiple statements can be comma-separated on a single line if this is aesthetically pleasing. If there are more than two possible branches (what other, less feline-friendly languages call an “else if”), then this is given by “MEBBE” blocks defined in between the “YA RLY” and “NO WAI” blocks. Comparison operations are fully listed in the LOLCODE 1.2 specification, but the “SAEM” (equality) and “DIFFRINT” (inequality) operators are likely to be the most commonly used comparisons, as is true of most modern programming languages. For Boolean operations, “BOTH OF <x> AN <y>” give us logical-AND, “EITHER OF <x> [AN] <y>” give us logical-OR, and “WON OF <x> [AN] <y>” give us logical-XOR.

Loops are signified using the “IM IN YR <label>” keyword to begin the loop construct and end with the “IM OUTTA YR <label>” construct; without a terminating condition at the start of the loop construct, given by a “TIL” or “WILE” expression, the loop will run infinitely until terminated by a “GTFO” expression, which returns control immediately after the “IM OUTTA YR” keyword.

Putting all of this together, for example, you have the following program:

```
HAI
I HAS A NAME ITZ ""
IM IN YR LOOP
  VISIBLE "Gimme ur name or 'gtfo' to GTFO"
  GIMMEH NAME
  SAEM NAME AN "gtfo", O RLY?
    YA RLY, VISIBLE "L8r!", GTFO
  NO WAI, VISIBLE "Yo " NAME
OIC
IM OUTTA YR LOOP
KTHXBYE
```

Notice that the loop “LOOP” is an infinite loop, with no termination condition, using “GTFO” in the “YA RLY” branch of the “SAEM ... O RLY?” conditional to terminate the program in the event the user types in “gtfo” as input.

## HOW DUZ I ...

Good programming practice demands that code be segregated into easily consumable units, and LOLCODE wouldn't be the serious business-ready language that it is without similar capability. By

using the “HOW DUZ I” and “IF U SAY SO” keyword pairs, you can define functions that can be invoked to carry out operations:

```
HAI

HOW DUZ I GREET
  I HAS A NAME ITZ "Ted"
  VISIBLE "d00d type in ur name"
  GIMMEH NAME
  VISIBLE "d00d ur name is " NAME
IF U SAY SO

GREET

KTHXBYE
```

This will define a single function, “GREET,” that's then used from the main program to greet the user and echo back his name. Parameters to the function are given by “YR”-name pairs.

## I CAN HAZ LIBRARY?

Of course, the real advantage of LOLCODE isn't in writing the boring and tedious parts of the program (a real cat would never stoop to boring and tedious, of course); instead, LOLCODE's true advantage lies in capturing the Important Business Logic and exposing it through a Web service or Windows Presentation Foundation (WPF) GUI or something. For this reason, the LOLCODE compiler also supports the ubiquitous “/target:library” option to produce a standard .NET library assembly—doing this, however, still produces a “.exe”-suffixed assembly, with a Main method present in the program, even though the Main simply does nothing. This is actually quite in keeping with other modern languages that assume that the top of the file is the entry point to the program as a whole; LOLCODE is simply taking the extra step of always providing a Main, because one (even though empty) is always present.

Whether compiling as a library or an executable, LOLCODE always wraps the generated code into a class called “Program.” There is no namespace functionality—no self-respecting feline would ever accept a last name, either.

## Soon to Be a Mission-Critical Best Practice

While LOLCODE is powerful, unfortunately its acceptance has not yet reached a point where many business analysts and customer representatives understand the benefits of expressing business logic in it. Fortunately, being open source, extensions to the language are easy, and this language is expected to become an industry-standard mission-critical best practice by 2014. For best results, architects are encouraged to demonstrate the power of LOLCODE in hands-on coding sessions, preferably in a meeting with high-level executives with no forewarning or advance notice. Happy coding! ■

---

**TED NEWARD** is totally making all of this up, except for the parts about the LOLCODE language—all of that is absolutely true. He has written more than 100 articles and authored or coauthored a dozen books, including “Professional F# 2.0” (Wrox, 2010), but none as a joke like this article is. He is an F# MVP and speaks at conferences around the world. He has been advised by counsel not to publish any means of reaching him, so as to avoid angry letters from architects who followed that last bit of advice, but longtime readers of this column will already know how to reach him.

---

**THANKS** to the following technical experts in the Neward family for reviewing this article: Scooter, Shadow, Princess and Mr. Fluff Fluff



# Power Your Modern Apps with Windows Azure Mobile Services

Regardless of the platform you've targeted for app publishing, Windows Azure Mobile Services (WAMS) is one back end to rule them all. It's a key component of the Windows Azure platform and it's *the* back end for cross-platform modern app and Web development. In this article I'll cover setting up WAMS to work in Visual Studio, provide an overview of the WAMS API and explain the coding necessary to build a modern app with WAMS.

## The Cross-Platform Architecture of WAMS

There are many Windows Azure services you can use for managing data and powering an app's back end, and it isn't uncommon for apps to consume more than one Windows Azure service. Here are some of the available services:

- **WAMS:** A cross-platform, full-featured set of back-end services and resources geared specifically for fast app building.
- **SQL Azure:** This is the same popular SQL Server, but in the cloud, with an easy-to-use Web admin interface. It's cost-effective for smaller startups, companies and ISVs.
- **Windows Azure Table Storage:** A NoSQL way to work with tabular and sometimes-not-so-tabular data.
- **Windows Azure Binary Large Object (BLOB) Storage:** A highly scalable way to contain data in key/value pairs in the cloud without the worries and restrictions of structured data.
- **Windows Azure Web Sites:** In addition to Web site hosting, Windows Azure Web Sites can run ASP.NET and ASP.NET Web APIs. This is a great way to power legacy Web sites, programs and apps over HTTP without having to do much architectural rework.

An app's architecture will depend on its requirements. If the app needs to store large amounts (for example, tens or hundreds of gigabytes of data) of media or binary content, using Windows Azure BLOB Storage is likely a better fit. For most apps that just read and write textual data with some accompanying pictures, WAMS is a straightforward and easy solution. Many apps must deal with legacy data, so migrating from SQL Server, SQL Server Compact (SQL CE) or any of the Microsoft SQL family of databases straight to SQL Azure might be the best route if you need a DBA to administer the data.

For most apps, WAMS is quite suitable because not only does it have data storage, but also a full set of back-end services on top of the database created for the express purposes of supporting common app infrastructure scenarios like push notifications and authentication.

## WAMS Core Services

The following features are core WAMS services:

- **Data:** Of course, housing and manipulating data is essential to every app, so WAMS supports it. For each mobile service, there's a SQL Azure database behind it.
- **Messaging:** Push notifications are increasingly important because users want to stay up-to-date. As apps get progressively more complex, intuitive and user-friendly—in other words, more modern—features such as push messaging and real-time communication become commonplace. Fortunately, using push notifications in WAMS is as easy as a call to send a toast (a small pop-up message in the top- or bottom-right of the OS) notification like so:

```
push.wns.sendToastText04(channel, { text1: item.text });
```

- **Authentication:** Securing data, especially user data, is as important as the data itself. A cornerstone of modern apps is authenticating via widely popular Web sites—such as Facebook or Twitter—so WAMS allows you to authenticate with any of the following identity providers:
  1. Microsoft Account (the authentication provider formerly known as Windows Live ID)
  2. Facebook login
  3. Twitter login
  4. Google login

For most apps that just read and write textual data with some accompanying pictures, WAMS is a straightforward and easy solution.

WAMS sports several libraries for the Windows family of app development, including the Windows Runtime (WinRT) library for managed clients, the Windows Library for JavaScript (WinJS) client library and Representational State Transfer (REST) APIs for everything—core services, authentication and messaging. There are even iOS and Android client libraries for cross-platform parity.



Figure 1 The Person Class

```
// Data model/class code.
public class Person
{
    public int Id { get; set; }

    public string Name { get; set; }

    [DataMember(Name = "Birthday")]
    public DateTime Date { get; set; }

    public string Picture { get; set; }

    public string Notes { get; set; }
}
```

## Use the WAMS API in Windows Store or Windows Phone Projects

Whether you're building a Windows Store or Windows Phone app, the code will be almost identical, and the JavaScript code is remarkably similar on the WinJS side. Once you have the proper development tools and WAMS set up and configured (see [bit.ly/NAAQz8](http://bit.ly/NAAQz8) for more on this), writing the code to access objects and data in a mobile service is easy, but different from traditional client/server or *n*-tier app development. This is because WAMS integrates with the app in such a way that certain changes in the app's model can cause changes in the underlying data structures. In other words, you can throw out many of the traditional SQL data definition statements and simply modify a class's member, recompile, run the app and verify the changes at the database level. In addition to this, JavaScript takes the place of SQL as the syntax du jour for database maintenance such as validation scripts or creating constraints on the data. Being able to easily script database changes from the app, command prompt and Web administration tool shortens the app-building process.

The Microsoft.WindowsAzure.MobileServices namespace provides standard-issue access to objects in the mobile service or its underlying SQL Azure database. The following code sample creates an instance of MobileServiceClient:

```
public static MobileServiceClient MobileService = new MobileServiceClient(
    "https://your-very-own-service-url.azure-mobile.net/",
    "your-very-own-key-that-is-a-gigantic-string-of-characters-and-numbers"
);
```

As you can see, your WAMS app URL and key are parameters of the MobileServiceClient constructor. To find your URL and key, log in to the Windows Azure portal and navigate to your mobile service, then click the cloud icon to the left of the dashboard menu item.

In the preceding code sample, the MobileServiceClient class behaves somewhat like a connection object, but without those pesky open and close methods from the era of the SqlConnection object. The MobileServiceClient class manages the connectivity

for you. If your data is set for public consumption, then you don't need to call the MobileServiceClient.login method, yet you can access tables, run queries and sort data. Of course, more complex or security-conscious operations may require credentials.

Once you have a valid instance of a MobileServiceClient class, you can then use GetTable<T> to interact with an underlying WAMS table:

```
private IMobileServiceTable<Person> personsTable =
    MobileService.GetTable<Person>();
```

The type parameter T in IMobileServiceTable<T> causes the compiler to inspect the properties and information about the parameter—in this case the Person class in **Figure 1**—by mapping it to the underlying table at the database level. This allows you to add or modify properties during development and see the schema changes reflected in the cloud at run time, resulting in a more symbiotic alliance between the code and the database. However, schema changes at the database level don't happen automatically for every change in code. For example, adding a property to the code in **Figure 1** creates a new column in the table, but deleting an existing property doesn't cause a deletion.

GetTable<T> returns an IMobileServiceTable object that represents the actual underlying table in WAMS and contains methods for inserting, updating, deleting and sorting services, as shown here:

```
// Perform delete asynchronously where item is of type Person, see Figure 1.
await itemsTable.DeleteAsync(item);
// Select specific records with LINQ query
var people =
    personsTable.Select(p => p.Birthday > DateTime.Now.AddDays(14));
```

You can build your data layer in the code by creating a custom class that matches the table and member names. This technique is known as convention over configuration, which is a way to cut down on bloated XML configuration code in favor of continuity and consistency in naming, resulting in less code that's more maintainable. For example, the class and schema in **Figure 1** demonstrate that the client class code maps to a WAMS table and its members.

The code in **Figure 1** creates the database schema shown in **Figure 2**.

WAMS infers data types from the property's data type in code. Some types, however, can't be represented with data attributes, specifically image and binary data. This means that when you want to store an image, you should use Windows Azure BLOB storage because it's more scalable, cheaper and performs much better than the alternative, which is Base64 encoding to a string and then storing it in a WAMS table. Windows Azure BLOB storage has a REST API and managed API so developers can easily access the services across platforms and preferences.

To perform insert, update or delete operations, just make an instance of your custom object and call the corresponding method from your IMobileServiceTable object:

```
Person person = new Person {
    Name = "Alan Turing",
    Birthday = DateTime dte = new DateTime(1912,6,23),
    Picture = base64string, // Image encoding done elsewhere in code.
    Notes = "A father of modern computer science. There is a test " +
        "named after him that I fail regularly";
await personsTable.InsertAsync(person);
```

Because the call to InsertAsync is clearly asynchronous, the call doesn't block UI code from running, and you can manage data in the background without interfering with the user's activities. Windows Store and Windows Phone apps work in an asynchronous fashion by default, as you can't always count on reliable connectivity.

Figure 2 Convention over Configuration  
Matches Class Code and Database Schema Names

Column	Type
Id	SQL Bigint
Name	String
Date	DateTime
Picture	String
Notes	String

If you're working in XAML, you can data bind the table by calling the `IMobileServiceTable` object's `ToListAsync` method. This method preps the data and returns it into an object that binds WAMS tables easily to XAML `ListView` elements, as the following code sample demonstrates:

```
var results = await todoTable.ToListAsync();
items = new ObservableCollection<TodoItem>(results);
ListItems.ItemsSource = items;
```

The preceding code is similar to traditional data-binding code in the Microsoft .NET Framework. You can also call methods on the `IMobileServiceTable` object to return an `IList<T>` or `IEnumerable<T>` if you wish to manually loop through the data, rather than data bind.

## Managing Advanced WAMS Features

You will, of course, need to administer and maintain the mobile service, its data, security and all the usual tasks that go with back-end maintenance, as not all of that kind of work should be done in the app. Fortunately, WAMS gives you a few choices in regard to back-end administrative tasks:

- **Command-line tool:** This is great no matter what platform you use for development; you can use any command prompt once you've downloaded and installed the command-line libraries (see [bit.ly/14Q49bi](http://bit.ly/14Q49bi) for more on this).
- **Web administration:** Great for cross-platform back-end development, and also known as the Windows Azure portal, this tool lets you do all the basics online—create databases, run server-side scripts, manage services, manage security and so on (the portal is accessed from the main Windows Azure site at [bit.ly/4yqVhP](http://bit.ly/4yqVhP)).
- **SQL Server Management Studio:** A classic Microsoft database administration tool that you can use to connect and manage the databases behind a mobile service (see [bit.ly/VdqpZH](http://bit.ly/VdqpZH) for more on this).

Working with WAMS requires  
no hefty software installs on the  
client because of its powerful  
Web administration and  
command-line tools.

WAMS contains a succinct yet complete WAMS server-side script reference ([bit.ly/XvsVec](http://bit.ly/XvsVec)). You can launch commands from the command-line tool or from the Windows Azure portal online. At the command prompt, simply enter the command “azure mobile” to see a list of all available commands and “azure mobile list” to list all your mobile services.

Just like your app code, server-side script functions are “registered” to a table via naming conventions, so the scripts must match method signatures for insert, read, update and delete operations. Here are the data-manipulation script signatures in JavaScript:

- **Insert function:** `insert (item, user, request) { ... }`
- **Read function:** `read (query, user, request) { ... }`
- **Update function:** `update (item, user, request) { ... }`
- **Delete function:** `del (id, user, request) { ... }`

Notice that everything you need is included as method parameters, that is, the data row, the user identity and the request itself. This allows you to perform row-level security as well as run any type of server-side rules such as data validation when these activities occur.

As an example, you might want to perform validation to limit the incoming string's length for insertions. You can do so by using this code:

```
function insert(item, user, request) {
    if (item.text.length > 20) {
        request.respond(statusCodes.BAD_REQUEST,
            'The length of the input text must be less than 20');
    } else {
        request.execute();
    }
}
```

While you can modify these scripts at the Windows Azure portal, you can also do so in your favorite text editor and save the file with a .js extension, for example, `person.insert.js`. You can then upload the prepared script via the command line with a command with this syntax and signature:

```
azure mobile script upload <service-name> table/<table-name>.<operation>.js
```

A command to upload the previous sample script would look something like this (where “Notable” is the service name and “person” is the table name):

```
azure mobile script upload NotablePeople table/person.insert.js
```

As you might expect, the results of a validation error can be caught by a try/catch statement in any client-side language.

Alas, SQL developers might be feeling out of touch at this point, with all the server-side JavaScript trends lately, but they can feel more at home by using SQL inside an mssql script object, even if it's mixed with JavaScript:

```
mssql.query('select * from people', {
    success: function(results) {
        console.log(results);
    }
});
```

This is especially helpful if you come from an enterprise development background that's full of SQL, as it's much different than the dynamic nature of JavaScript.

To review, WAMS is a complete set of back-end services for rapid, cross-platform app building. All of the Windows Azure services except SQL Azure automatically implement REST-friendly URLs, and because REST is cross-platform, there's no need to worry about ensuring your app architecture will support new client devices or user agents in the future. In the meantime, working with WAMS requires no hefty software installs on the client because of its powerful Web administration and command-line tools, as well as easy-to-use APIs for everything from basic data storage to bulk e-mailing and push notifications. ■

---

**RACHEL APPEL** is a developer evangelist at Microsoft New York City. Reach her via her Web site at [rachelappel.com](http://rachelappel.com) or by e-mail at [rachel.appel@microsoft.com](mailto:rachel.appel@microsoft.com). You can also follow her latest updates on Twitter at [twitter.com/rachelappel](https://twitter.com/rachelappel).

---

**THANKS** to the following technical expert for reviewing this article:  
*Paul Batum (Microsoft)*

# HTML5+jQuery

Any App - Any Browser - Any Platform - Any Device



**IGNITEUI**<sup>TM</sup>  
INFRAGISTICS JQUERY CONTROLS



Download Your **Free Trial!**  
[www.infragistics.com/igniteui-trial](http://www.infragistics.com/igniteui-trial)



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

Copyright 1996-2013 Infragistics, Inc. All rights reserved. Infragistics and Infragistics 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.





# Streaming and Manipulating Audio Files in Windows 8

Many Windows users these days have a Music Library on their hard drives containing perhaps thousands or tens of thousands of MP3 and WMA files. To play this music on the PC, generally such users run the Windows Media Player or the Windows 8 Music application. But for programmers, it's good to know we can write our own programs to play these files. Windows 8 provides programming interfaces to access the Music Library, obtain information about the individual music files (such as artist, title and duration), and play these files using `MediaElement`.

`MediaElement` is the easy approach, and of course there are alternatives that make the job more difficult but also add a lot of versatility. With two DirectX components—`Media Foundation` and `XAudio2`—it's possible for an application to get much more involved in this process. You can load chunks of decompressed audio data from music files, and analyze that data or manipulate it some way before (or instead of) playing the music. Have you ever wondered what a Chopin Étude sounds like when played backward at half speed? Well, neither have I, but one of the programs accompanying this article will let you find out.

## Pickers and Bulk Access

Certainly the easiest way for a Windows 8 program to access the Music Library is through the `FileOpenPicker`, which can be initialized in a C++ program for loading audio files like this:

```
FileOpenPicker^ fileOpenPicker = ref new FileOpenPicker();
fileOpenPicker->SuggestedStartLocation = PickerLocationId::MusicLibrary;
fileOpenPicker->FileTypeFilter->Append(".wma");
fileOpenPicker->FileTypeFilter->Append(".mp3");
fileOpenPicker->FileTypeFilter->Append(".wav");
```

Call `PickSingleFileAsync` to display the `FileOpenPicker` and let the user select a file.

Code download available at [archive.msdn.microsoft.com/mag201304DXF](http://archive.msdn.microsoft.com/mag201304DXF).

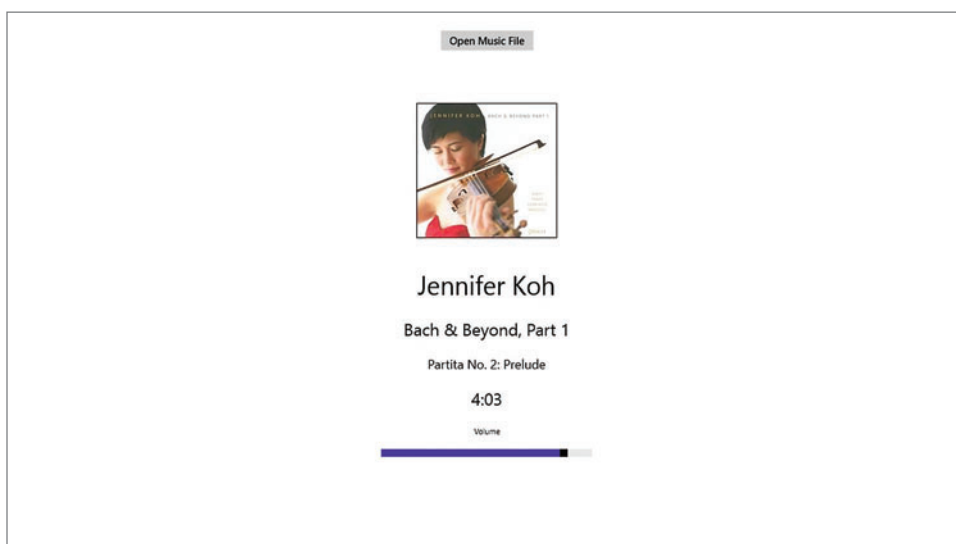


Figure 1 The `StreamMusicFile` Program Playing a Music File

For a free-form exploration of the folders and files, it's also possible for the application manifest file to indicate it wants more extensive access to the Music Library. The program can then use the classes in the `Windows::Storage::BulkAccess` namespace to enumerate the folders and music files on its own.

Regardless of the approach you take, each file is represented by a `StorageFile` object. From that object you can obtain a thumbnail, which is an image of the music's album cover (if it exists). From the `Properties` property of `StorageFile`, you can obtain a `MusicProperties` object, which provides the artist, album, track name, duration and other standard information associated with the music file.

By calling `OpenAsync` on that `StorageFile`, you can also open it for reading and obtain an `IRandomAccessStream` object, and even read the entire file into memory. If it's a WAV file, you might consider parsing the file, extracting the waveform data and playing the sound through `XAudio2`, as I've described in recent installments of this column.

But if it's an MP3 or WMA file, that's not so easy. You'll need to decompress the audio data, and that's a job you'll probably not want to take on yourself. Fortunately, the `Media Foundation` APIs include facilities to decompress MP3 and WMA files and put the data into a form that can be passed directly to `XAudio2` for playing.

Another approach to getting access to decompressed audio data is through an audio effect that's attached to a MediaElement. I hope to demonstrate this technique in a later article.

## Media Foundation Streaming

To use the Media Foundation functions and interfaces I'll be discussing here, you'll need to link your Windows 8 program with the mfplat.lib and mfreadwrite.lib import libraries, and you'll need #include statements for mfapi.h, mfidl.h and mfreadwrite.h in your pch.h file. (Also, be sure to include initguid.h before mfapi.h or you'll get link errors that might leave you baffled for many unproductive hours.) If you're also using XAudio2 to play the files (as I'll be doing here), you'll need the xaudio2.lib import library and xaudio2.h header file.

Among the downloadable code for this column is a Windows 8 project named StreamMusicFile that demonstrates pretty much the minimum code necessary to load a file from the PC's Music Library, decompress it through Media Foundation and play it through XAudio2. A button invokes the FileOpenPicker, and after you've selected a file, the program displays some standard information (as shown in **Figure 1**) and immediately begins playing the file. By default, the volume Slider at the bottom is set to 0, so you'll need to increase that to hear anything. There's no way to pause or stop the file except by terminating the program or bringing another program to the foreground.

In fact, the program doesn't stop playing a music file even if you click the button and load in a second file. Instead, you'll discover that both files play at the same time, but probably not in any type of coherent synchronization. So that's something this program can do that the Windows 8 Music application and Media Player cannot: play multiple music files simultaneously!

Figure 2 Creating and Initializing an IMFSourceReader

```
ComPtr<IMFSourceReader> MainPage::CreateSourceReader(IRandomAccessStream^
randomAccessStream)
{
    // Start up Media Foundation
    HRESULT hresult = MFStartup(MF_VERSION);

    // Create a IMFByteStream to wrap the IRandomAccessStream
    ComPtr<IMFByteStream> mfByteStream;
    hresult = MFCreateMFByteStreamOnStreamEx((IUnknown *)randomAccessStream,
&mfByteStream);

    // Create an attribute for low latency operation
    ComPtr<IMFAttributes> mfAttributes;
    hresult = MFCreateAttributes(&mfAttributes, 1);
    hresult = mfAttributes->SetUINT32(MF_LOW_LATENCY, TRUE);

    // Create the IMFSourceReader
    ComPtr<IMFSourceReader> mfSourceReader;
    hresult = MFCreateSourceReaderFromByteStream(mfByteStream.Get(),
mfAttributes.Get(),
&mfSourceReader);

    // Create an IMFMediaType for setting the desired format
    ComPtr<IMFMediaType> mfMediaType;
    hresult = MFCreateMediaType(&mfMediaType);
    hresult = mfMediaType->SetGUID(MF_MT_MAJOR_TYPE, MFMediaType_Audio);
    hresult = mfMediaType->SetGUID(MF_MT_SUBTYPE, MFAudioFormat_Float);

    // Set the media type in the source reader
    hresult = mfSourceReader->SetCurrentMediaType(MF_SOURCE_READER_FIRST_AUDIO_STREAM,
0, mfMediaType.Get());

    return mfSourceReader;
}
```

The method shown in **Figure 2** shows how the program uses an IRandomAccessStream from a StorageFile to create an IMFSourceReader object capable of reading an audio file and delivering chunks of uncompressed audio data.

For clarity, **Figure 2** excludes all code dealing with errant HRESULT return values. The actual code throws exceptions of type COMException, but the program doesn't catch these exceptions as a real application would.

MediaElement is the  
easy approach, and of course  
there are alternatives that make  
the job more difficult but also  
add a lot of versatility.

In short, this method uses the IRandomAccessStream to create an IMFByteStream object encapsulating the input stream, and then uses that to create an IMFSourceReader, which can perform the actual decompression.

Notice the use of an IMFAttributes object to specify a low-latency operation. This isn't strictly required, and you can set the second argument to the MFCreateSourceReaderFromByteStream function to nullptr. However, as the file is being read and played, the hard drive is being accessed and you don't want those disk operations to create audible gaps in the playback. If you're *really* nervous

Figure 3 The AudioFilePlayer Constructor in StreamMusicFile

```
AudioFilePlayer::AudioFilePlayer(ComPtr<IXAudio2> pXAudio2,
ComPtr<IMFSourceReader> mfSourceReader)
{
    this->mfSourceReader = mfSourceReader;

    // Get the Media Foundation media type
    ComPtr<IMFMediaType> mfMediaType;
    HRESULT hresult = mfSourceReader->GetCurrentMediaType(MF_SOURCE_READER_
FIRST_AUDIO_STREAM,
&mfMediaType);

    // Create a WAVEFORMATEX from the media type
    WAVEFORMATEX* pWaveFormat;
    unsigned int waveFormatLength;
    hresult = MFCreateWaveFormatExFromMFMediaType(mfMediaType.Get(),
&pWaveFormat,
&waveFormatLength);

    // Create the XAudio2 source voice
    hresult = pXAudio2->CreateSourceVoice(&pSourceVoice, pWaveFormat,
XAUDIO2_VOICE_NOPITCH, 1.0f, this);

    // Free the memory allocated by function
    CoTaskMemFree(pWaveFormat);

    // Submit two buffers
    SubmitBuffer();
    SubmitBuffer();

    // Start the voice playing
    pSourceVoice->Start();
    endOfFile = false;
}
```

about this problem, you might consider reading the entire file into an `InMemoryRandomAccessStream` object and using that for creating the `IMFByteStream`.

When a program uses Media Foundation to decompress an audio file, the program has no control over the sampling rate of the uncompressed data it receives from the file, or the number of channels. This is governed by the file. However, the program can specify that the samples be in one of two different formats: 16-bit integers (used for CD audio) or 32-bit floating-point values (the C

float type). Internally, XAudio2 uses 32-bit floating-point samples, so fewer internal conversions are required if 32-bit floating-point samples are passed to XAudio2 for playing the file. I decided to go that route in this program. Accordingly, the method in **Figure 2** specifies the format of the audio data it desires with the two identifiers `MFMediaType_Audio` and `MFAudioFormat_Float`. If decompressed data is required, the only alternative to this second identifier is `MFAudioFormat_PCM` for 16-bit integer samples.

At this point, we have an object of type `IMFSourceReader` poised to read and decompress chunks of an audio file.

**Figure 4 The Audio-Streaming Pipeline in `StreamMusicFile`**

```
void AudioFilePlayer::SubmitBuffer()
{
    // Get the next block of audio data
    int audioBufferLength;
    byte * pAudioBuffer = GetNextBlock(&audioBufferLength);

    if (pAudioBuffer != nullptr)
    {
        // Create an XAUDIO2_BUFFER for submitting audio data
        XAUDIO2_BUFFER buffer = {0};
        buffer.AudioBytes = audioBufferLength;
        buffer.pAudioData = pAudioBuffer;
        buffer.pContext = pAudioBuffer;
        HRESULT hresult = pSourceVoice->SubmitSourceBuffer(&buffer);
    }
}

byte * AudioFilePlayer::GetNextBlock(int * pAudioBufferLength)
{
    // Get an IMFSourceReader object
    ComPtr<IMFSample> mfSample;
    DWORD flags = 0;
    HRESULT hresult = mfSourceReader->ReadSample(MF_SOURCE_READER_FIRST_AUDIO_STREAM,
                                                0, nullptr, &flags, nullptr,
                                                &mfSample);

    // Check if we're at the end of the file
    if (flags & MF_SOURCE_READERF_ENDOFSTREAM)
    {
        endOfFile = true;
        *pAudioBufferLength = 0;
        return nullptr;
    }

    // If not, convert the data to a contiguous buffer
    ComPtr<IMFMediaBuffer> mfMediaBuffer;
    hresult = mfSample->ConvertToContiguousBuffer(&mfMediaBuffer);

    // Lock the audio buffer and copy the samples to local memory
    uint8 * pAudioData = nullptr;
    DWORD audioDataLength = 0;
    hresult = mfMediaBuffer->Lock(&pAudioData, nullptr, &audioDataLength);
    byte * pAudioBuffer = new byte[audioDataLength];
    CopyMemory(pAudioBuffer, pAudioData, audioDataLength);
    hresult = mfMediaBuffer->Unlock();

    *pAudioBufferLength = audioDataLength;
    return pAudioBuffer;
}

// Callback methods from IXAudio2VoiceCallback
void _stdcall AudioFilePlayer::OnBufferEnd(void* pContext)
{
    // Remember to free the audio buffer!
    delete[] pContext;

    // Either submit a new buffer or clean up
    if (!endOfFile)
    {
        SubmitBuffer();
    }
    else
    {
        pSourceVoice->DestroyVoice();
        HRESULT hresult = MFShutdown();
    }
}
```

## Playing the File

I originally wanted to have all the code for this first program in the `MainPage` class, but I also wanted to use an XAudio2 callback function. That's a problem because (as I discovered) a Windows Runtime type like `MainPage` can't implement a non-Windows Runtime interface like `IXAudio2VoiceCallback`, so I needed a second class, which I called `AudioFilePlayer`.

After obtaining an `IMFSourceReader` object from the method shown in **Figure 2**, `MainPage` creates a new `AudioFilePlayer` object, also passing to it an `IXAudio2` object created in the `MainPage` constructor:

```
new AudioFilePlayer(pXAudio2, mfSourceReader);
```

From there, the `AudioFilePlayer` object is entirely on its own and pretty much self-contained. That's how the program can play multiple files simultaneously.

To play the music file, `AudioFilePlayer` needs to create an `IXAudio2-SourceVoice` object. This requires a `WAVEFORMATEX` structure indicating the format of the audio data to be passed to the source voice, and that should be consistent with the audio data being delivered by the `IMFSourceReader` object. You can probably guess at the correct parameters (such as two channels and a 44,100 Hz sampling rate), and if you get the sampling rate wrong, XAudio2 can perform sample rate conversions internally. Still, it's best to obtain a `WAVEFORMATEX` structure from the `IMFSourceReader` and use that, as shown in the `AudioFilePlayer` constructor in **Figure 3**.

Getting that `WAVEFORMATEX` structure is a bit of a nuisance that involves a memory block that must then be explicitly freed, but by the conclusion of the `AudioFilePlayer` constructor, the file is ready to be played.

To keep the memory footprint of such a program to a minimum, the file should be read and played in small chunks. Both Media Foundation and XAudio2 are very conducive to this approach. Each call to the `ReadSample` method of the `IMFSourceReader` object obtains access to the next block of uncompressed data until the file is entirely read. For a sampling rate of 44,100 Hz, two channels and 32-bit floating-point samples, my experience is that these blocks are usually 16,384 or 32,768 bytes in size, and sometimes as little as 12,288 bytes (but always a multiple of 4,096), indicating about 35 to 100 milliseconds of audio each.

Following each call to the `ReadSample` method of `IMFSourceReader`, a program can simply allocate a local block of memory, copy the data to that, and then submit that local block to the `IXAudio2SourceVoice` object with `SubmitSourceBuffer`.

`AudioFilePlayer` uses a two-buffer approach to play the file: While one buffer is being filled with data, the other buffer is playing. **Figure 4** shows the entire process, again without the checks for errors.





Figure 5 The DeeJay Program

To get temporary access to the audio data, the program needs to call `Lock` and then `Unlock` on an `IMFMediaBuffer` object representing the new block of data. In between those calls, the `GetNextBlock` method in **Figure 4** copies the block into a newly allocated byte array.

The `SubmitBuffer` method in **Figure 4** is responsible for setting the fields of an `XAUDIO2_BUFFER` structure in preparation for submitting the audio data for playing. Notice how this method also sets the `pContext` field to the allocated audio buffer. This pointer is passed to the `OnBufferEnd` callback method seen toward the end of **Figure 4**, which can then delete the array memory.

When a file has been entirely read, the next `ReadSample` call sets an `MF_SOURCE_READERF_ENDOFSTREAM` flag and the `IMFSample` object is null. The program responds by setting an `endOfFile` field variable. At this time, the other buffer is still playing, and a last call to `OnBufferEnd` will occur, which uses the occasion to release some system resources.

There's also an `OnStreamEnd` callback method that's triggered by setting the `XAUDIO2_END_OF_STREAM` flag in the `XAUDIO2_BUFFER`, but it's hard to use in this context. The problem is that you can't set that flag until you receive an `MF_SOURCE_READERF_ENDOFSTREAM` flag from the `ReadSample` call. But `SubmitSourceBuffer` does not allow null buffers or buffers of zero size, which means you have to submit a non-empty buffer anyway, even though no more data is available!

## Spinning a Record Metaphor

Of course, passing audio data from Media Foundation to `XAudio2` is not nearly as easy as using the Windows 8 `MediaElement`, and hardly worth the effort unless you're going to do something interesting with the audio data. You can use `XAudio2` to set some special effects (such as echo or reverb), and in the next installment of this column I'll apply `XAudio2` filters to sound files.

Meanwhile, **Figure 5** shows a program named `DeeJay` that displays an on-screen record and rotates it as the music is playing at a default rate of 33 1/3 revolutions per minute.

Not shown is an application bar with a `Load File` button and two sliders—one for the volume and another to control the playback speed. This slider has values that range from -3 to 3 and indicate a speed ratio. The default value is 1. A value of 0.5 plays back the file at half speed, a value of 3 plays the file back three times as fast, a value of 0 essentially pauses playback, and negative values play the file backward (perhaps allowing you to hear hidden messages encoded in the music).

Because this is Windows 8, of course you can also spin the record with your fingers, thus justifying the program's name. `DeeJay`

allows for single-finger rotation with inertia, so you can give the record a good spin in either direction. You can also tap the record to move the “needle” to that location.

I very, very, very much wanted to implement this program in a manner similar to the `StreamMusicFile` project with alternating calls to `ReadSample` and `SubmitSourceBuffer`. But problems arose when attempting to play the file backward. I really needed `IMFSourceReader` to support a `ReadPreviousSample` method, but it does not.

Of course, passing audio data from Media Foundation to `XAudio2` is not nearly as easy as using the Windows 8 `MediaElement`.

What `IMFSourceReader` does support is a `SetCurrentPosition` method that allows you to move to a previous location in the file. However, subsequent `ReadSample` calls begin returning blocks earlier than that position. Most of the time, a series of calls to `ReadSample` eventually meet up at the same block as the last `ReadSample` call before `SetCurrentPosition`, but sometimes they don't, and that made it just too messy.

I eventually gave up, and the program simply loads the entire uncompressed audio file into memory. To keep the memory footprint down, I specified 16-bit integer samples rather than 32-bit floating-point samples, but still it's about 10MB of memory per minute of audio, and loading in a long movement of a Mahler symphony would commandeer about 300MB.

Those Mahler symphonies also mandated that the entire file-loading method be executed in a secondary thread, a job that's greatly simplified by the `create_task` function available in Windows 8.

To ease working with the individual samples, I created a simple structure named `AudioSample`:

```
struct AudioSample
{
    short Left;
    short Right;
};
```

So instead of working with an array of bytes, the `AudioFilePlayer` class in this program works with an array of `AudioSample` values. However, this means that the program is basically hardcoded for stereo files. If it loads an audio file that doesn't have exactly two channels, it can't play that file!

The asynchronous file-reading method stores the data it obtains in a structure I call `LoadedAudioFileInfo`:

```
struct LoadedAudioFileInfo
{
    AudioSample* pBuffer;
    int bufferLength;
    WAVEFORMATEX waveFormat;
};
```

The `pBuffer` is the big block of memory, and `bufferLength` is the product of the sampling rate (probably 44,100 Hz) and the duration of the file in seconds. This structure is passed directly to the `AudioFilePlayer` class. A new `AudioFilePlayer` is created for each loaded file, and it replaces any previous `AudioFilePlayer` instance. For cleaning up, `AudioFilePlayer` has a destructor that deletes the big array holding the entire file, as well as two smaller arrays used for submitting buffers to the `IXAudio2SourceVoice` object.

The keys to playing the file forward and backward at various speeds are two fields in `AudioFilePlayer` of type `double`: `audioBufferIndex` and `speedRatio`. The `audioBufferIndex` variable points to a location within the big array containing the entire uncompressed file. The `speedRatio` variable is set to the same values as the slider, -3 through 3. When the `AudioFilePlayer` needs to transfer audio data from the big buffer into the smaller buffers for submission, it

increments `audioBufferIndex` by `speedRatio` for each sample. The resultant `audioBufferIndex` is (in general) between two file samples, so the method in **Figure 6** performs an interpolation to derive a value that's then transferred to the submission buffer.

## The Touch Interface

To keep the program simple, the entire touch interface consists of a Tapped event (to position the "needle" at a different location on the record) and three Manipulation events: the `ManipulationStarting` handler initializes single-finger rotation; the `ManipulationDelta` handler sets a speed ratio for the `AudioFilePlayer` that overrides the speed ratio from the slider; and the `ManipulationCompleted` handler restores the speed ratio in `AudioFilePlayer` to the slider value after all inertial movement has completed.

Rotational velocity values  
are directly available from  
event arguments of the  
`ManipulationDelta` handler.

Rotational velocity values are directly available from event arguments of the `ManipulationDelta` handler. These are in units of degrees of rotation per millisecond. If you consider that a standard long-playing record speed of 33 1/3 revolutions per minute is equivalent to 200° per second, or 0.2° per millisecond, I merely needed to divide the value in the `ManipulationDelta` event by 0.2 to obtain the speed ratio I required.

However, I discovered that the velocities reported by the `ManipulationDelta` are quite erratic, so I had to smooth them out with some simple logic involving a field variable named `smoothVelocity`:

```
smoothVelocity = 0.95 * smoothVelocity +
    0.05 * args->Velocities.Angular / 0.2;
pAudioFilePlayer->SetSpeedRatio(smoothVelocity);
```

On a real turntable, you can stop rotation by simply pressing your finger on the record. But that doesn't work here. Actual movement of your finger is necessary for Manipulation events to be generated, so to stop the record you need to press and then move your finger (or mouse or pen) a bit.

The inertial deceleration logic also doesn't match up with reality. This program allows inertial movement to finish entirely before restoring the speed ratio to the value indicated by the slider. In reality, that slider value should exhibit a type of pull on the inertial values, but that would have complicated the logic considerably.

Besides, I couldn't really detect an "unnatural" inertial effect. Undoubtedly a real DJ would feel the difference right away. ■

**Figure 6** Interpolating Between Two Samples in `DeeJay`

```
AudioSample AudioFilePlayer::InterpolateSamples()
{
    double left1 = 0, left2 = 0, right1 = 0, right2 = 0;

    for (int i = 0; i < 2; i++)
    {
        if (pAudioBuffer == nullptr)
            break;

        int index1 = (int)audioBufferIndex;
        int index2 = index1 + 1;
        double weight = audioBufferIndex - index1;

        if (index1 >= 0 && index1 < audioBufferLength)
        {
            left1 = (1 - weight) * pAudioBuffer[index1].Left;
            right1 = (1 - weight) * pAudioBuffer[index1].Right;
        }

        if (index2 >= 0 && index2 < audioBufferLength)
        {
            left2 = weight * pAudioBuffer[index2].Left;
            right2 = weight * pAudioBuffer[index2].Right;
        }
    }

    AudioSample audioSample;
    audioSample.Left = (short)(left1 + left2);
    audioSample.Right = (short)(right1 + right2);
    return audioSample;
}
```

**CHARLES PETZOLD** is a longtime contributor to MSDN Magazine and the author of "Programming Windows, 6th edition" (O'Reilly Media, 2012), a book about writing applications for Windows 8. His Web site is [charlespetzold.com](http://charlespetzold.com).

**THANKS** to the following technical experts for reviewing this article:  
*Richard Fricks (Microsoft)*

# FOSE

EXPERIENCE TECHNOLOGY

**MAY 14-16, 2013**  
**WASHINGTON, DC**

WALTER E. WASHINGTON CONVENTION CENTER



## CYBERSECURITY

### The Business of Cybersecurity

Detect, determine and develop your strategy.

## Keynotes

General  
McChrystal



Jan R.  
Frye

Joe  
Thiesmann



Steven  
VanRoekel  
(Invited)

## PROJECT MANAGEMENT

### Improve Efficiencies & Manage Risk

Learn skill improvement, agile development, metrics & more.

## CLOUD & VIRTUALIZATION

### Benefit Today & Plan for Tomorrow

Do more with less. Obtain new guidance on cloud initiatives.



## Government Tech Talks

Valuable technology implementation —3 strategies in 15 minutes!

## First @ FOSE

See industry giants' latest products and services.

# EXPERIENCE TECHNOLOGY

**SEE IT. HEAR IT. TRY IT. KNOW IT.**  
**Experience NEW Solutions at FOSE.**

## BIG DATA & BUSINESS INTELLIGENCE

### Extract Meaning

Use your bits and bytes to reach objectives.

## App Arcade

Experience the latest apps for government.

## MOBILE GOVERNMENT

### Digital Government Strategy

Achieve your BYOD, security, API and Enterprise System goals.

## FREE EXPO\*

Hands-on demos, free education, networking and more!

## REGISTER TODAY!

**Early Bird Special—SAVE \$100!**

**USE PRIORITY CODE: FOSEAD2**

# FOSE.com

\*Expo is free for government; \$50 for industry suppliers.



3-DAY PAID CONFERENCE delivering best practices & case studies from some of the biggest names in government.



FREE EXPO\* showcasing industry partners & their solutions to help you reach your mission goals!

PLATINUM PARTNER



GOLD SPONSORS



SILVER SPONSORS



TECHNOLOGY SPONSORS



1105 GOVERNMENT  
EVENTS

PRODUCED BY





# Coining Currency

One of the delights of writing this column is that I get to see lots of new things. Sometimes their creators forget to name them, so I get to do it. And because they're new, I sometimes have to coin new words to describe them. Here are some of my favorites:

**hassle budget** (n.): The amount of security-related overhead that a user is willing to tolerate before he either throws away your product or figures out a workaround. "Wow, that User Account Control popping up all the time asking, 'Are you sure?' is a real pain in the ass, especially because I've never once said 'no' to it. It's way over my hassle budget. I'm turning the thing off."

The key distinguishing feature  
of the marketingbozo  
is his constant spouting of  
technical buzzwords he  
doesn't understand.

**marketingbozo** (n.): A person who attempts to sell a product without understanding it, and without understanding his lack of understanding. "Bob, you marketingbozo! You poured our entire budget into promoting caffeine-free diet Jolt Cola. We're broke, and you're fired."

The key distinguishing feature of the marketingbozo is his constant spouting of technical buzzwords he doesn't understand. At a TechEd many years ago, I stopped to chat with a marketingbozo who was turning cartwheels over the fact that the software he sold had just been made object-oriented. "Please forgive my ignorance," I asked him, "but what exactly is object-oriented software, how does it differ from software that's not object-oriented, and why is that difference something I care about buying?" Watching the poor sod wriggle until his technical guy came back and recognized me (we both laughed) was the most fun I had at the whole conference. Call me easily amused.

**armadillo** (n.): A technology product that fails because its functionality falls between two successful niches, offering the drawbacks

of both but the advantages of neither. I got the idea from a Texan student of mine, who observed, "There ain't nothing in the middle of the road 'cept yellow lines and squashed armadillos." If something is neither fish nor fowl, it's probably an armadillo. "This device is too hot, heavy and expensive for a tablet, and way too underpowered and too hard to type on for a notebook. What an armadillo!"

**MINFU** (n.): An acronym standing for Microsoft Nomenclature Foul-Up, based on the military acronyms SNAFU and FUBAR that crossed into general usage decades ago. I coined it in a 1998 column on Byte.com, and it's been in three editions of my Microsoft Press book. Several other authors have picked it up, most notably David Chappell.

For example, a marketingbozo once attempted to name the in-place activation of an embedded object in Office as "Visual Editing." I guess he wanted to distinguish it from tactile editing, or perhaps olfactory editing. I said to the guy, "Well, my embedded objects are sound presentations, so they're not visual. And I don't edit them in place, I play them. Do you still want me to call it Visual Editing, even though it's not editing and it's not visual?"

And now, of course, the latest MINFU is "Metro." Microsoft used that name to describe its new tile-based interface, which debuted with the Windows 7 phone, in November of 2010. Almost two years later, just before the debut of Windows 8, Microsoft shouted, "Whoa! Hold everything!" It turns out that a German supermarket company named Metro AG claimed rights to the name. A friend of mine barely managed to retrieve his book manuscript containing the "M" word the day before the presses rolled.

So what do we call apps of the type "Formerly Known as Metro"? At the time of this writing, I've been directed to use the term "Windows Store app" to describe the tile-based interface. What to call these apps when you get them from other sources, I don't quite know. ATCHCFTSETYAGTSE (Apps That Could Have Come from the Store Even Though You Actually Got Them Somewhere Else)? MINFU. ■

---

**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](mailto:rollthunder.com).



100s of UI controls for all .NET platforms  
including grids, charts, reports & schedulers  
Visual Studio 2012 support  
Windows 8 Studios for WinRT XAML & WinJS  
New Modern UI themes

ComponentOne®  
**Studio® Enterprise**

**ComponentOne®**  
a division of GrapeCity®

Download your free trial @  
**[componentone.com/se](http://componentone.com/se)**

© 2013 GrapeCity, inc. All rights reserved. All other product and brand names are trademarks and/or registered trademarks of their respective holders.

# Read and write Excel, Word, and PDF files in your .NET apps



Our complete suite of  
**file manipulation products**

Offer price

**\$995**

Regular price

**\$1,995**

Includes 1 year of  
support and updates

No dependency on Office automation | Royalty free, no per-server fees | Deploy anywhere, including cloud platforms

## Design your solution

- ★ Excel: Reports with charts, pivot tables, formulas, and more.
- ★ Word: Mail merge, complex formatting, export to PDF, and more.
- ★ PDF: Complex documents from code, Word templates, and more.



DOWNLOAD NOW

[syncfusion.com/msdn/april](https://syncfusion.com/msdn/april)



*\*All trademarks mentioned belong to their respective owners.*