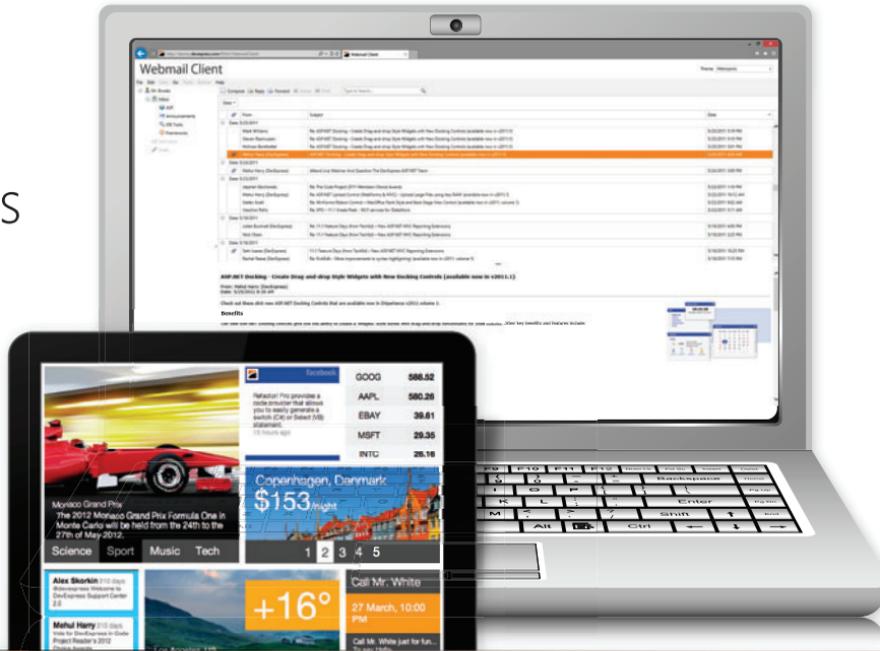




When the web means business

Performance
Elegance
Productivity



Download your
30-day FREE trial at
www.DevExpress.com

DXv2

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



msdn

magazine



Parse JSON Strings
in WinRT Components.....32

Parse JSON Strings in Windows Runtime Components Craig Shoemaker	32
Leverage Multiple Code Frameworks with One ASP.NET Jeff Fritz	42
Migrating Legacy .NET Libraries to Target Modern Platforms Josh Lane	50
Windows Phone Video Capture: A Best-of-Breed Approach Chris Barker	56
Understanding and Using the SharePoint 2013 REST Interface Jim Crowley	62
Shortest-Path Graph Analysis Using a CLR Stored Procedure James McCaffrey	68

COLUMNS

CUTTING EDGE

Social Authentication
in ASP.NET MVC 4
Dino Esposito, page 6

WINDOWS WITH C++

Introducing Direct2D 1.1
Kenny Kerr, page 12

WINDOWS AZURE INSIDER

Geo-Protection for Video Blobs
Using a Node.js Media Proxy
Bruno Terkaly and
Ricardo Villalobos, page 24

TEST RUN DATA

Data Clustering Using
Category Utility
James McCaffrey, page 76

DIRECTX FACTOR

Exploring Filters in XAudio2
Charles Petzold, page 82

DON'T GET ME STARTED

Do As I Say, Not As I Do
David Platt, page 88

Desktop

Deliver high performance, scalable and stylable touch-enabled enterprise applications in the platform of your choice.



Native Mobile

Develop rich, device-specific user experience for iOS, Android, and Windows Phone, as well as mobile cross-platform apps with Mono-Touch.



Download Your Free Trial
infragistics.com/enterprise-READY



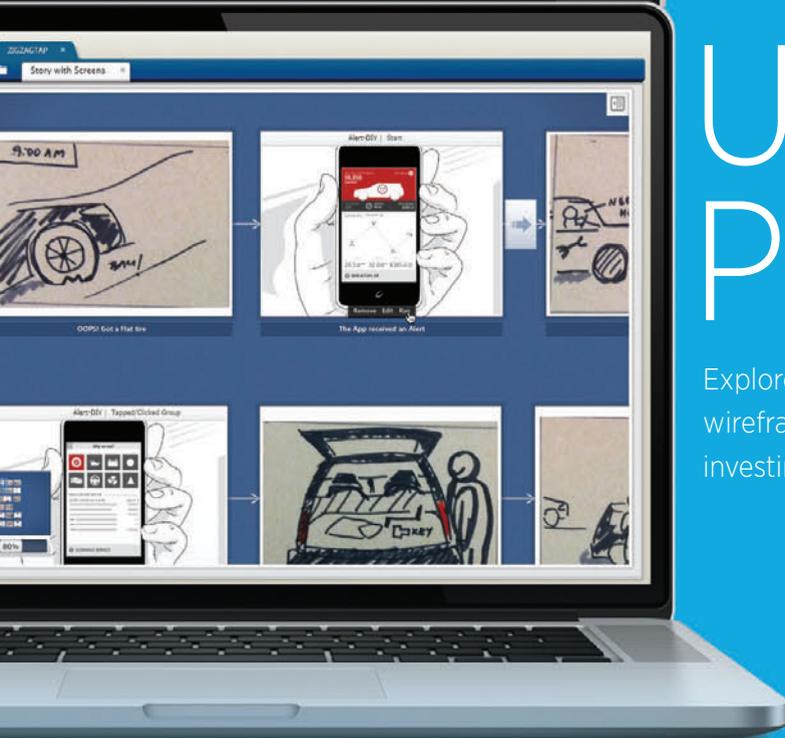
Cross-Platform

Build standards-based, touch-enabled HTML5 & jQuery experiences for desktop, tablet, and mobile delivery, including multi-device targeting with frameworks such as PhoneGap and MVC.



UX Prototyping

Explore design ideas through rapid, user-centered wireframing, prototyping, and evaluation before investing a single line of code.



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.



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

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 1-800-IT-FINDS

msdn magazine

MAY 2013 VOLUME 28 NUMBER 5

BJÖRN RETTIG Director

MOHAMMAD AL-SABT Editorial Director/mmeditor@microsoft.com

PATRICK O'NEILL Site Manager

MICHAEL DESMOND Editor in Chief/mmeditor@microsoft.com

DAVID RAMEL Technical Editor

SHARON TERDEMAN Features Editor

WENDY HERNANDEZ Group Managing Editor

KATRINA CARRASCO Associate Managing Editor

SCOTT SHULTZ Creative Director

JOSHUA GOULD Art Director

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

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

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 or call (847) 763-9560. POSTMASTER: Send address changes to MSDN Magazine, P.O. Box 2166, Skokie, IL 60076. Canada Publications Mail Agreement No: 40612608. Return Undeliverable Canadian Addresses to Circulation Dept. or XPO Returns: P.O. Box 201, Richmond Hill, ON L4B 4R5, Canada.

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

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

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

Media Kits: Direct your Media Kit requests to Matt Morollo, VP Publishing, 508-532-1418 (phone), 508-875-6622 (fax), mmorollo@1105media.com

Reprints: For single article reprints (in minimum quantities of 250-500), e-prints, plaques and posters contact: PARS International, Phone: 212-221-9595, E-mail: 1105reprints@parsintl.com, www.magreprints.com/QuickQuote.asp

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

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



Printed in the USA



Manage Information Workflows with Altova® FlowForce® Server

Finally, there is a flexible workflow orchestration tool to **automate essential business processes** defined using the award-winning Altova® MissionKit® developer tools.

Introducing FlowForce, the powerful new server tool for managing today's multi-step, enterprise-level data validation, processing, aggregation, and reporting tasks.

FlowForce Server is at the center of Altova's new line of server tools optimized for today's high-performance, parallel computing environments:

- **FlowForce® Server** for orchestrating events, triggers, and the automation of processes
- **MapForce® Server** for automating any-to-any data mapping and aggregation processes
- **StyleVision® Server** for automating business report generation in HTML, PDF, and Word

Tight integration with Altova MissionKit developer tools including MapForce and StyleVision lets you design data mapping projects and report templates, and then easily deploy them to FlowForce Server for automation with comprehensive management options and secure access control.



Learn more and download a free trial of Altova FlowForce and the complete line of Altova server tools at www.altova.com/server





Design Matters

David Platt, in this month's Don't Get Me Started column, returns to a rant he's made before in the pages of this magazine: that the compelling new visual capabilities of Windows 8 may inspire as much bad design in new applications as they do compelling design. Now Platt turns his ire on live tiles, noting that Microsoft's own developers are guilty of breaking the published Microsoft design guidance about how often live tiles should update on the screen.

As a guy steeped in the practice of UI design, Platt saw this coming from a mile away. "Every time a new graphical environment arrives, users pay the price while designers relearn the basic principles that should be branded on their hearts: Users don't care about your app for itself—they only care about their own problems and their own pleasures," Platt presciently wrote in his November 2012 column ("Here We Go Again," msdn.microsoft.com/magazine/jj721604).

Microsoft, Hollis said, is in a race with other developer ecosystems to get its community fully vested in the practice of design. It's a race that Microsoft is late entering, but serious about winning.

This new graphical environment is different, however. When XAML was introduced back in 2006, it opened design opportunities for Microsoft developers writing Windows Presentation Foundation (WPF) or Silverlight applications. But as Billy Hollis, a consultant at Next Version Systems and a design expert in the Microsoft development space, told me during an interview at the Visual Studio Live! conference in Las Vegas recently, the XAML design revolution never really took off.

"We thought a lot of people would [pursue design] in the XAML era, and almost no one did. You look at most XAML apps and they're just kind of Windows Forms designs with pretty colors," Hollis said. "I call them decorated apps. They're not designed, they're decorated."

But the new Microsoft design language introduced with Windows 8 and Windows Runtime represents a far more strategic shift. Microsoft, Hollis said, is in a race with other developer ecosystems to get its community fully vested in the practice of design. It's a race that Microsoft is late entering, but serious about winning. The problem, explained Hollis, is that changing developer priorities takes time.

"First of all, there's a foundation you have to learn," Hollis said. "You have to know certain principles. And then there's a process shift you have to understand. It's a discipline. And most developers are not going to do that. They got into this business to write code and that's what they're going to do."

We're seeing symptoms of this challenge as Microsoft aggressively pursues its new UX strategy. As Hollis noted, individuals and organizations can't flip a switch and emerge with a mature appreciation of application design—it takes three or four years to master the discipline. In that regard, issues like overly helpful live tiles are an unsurprising outgrowth of an ecosystem in transition.

So how can Microsoft and its developers best move forward? Hollis urged devs to take note of the revolution they see unfolding on their phones and tablets, and to apply those lessons to their own practice. But most of all, he wants them to be bold.

"The biggest thing I'm trying to help them to understand is that they have to make a break with the past. That's the hardest thing to accomplish," he said.

Hollis said developers need to do two things to kick off the transition. One is to gain an appreciation for the importance of design, or at the very least to be willing to work with those who do. The second is to get emotional about the way their—and others'—software looks and acts.

"They need to feel emotionally unsatisfied if they end up with a modern app on Windows 8 that's a kluge-up adaptation of what they had before," Hollis said. "They need to feel some emotional rejection of that. That will fuel the rest of the process."

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

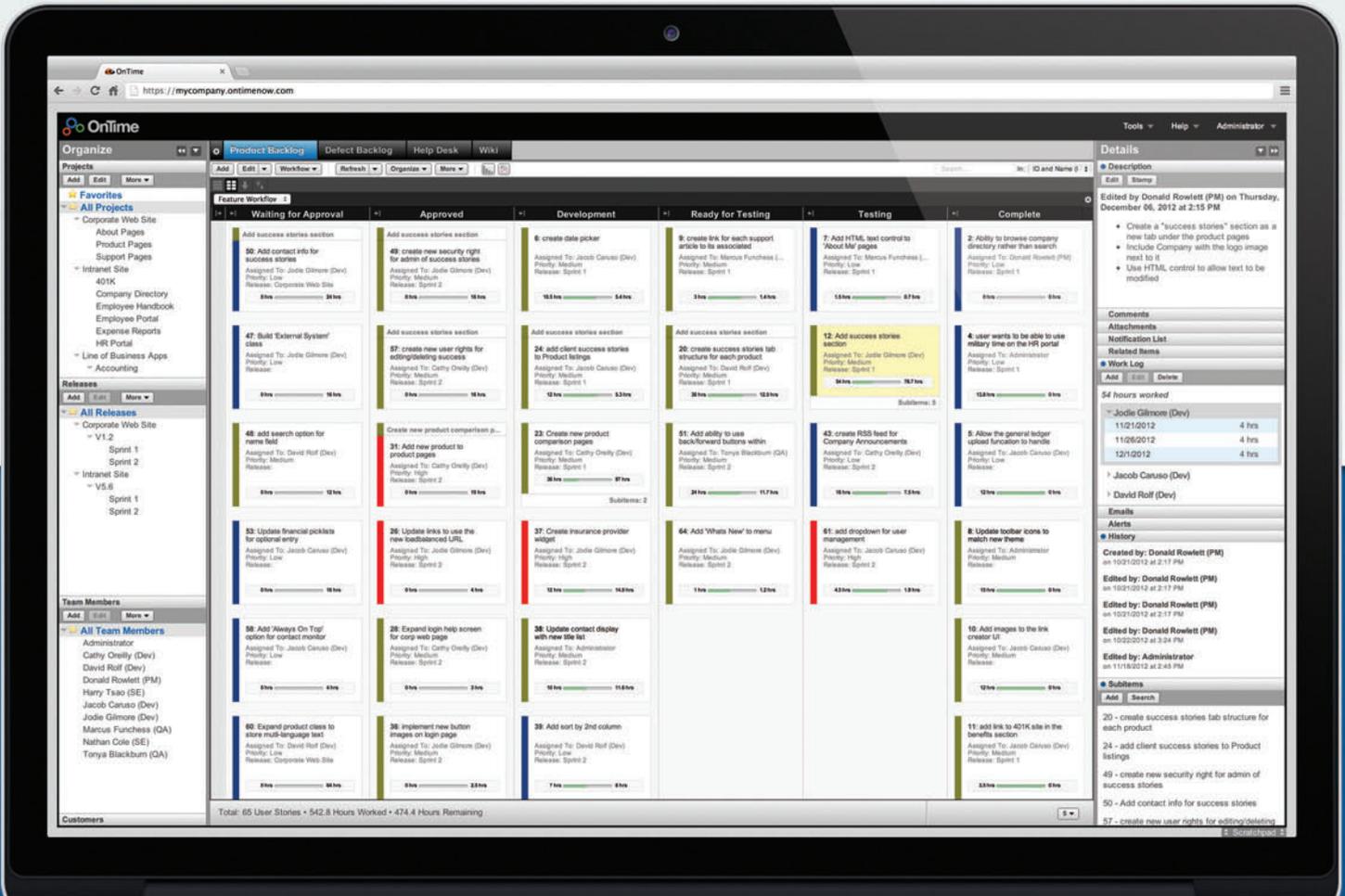
A listing of Microsoft Corporation trademarks can be found at microsoft.com/library/toolbar/3.0/trademarks/en-us.mspx. Other trademarks or trade names mentioned herein are the property of their respective owners.

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



\$7

per user per month





Social Authentication in ASP.NET MVC 4

As I see things, most Web sites that need to authenticate users will use social authentication gateways. In this context, a social authentication gateway is merely the authentication platform publicly exposed by popular social networks such as Twitter and Facebook. If you think back to the early days of ASP.NET, you can't help but notice some likeness between the idea behind the Passport initiative and today's social authentication gateways. They certainly aren't the same thing, but in both scenarios you can recognize the goal of making users' lives easier by reducing the number of credentials of which they must keep track.

Using the same set of credentials has pros and cons. On one hand it lowers the security barrier around your privacy. By always using the same username and password, you give hackers more time to guess your secrets; at the same time, you expose a lot more information once hacked. On the other hand, though, using the same credentials over and over makes it easier for you to connect and try out new services. For service owners this is great because they can see the number of users grow more quickly, which means the service has more chance to be successful.

In a nutshell, more and more Web sites aiming at attracting a large user base today offer the double option of registering by choosing your own credentials or through a social network authentication gateway. The trend is so interesting that in the newest version of ASP.NET MVC 4 you find an ad hoc framework to authenticate users via a number of social networks. In this article I'll review the code you get from the ASP.NET MVC 4 project template. Social authentication gateways use the Open Authentication (OAuth) protocol, which turns out to be quite a verbose protocol. Therefore, the resulting code isn't exactly trivial and may necessitate some further explanation.

The Quickest Way to MVC Authentication

ASP.NET MVC gives you the chance to start coding from a template of code that already includes authentication. The Internet Application template you get with ASP.NET MVC 4 extends the support to social gateways. Let's assume, then, that you have a brand-new Internet Application project. On the first build

of the code, nothing special happens. The page in **Figure 1** informs you that no external authentication service has been enabled yet.

You can enable any of the supported services by making some light changes to the global.asax codebehind file. In ASP.NET MVC 4 it's recommended that projects include an App_Start folder with xxxConfig classes that perform initialization tasks. Such classes are plain containers of static methods that better organize the bootstrapping code of the application. Here's a snapshot:

```
protected void Application_Start()
{
    ...
    AuthConfig.RegisterAuth();
}

public static class AuthConfig
{
    public static void RegisterAuth()
    {
        OAuthWebSecurity.RegisterTwitterClient(
            consumerKey: yourTwitterAppKey,
            consumerSecret: yourTwitterAppSecret);

        OAuthWebSecurity.RegisterFacebookClient(
            appId: yourFacebookAppKey,
            appSecret: yourFacebookAppSecret);
    }
}
```

The RegisterAuth method contains the code to let users log in using their accounts from other sites such as Facebook and Twitter:

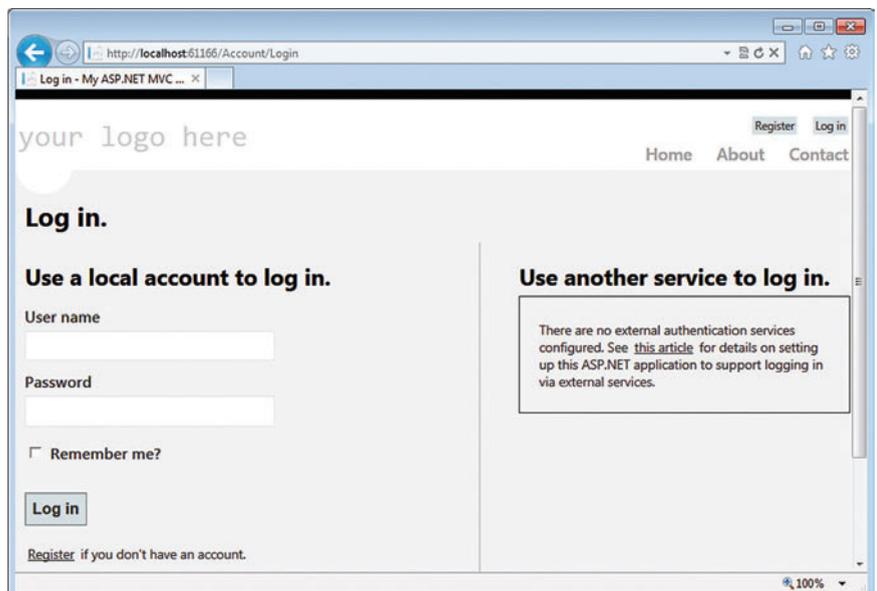


Figure 1 The ASP.NET MVC 4 Template When No External Authentication Service Is Configured



FILE APIs

CONVERT
PRINT
CREATE
COMBINE
MODIFY



Files
from your
CODE

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.

... and many others!



Scan for 20%
Coupon Code



Follow us on
Facebook & Twitter

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

US Sales: 1.888.277.6734
sales@aspose.com

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

AU Sales: +61 2 8003 5926
sales.asiapacific@aspose.com

The OAuthWebSecurity class comes from the Web Pages framework and is located in the Microsoft.Web.WebPages.OAuth namespace. This class wraps up core functionality of the OAuth protocol as implemented in the DotNetOpenAuth (DNOA) library (see dotnetopenauth.net).

In order to authenticate users against a social network you first need to create an application within the social network. So you need a Twitter application and a Facebook application to authenticate users via Twitter and Facebook from within your site. Most likely you'll create a Twitter/Facebook application with the same name as your site and configure it to link back to the Web site. What's referenced here as a Twitter/Facebook application isn't really a full-fledged application; moreover, it has a special developer token to programmatically access Twitter, Facebook and other social networks. In past installments of this column dedicated to Facebook programming I covered this aspect fairly in-depth. For the purpose of this article, a Twitter/Facebook application consists of a pair of strings—one known as the key and the other known as the secret. The key and secret are uniquely associated with the social application. You initialize OAuthWebSecurity by passing the key and secret for each social network you intend to support.

By simply adding calls to RegisterTwitterClient and RegisterFacebookClient, the UI of the sample project changes to show those registration options as buttons. If a user clicks the Log in button, she'll be redirected to the Twitter/Facebook site to proceed with authentication. If it all works fine she'll then be redirected to the original site and be recognized by ASP.NET as an authenticated user.

Sounds like a very simple thing, right? Well, there are a lot of nitty-gritty details under the hood.

Taking the OAuth Route to Authentication

When the user clicks on the Twitter button the site navigates to Account/ExternalLogin. Let's have a look at the code for the method (the code is located in the AccountController.cs file):

```
public ActionResult ExternalLogin(String provider, String returnUrl)
{
    return new ExternalLoginResult(provider,
        Url.Action("ExternalLoginCallback",
            new { ReturnUrl = returnUrl }));
}
```

The ExternalLoginResult class is a sort of wrapper for the following code that really does the job of contacting the authentication gateway:

```
OAuthWebSecurity.RequestAuthentication(Provider, ReturnUrl);
```

The ExternalLoginResult class is a helper class also found in the AccountController.cs file. You should note that in the project template code the name of the provider is resolved by looking at the name attribute of the button:

```
<button type="submit"
    name="provider"
    value="@p.AuthenticationClient.ProviderName"
    title="Log in using your @p.DisplayName account">
    @p.DisplayName
</button>
```

At the end of the day, the RequestAuthentication method receives the name of the authentication provider (Twitter, Facebook or any of the other supported providers) and the URL to return. By default, OAuthWebSecurity supports the following

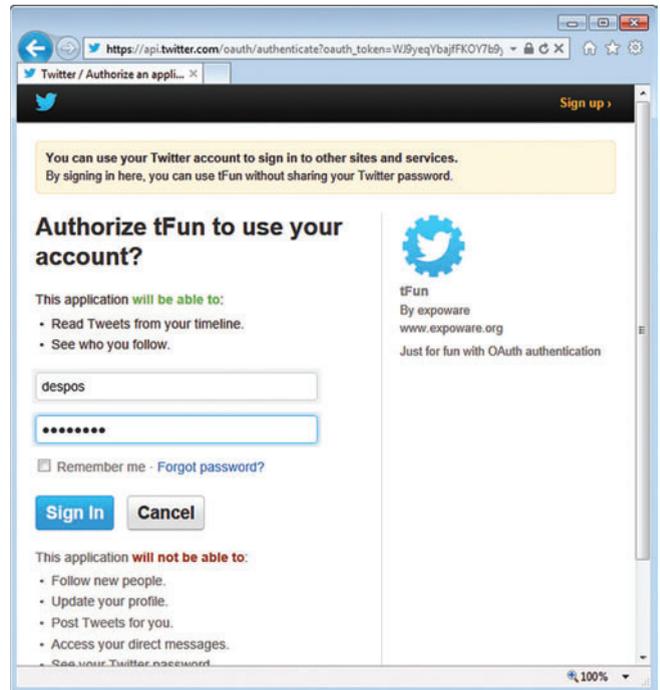


Figure 2 Authenticating via Twitter

providers: Twitter, Facebook, LinkedIn, Google, Microsoft and Yahoo. Internally, the method uses the DNOA library to carry the task. Let's see what happens if the user chooses to authenticate via Twitter.

As a first step, the default Twitter authentication page is displayed to the user. The page contains the name of the Twitter application that's conducting the task—it's tFun in the sample shown in Figure 2. When you configure a Twitter application, you also indicate the permissions the user needs to grant to the application upon login. By properly configuring the social application you can give the ASP.NET Web site the permission to follow new users or post on behalf of the connected user. This isn't the case with the sample application.

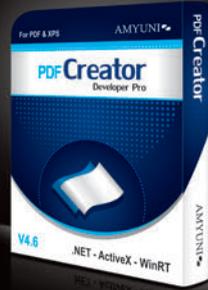
If the user enters credentials that Twitter (or the social network of choice) recognizes as valid, the Twitter site redirects back to the provided return URL. The next method where you regain control past the authentication is ExternalLoginCallback. What you know at this point is only that the user who's trying to access your application has been successfully recognized as a Twitter user. You don't know anything about her, not even her username. I can hardly think of an application that needs authenticated users and can blissfully ignore usernames or e-mail addresses. Back from the authentication step, the application only receives a code but hasn't yet been authorized to access the Twitter API programmatically. For this to happen, the code received at this stage must be exchanged for an access token (usually time-limited to prevent misuse). This is the purpose of the call to the method VerifyAuthentication you find in the body of ExternalLoginCallback. The AuthenticationResult object you get back from VerifyAuthentication brings back some information about the user. The actual information you get may be slightly different depending on the provider; however, it usually contains at least the username.

Powerful Tools for Developers

New!
V4.6

Create & Edit PDFs in .Net - ActiveX - WinRT

- Edit, process and print PDF 1.7 documents programmatically.
- Fast and lightweight 32 and 64-bit components for .Net, COM and WinRT applications.
- Create, fill-out and annotate PDF forms.



Complete Suite of Accurate PDF Components

- All your PDF processing, conversion and editing in a single package.
- Combines Amyuni PDF Converter and PDF Creator for easy licensing, integration and deployment.
- Includes our Microsoft certified PDF Converter printer driver.
- Export PDF documents into other formats such as JPEG, Word, Excel or Silverlight.

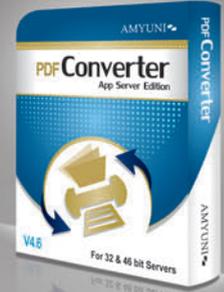


Advanced HTML to PDF & XAML

- Direct conversion of HTML files into PDF and XAML without the use of a web browser or a printer driver.
- Easy Integration and deployment within developer's applications.
- WebkitPDF is based on the Webkit Open Source library and Amyuni PDF Creator.



High Performance PDF Printer Driver



- Our high-performance printer driver optimized for Web, Application and Print Servers. Print to PDF in a fraction of the time needed with other tools. WHQL tested for Windows 32 and 64-bit including Windows Server 2008 and Windows 8.
- Standard PDF features included with a number of unique features. Interface with any .Net or ActiveX programming language.
- Easy licensing and deployment to fit system administrator's requirements.



AMYUNI

All development tools available at

www.amyuni.com

USA and Canada

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

Europe

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

From Authentication to Membership

Authenticating a user is only the first step. Next, you need to track the user by name within the site. In a classic ASP.NET membership system you first display a login form, validate credentials, and then create an authentication cookie with username and, optionally, other key information. Twitter or Facebook save you the burden of arranging a login form and validating the credentials, plus the nontrivial burden of storing and managing accounts with sensitive information such as passwords.

The bottom line, though, is that nearly any application that needs authenticated users also needs a membership system where each regular user is tracked by name. Building such a system is still your responsibility. The ASP.NET MVC 4 template comes to the rescue by offering an extra step where the user is automatically given a chance to enter her display name, which is then saved to a local membership table. This is required only the first time a user logs in to a given site. In other words, the form shown in **Figure 3** serves the purpose of joining registration and first login.

The name entered at this stage is used to create the ASP.NET authentication cookie, which definitely closes the circle. You used Twitter to check credentials, asked the user to enter her display name and created a regular authentication cookie. From now on, everything works as usual in ASP.NET for sites subject to authentication.

The default ASP.NET MVC 4 project template saves user data to an .mdf local database created under the App_Data folder. The table is managed using the simple membership API inherited from the Web Pages framework.

The following code shows how the sample project template gets the display name of the page of **Figure 3**:

```
var loginData = OAuthWebSecurity.SerializeProviderUserId(
    result.Provider, result.ProviderUserId);
var name = OAuthWebSecurity
    .GetOAuthClientData(result.Provider)
    .DisplayName;
return View("ExternalLoginConfirmation",
    new RegisterExternalLoginModel {
        UserName = result.UserName,
        ExternalLoginData = loginData
    });
```

The call to `GetOAuthClientData` is where you access any information that the Twitter provider shares about the logged-in user. In the method `ExternalLoginConfirmation` two key things happen, summarized by the following code:

```
OAuthWebSecurity.CreateOrUpdateAccount(provider, providerUserId, model.UserName);
OAuthWebSecurity.Login(provider, providerUserId, createPersistentCookie: false);
```

The first line sets up the new record in the membership local database for the application. The second line actually creates the authentication cookie. The default template provides for a bunch of database tables such as `UserProfiles` and `webPages_OAuth-Membership`. The latter table stores a record with the name of the provider (that is, Twitter), the provider unique ID for the user, and

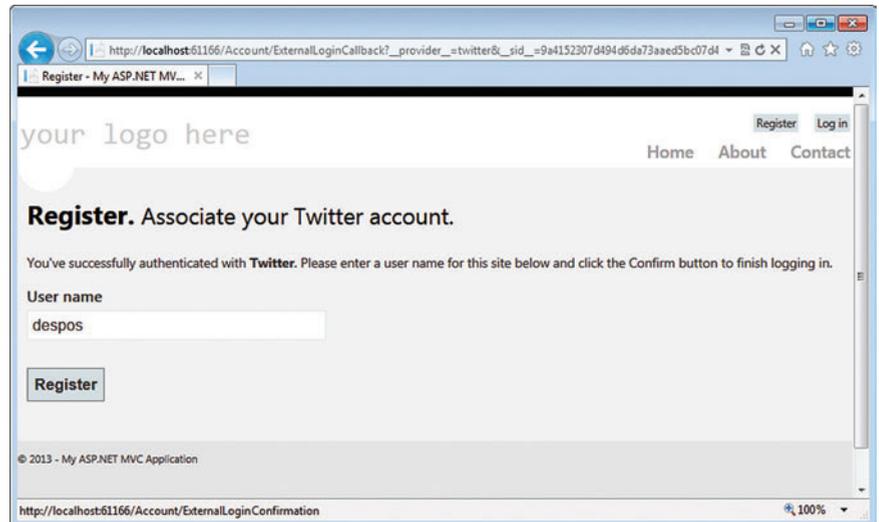


Figure 3 First Login and the Complete Registration to the Site

a pointer to an internal ID that uniquely identifies the user in the `UserProfiles` table with the display name the user herself chose in the page shown in **Figure 3**.

Final Considerations

The OAuth protocol manages the interaction between a provider and a client application. Twitter, Facebook and a few other popular social networks expose their APIs via OAuth. A client application can use the Twitter API for two main purposes: plain user authentication and operating against the provider on behalf of a consenting user. In both cases the client application must log in to the provider and get an access token. The access token is limited in time (but can programmatically be refreshed) and is authorized to perform only the operations that the end user approved when she entered credentials (see **Figure 2**). What happens once the application holds an access token depends on the needs of the application. The token can be used to retrieve, say, the e-mail address, and store it in an application-specific membership system. The token can also be used to post on behalf of the user. The ASP.NET authentication cookie remains valid even when the user disconnects from Twitter. An application, however, can't post if the user is disconnected from Twitter.

The ASP.NET MVC API—specifically the `OAuthWebSecurity` class—addresses the authentication scenario nicely, but it doesn't cover interacting with the social provider beyond getting a display name. It also integrates well with the simple membership provider of Web Pages for storing Twitter and Facebook user names locally. ■

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

THANKS to the following technical expert for reviewing this article:
Mani Subramanian (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

 **telerik**





Introducing Direct2D 1.1

Windows 8 launched with a major new version of Direct2D. Since then, Direct2D has made it into Windows RT (for ARM devices) and Windows Phone 8, both of which are based on this latest version of Windows. Support for Direct2D on the phone OS isn't yet official, but it's just a matter of time. What about Windows 7? A platform update is being prepared to bring the latest version of the DirectX family of APIs to Windows 7. It includes the latest versions of Direct2D, Direct3D, DirectWrite, the Windows Imaging Component, the Windows Animation Manager and so on. A major driver for this is, of course, Internet Explorer. Anywhere you find Internet Explorer 9 or higher, you'll find Direct2D. By the time you read this, it's likely that Internet Explorer 10 will be available on Windows 7 and that, too, will require Direct2D 1.1 to be installed as a matter of course. Given its ubiquity, there's really no reason not to use Direct2D 1.1. But what is Direct2D 1.1 and how can you start using it? That's the topic of this month's column.

Direct2D 1.1 might sound like a minor version update, and in some ways, it is. It doesn't fundamentally change the API. Everything you know about Direct2D continues to be every bit as relevant today. It's still modeled around device-specific and device-independent resources, render targets, geometries, brushes and so on. But in version 1.1, Direct2D grows up. The original version of Direct2D that launched with Windows 7 was in some ways an outsider to DirectX. It lagged behind, being tied to DirectX 10 rather than 11, the version of DirectX that it launched with. Even though it provided an excellent interop story for GDI and the Windows Imaging Component, it didn't provide the best possible experience for working with DirectX itself. It wasn't bad, but in Direct2D 1.1 things get a whole lot better. Direct3D and Direct2D are now in many ways siblings in the DirectX family. Thanks to this greater parity, even more of the graphics processing unit (GPU) is now available to the Direct2D developer without the need to jump out of the 2D abstraction. Moreover, when you do need to make the leap, it's both simple and efficient.

In my last column (msdn.microsoft.com/magazine/jj991972), I showed how you could use `ID2D1HwndRenderTarget`, the Direct2D HWND render target, in a desktop window. This mechanism continues to be supported by Direct2D 1.1 and is still the simplest way to get started with Direct2D, as it hides all of the underlying Direct3D and DirectX Graphics Infrastructure (DXGI) plumbing. To take advantage of the improvements in Direct2D 1.1 you need to eschew this render target, however, and instead opt for `ID2D1DeviceContext`, the new device context render target. At

Figure 1 Error Handling

```
#define ASSERT ATASSERT
#define VERIFY ATLVERIFY

#ifdef _DEBUG
#define HR(expression) ASSERT(S_OK == (expression))
#else
struct ComException
{
    HRESULT const hr;
    ComException(HRESULT const value) : hr(value) {}
};
inline void HR(HRESULT const hr)
{
    if (S_OK != hr) throw ComException(hr);
}
#endif
```

first, this might sound like something from Direct3D, and in some ways, it is. Like the `HWND` render target, the Direct2D device context inherits from the `ID2D1RenderTarget` interface and is thus very much a render target in the traditional sense, but it's a whole lot more powerful. Creating it, however, is a bit more involved but well worth the effort, as it provides many new features and is the only way to use Direct2D with the Windows Runtime (WinRT). Therefore, if you want to use Direct2D in your Windows Store or Windows Phone apps, you'll need to embrace the Direct2D device context. In this month's column I'll show you how to use the new render target in a desktop app. Next month, I'll show you how the render target works with the Windows Runtime—this has more to do with the Windows Runtime than it does with Direct2D. The bulk of what you need to learn has to do with managing the Direct3D device, the swap chain, buffers and resources.

The nice thing about the original Direct2D `HWND` render target was that you really didn't need to know anything about Direct3D or DirectX to get stuff done. That's no longer the case. Fortunately, there isn't a whole lot you need to know, as DirectX can certainly be daunting. DirectX is really a family of closely related APIs, of which Direct3D is the most well-known, while Direct2D is starting to steal some attention thanks to its relative ease of use and incredible power. Along the way, different parts of DirectX have come and gone. One relatively new member of the family is DXGI, which debuted with Direct3D 10. DXGI provides common GPU resource management facilities across the various DirectX APIs. Bridging the gap between Direct2D and Direct3D involves DXGI. The same goes for bridging the gap between Direct3D and the desktop's `HWND` or the WinRT `CoreWindow`. DXGI provides the glue that binds them all together.

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
or call 1-800-MELISSA (635-4772)

MELISSA DATA[®]
Your Partner in Data Quality

Headers and Other Glue

As I've done in the past, I'll continue to use the Active Template Library (ATL) on the desktop just to keep the examples concise. You can use your own library or no library at all. It really doesn't matter. I covered these options in my February 2013 column (msdn.microsoft.com/magazine/jj891018). To begin, you need to include the necessary Visual C++ libraries:

```
#include <wr1.h>
#include <atlbase.h>
#include <atlwin.h>
```

The Windows Runtime C++ Template Library (WRL) provides the handy ComPtr smart pointer, and the ATL headers are there for managing the desktop window. Next, you need the latest DirectX headers:

```
#include <d2d1_1.h>
#include <d3d11_1.h>
```

The first one is the new Direct2D 1.1 header file. The Direct3D 11.1 header is needed for device management. To keep things simple I'll assume the WRL and Direct2D namespaces are as follows:

```
using namespace Microsoft::WRL;
using namespace D2D1;
```

Next, you need to tell the linker how to resolve the factory functions you'll be using:

```
#pragma comment(lib, "d2d1")
#pragma comment(lib, "d3d11")
```

I tend to avoid talking about error handling. As with so much of C++, the developer has many choices for how errors can be handled. This flexibility is in many ways what draws me, and many others, to C++, but it can be divisive. Still, I get many questions

about error handling, so to avoid any confusion, **Figure 1** shows what I rely on for error handling in desktop apps.

The ASSERT and VERIFY macros are just defined in terms of the corresponding ATL macros. If you're not using ATL, then you could just use the C Run-Time Library (CRT) _ASSERT macro instead. Either way, assertions are vital as a debugging aid. VERIFY checks the result of an expression but only asserts in debug builds. In release builds the ASSERT expression is stripped out entirely while the VERIFY expression remains. The latter is useful as a sanity check for functions that must execute but shouldn't fail short of some apocalyptic event. Finally, HR is a macro that ensures the expression—typically a COM-style function or interface method—is successful. In debug builds, it asserts so as to quickly pinpoint the line of code in a debugger. In release builds, it throws an exception to quickly force the application to crash. This is particularly handy if your application uses Windows Error Reporting (WER), as the offline crash dump will pinpoint the failure for you.

The Desktop Window

Now it's time to start framing up the DesktopWindow class. First, I'll define a base class to wrap up much of the boilerplate windowing plumbing. **Figure 2** provides the initial structure.

DesktopWindow is a class template, as I rely on static or compile-time polymorphism to call the app's window class for drawing and resource management at appropriate points. Again, I've already

Figure 2 Desktop Window Skeleton

```
template <typename T>
struct DesktopWindow :
    CWindowImpl<DesktopWindowT,
        CWindow,
        CWinTraits<WS_OVERLAPPEDWINDOW | WS_VISIBLE>>
{
    ComPtr<ID2D1DeviceContext> m_target;
    ComPtr<IDXGISwapChain1> m_swapChain;
    ComPtr<ID2D1Factory1> m_factory;

    DECLARE_WND_CLASS_EX(nullptr, 0, -1);

    BEGIN_MSG_MAP(c)
        MESSAGE_HANDLER(WM_PAINT, PaintHandler)
        MESSAGE_HANDLER(WM_SIZE, SizeHandler)
        MESSAGE_HANDLER(WM_DISPLAYCHANGE, DisplayChangeHandler)
        MESSAGE_HANDLER(WM_DESTROY, DestroyHandler)
    END_MSG_MAP()

    LRESULT PaintHandler(UINT, WPARAM, LPARAM, BOOL &)
    {
        PAINTSTRUCT ps;
        VERIFY(BeginPaint(&ps));

        Render();

        EndPaint(&ps);
        return 0;
    }

    LRESULT DisplayChangeHandler(UINT, WPARAM, LPARAM, BOOL &)
    {
        Render();
        return 0;
    }

    LRESULT SizeHandler(UINT, WPARAM wparam, LPARAM, BOOL &)
    {
        if (m_target && SIZE_MINIMIZED != wparam)
        {
            ResizeSwapChainBitmap();

            Render();
        }

        return 0;
    }

    LRESULT DestroyHandler(UINT, WPARAM, LPARAM, BOOL &)
    {
        PostQuitMessage(0);
        return 0;
    }

    void Run()
    {
        D2D1_FACTORY_OPTIONS fo = {};

        #ifdef _DEBUG
        fo.debugLevel = D2D1_DEBUG_LEVEL_INFORMATION;
        #endif

        HR(D2D1CreateFactory(D2D1_FACTORY_TYPE_SINGLE_THREADED,
            fo,
            m_factory.GetAddressOf()));

        static_cast<T*>(this)->CreateDeviceIndependentResources();

        VERIFY(_super::Create(nullptr, nullptr, L"Introducing Direct2D 1.1"));

        MSG message;
        BOOL result;

        while (result = GetMessage(&message, 0, 0, 0))
        {
            if (-1 != result)
            {
                DispatchMessage(&message);
            }
        }
    }
};
```

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!

described the mechanics of ATL and desktop windows in my February 2013 column, so I won't repeat that here. The main thing to note is that the WM_PAINT, WM_SIZE and WM_DISPLAYCHANGE messages are all handled with a call to a Render method. The WM_SIZE message also calls out to a ResizeSwapChainBitmap method. These hooks are needed to let DirectX know what's happening with your window. I'll describe what these do in a moment. Finally, the Run method creates the standard Direct2D factory object, retrieving the new ID2D1Factory1 interface in this case, and optionally lets the app's window class create device-independent resources. It then creates the HWND itself and enters the message loop. The app's wWinMain function is then a simple two-liner:

```
int __stdcall wWinMain(HINSTANCE, HINSTANCE, PWSTR, int)
{
    SampleWindow window;
    window.Run();
}
```

Creating the Device

So far, most of what I've described has been a recap of what I've shown in previous columns for window management. Now I've come to the point where things get very different. The HWND render target did a lot of work for you to hide the underlying DirectX plumbing. Being a render target, the Direct2D device context still delivers the results of drawing commands to a target, but the target is no longer the HWND—rather, it's a Direct2D image. This image is an abstraction, which can literally be a Direct2D bitmap, a DXGI surface or even a command list to be replayed in some other context.

The first thing to do is to create a Direct3D device object. Direct3D defines a device as something that allocates resources, renders primitives and communicates with the underlying graphics hardware. The device consists of a device object for managing resources and a device-context object for rendering with those resources. The D3D11CreateDevice function creates a device, optionally returning pointers to the device object and device-context object. **Figure 3** shows what this might look like. I don't want to bog you down with Direct3D minutiae, so I won't describe every option in detail but instead will focus on what's relevant to Direct2D.

This function's second parameter indicates the type of device that you'd like to create. The D3D_DRIVER_TYPE_HARDWARE constant indicates that the device should be backed by the GPU for hardware-accelerated rendering. If a GPU is unavailable, then

Figure 3 Creating a Direct3D Device

```
HRESULT CreateDevice(D3D_DRIVER_TYPE const type, ComPtr<ID3D11Device> & device)
{
    UINT flags = D3D11_CREATE_DEVICE_BGRA_SUPPORT;

    #ifdef _DEBUG
    flags |= D3D11_CREATE_DEVICE_DEBUG;
    #endif

    return D3D11CreateDevice(nullptr,
                             type,
                             nullptr,
                             flags,
                             nullptr, 0,
                             D3D11_SDK_VERSION,
                             device.GetAddressOf(),
                             nullptr,
                             nullptr);
}
```

the D3D_DRIVER_TYPE_WARP constant may be used. WARP is a high-performance software rasterizer and is a great fallback. You shouldn't assume that a GPU is available, especially if you'd like to run or test your software in constrained environments such as Hyper-V virtual machines (VMs). Here's how you might use the function from **Figure 3** to create the Direct3D device:

```
ComPtr<ID3D11Device> d3device;
auto hr = CreateDevice(D3D_DRIVER_TYPE_HARDWARE, d3device);

if (DXGI_ERROR_UNSUPPORTED == hr)
{
    hr = CreateDevice(D3D_DRIVER_TYPE_WARP, d3device);
}

HR(hr);
```

Figure 3 also illustrates how you can enable the Direct3D debug layer for debug builds. One thing to watch out for is that the D3D11CreateDevice function will mysteriously fail if the debug layer isn't actually installed. This won't be a problem on your development machine, because Visual Studio would've installed it along with the Windows SDK. If you happen to copy a debug build onto a test machine, you might bump into this problem. This is in contrast to the D2D1CreateFactory function, which will still succeed even if the Direct2D debug layer isn't present.

A swap chain is a collection
of buffers used for displaying
frames to the user.

Creating the Swap Chain

The next step is to create a DXGI swap chain for the application's HWND. A swap chain is a collection of buffers used for displaying frames to the user. Typically, there are two buffers in the chain, often called the front buffer and the back buffer, respectively. The GPU presents the image stored in the front buffer while the application renders into the back buffer. When the application is done rendering it asks DXGI to present the back buffer, which basically swaps the pointers to the front and back buffers, allowing the GPU to present the new frame and the application to render the next frame. This is a gross simplification, but it's all you need to know for now.

To create the swap chain you first need to get hold of the DXGI factory and retrieve its IDXGIFactory2 interface. You can do so by calling the CreateDXGIFactory1 function, but given that you've just created a Direct3D device object, you can also use the DirectX object model to make your way there. First, you need to query for the device object's IDXGIDevice interface:

```
ComPtr<IDXGIDevice> dxdevice;
HR(d3device.As(&dxdevice));
```

Next, you need to retrieve the display adapter, virtual or otherwise, for the device:

```
ComPtr<IDXGIAdapter> adapter;
HR(dxdevice->GetAdapter(adapter.GetAddressOf()));
```

The adapter's parent object is the DXGI factory:

```
ComPtr<IDXGIFactory2> factory;
HR(adapter->GetParent(__uuidof(factory),
                    reinterpret_cast<void **>(factory.GetAddressOf())));
```



Enterprise reporting in minutes

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.

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

Download your
free 30-day trial at
DevExpress.com

DXv2

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



Figure 4 The DesktopWindow Render Method

```
void Render()
{
    if (!m_target)
    {
        CreateDevice();
        CreateDeviceSwapChainBitmap();
    }

    m_target->BeginDraw();
    static_cast<T *>(this)->Draw();
    m_target->EndDraw();

    auto const hr = m_swapChain->Present(1, 0);

    if (S_OK != hr && DXGI_STATUS_OCCLUDED != hr)
    {
        ReleaseDevice();
    }
}
```

This might seem like a lot more work than simply calling the `CreateDXGIFactory1` function, but in a moment you're going to need the `IDXGIDevice` interface, so it's really just one extra method call.

The `IDXGIFactory2` interface provides the `CreateSwapChainForHwnd` method for creating a swap chain for a given `HWND`. Before calling it, you need to prepare a `DXGI_SWAP_CHAIN_DESC1` structure describing the particular swap chain structure and behavior that you'd like. A bit of care is needed when initializing this structure, as it's what most distinguishes the various platforms on which you'll find `Direct2D`. Here's what it might look like:

```
DXGI_SWAP_CHAIN_DESC1 props = {};
props.Format = DXGI_FORMAT_B8G8R8A8_UNORM;
props.SampleDesc.Count = 1;
props.BufferUsage = DXGI_USAGE_RENDER_TARGET_OUTPUT;
props.BufferCount = 2;
```

The `Format` member describes the desired format for the swap chain buffers. I've chosen a 32-bit format with 8 bits for each color channel and alpha component. Overall, this provides the best performance and compatibility across devices and APIs. The `SampleDesc` member affects `Direct3D` image quality, as it relates to antialiasing. Generally, you'll want `Direct2D` to handle antialiasing, so this configuration merely tells `Direct3D` not to do anything. The `BufferUsage` member describes how the swap chain buffers will be used and allows `DirectX` to optimize memory management. In this case I'm indicating that the buffer will be used only for rendering output to the screen. This means that the buffer won't be accessible from the CPU, but the performance will be greatly improved as a result. The `BufferCount` member indicates how many buffers the swap chain will contain. This is typically no more than two to conserve memory, but it may be more for exceedingly high-speed rendering (although that's rare). In fact, to conserve memory, Windows Phone 8 allows only a single buffer to be used. There are a number of other swap chain members, but these are the only ones required for a desktop window. Now create the swap chain:

```
HR(factory->CreateSwapChainForHwnd(d3device.Get(),
    m_hWnd,
    &props,
    nullptr,
    nullptr,
    m_swapChain.GetAddressOf()));
```

The first parameter indicates the `Direct3D` device where the swap chain resources will be allocated, and the second is the target `HWND` where the front buffer will be presented. If all goes well, the swap chain

Figure 5 Resizing the Swap Chain

```
void ResizeSwapChainBitmap()
{
    m_target->SetTarget(nullptr);

    if (S_OK == m_swapChain->ResizeBuffers(0,
        0, 0,
        DXGI_FORMAT_UNKNOWN,
        0))
    {
        CreateDeviceSwapChainBitmap();
    }
    else
    {
        ReleaseDevice();
    }
}
```

is created. One thing to keep in mind is that only a single swap chain can be associated with a given `HWND`. This might seem obvious, or not, but it's something you need to watch out for. If this method fails, it's likely that you failed to release device-specific resources before re-creating the device after a device-loss event.

Creating the Direct2D Device

The next step is to create the `Direct2D` device. `Direct2D 1.1` is modeled more closely on `Direct3D` in that instead of simply having a render target, it now has both a device and device context. The device is a `Direct2D` resource object that's linked to a particular `Direct3D` device. Like the `Direct3D` device, it serves as a resource manager. Unlike the `Direct3D` device context, which you can safely ignore, the `Direct2D` device context is the render target that exposes the drawing commands, which you'll need. The first step is to use the `Direct2D` factory to create the device that's bound to the underlying `Direct3D` device via its `DXGI` interface:

```
ComPtr<ID2D1Device> device;

HR(m_factory->CreateDevice(dxdevice.Get(),
    device.GetAddressOf()));
```

This device represents the display adapter in `Direct2D`. I typically don't hold on to the `ID2D1Device` pointer, because that makes it simpler to handle device loss and swap chain buffer resizing. What you really need it for is to create the `Direct2D` device context:

```
HR(device->CreateDeviceContext(D2D1_DEVICE_CONTEXT_OPTIONS_NONE,
    m_target.GetAddressOf()));
```

What you now have is a `Direct2D` render target that you can use to batch up drawing commands as usual, but there's still one more step before you can do so. Unlike the `HWND` render target, which could only ever target the window for which it was created, the device context can switch targets at run time and initially has no target set at all.

Connecting the Device Context and Swap Chain

The next step is to set the swap chain's back buffer as the target of the `Direct2D` device context. The swap chain's `GetBuffer` method will return the back buffer as a `DXGI` surface:

```
ComPtr<IDXGISurface> surface;

HR(m_swapChain->GetBuffer(0, // buffer index
    __uuidof(surface),
    reinterpret_cast<void *>(surface.GetAddressOf())));
```

You can now use the device context's `CreateBitmapFromDxgiSurface` method to create a `Direct2D` bitmap to represent the `DXGI`

RACKSPACE OPEN-SOURCED THE CLOUD

DOWNLOAD. SPIN UP. BREAK OUT.

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 8,500 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



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

surface, but first you need to describe the bitmap's format and intended use. You can define the bitmap properties as follows:

```
auto props = BitmapProperties1(
    D2D1_BITMAP_OPTIONS_TARGET | D2D1_BITMAP_OPTIONS_CANNOT_DRAW,
    PixelFormat(
        DXGI_FORMAT_B8G8R8A8_UNORM, D2D1_ALPHA_MODE_IGNORE));
```

The `D2D1_BITMAP_OPTIONS_TARGET` constant indicates that the bitmap will be used as the target of a device context. The `D2D1_BITMAP_OPTIONS_CANNOT_DRAW` constant relates to the swap chain's `DXGI_USAGE_RENDER_TARGET_OUTPUT` attribute, indicating that it can be used only as an output and not as an input to other drawing operations. `PixelFormat` just describes to Direct2D what the underlying swap chain buffer looks like. You can now create a Direct2D bitmap to point to the swap chain back buffer and point the device context to this bitmap:

```
ComPtr<ID2D1Bitmap1> bitmap;

HR(m_target->CreateBitmapFromDxgiSurface(surface.Get(),
                                         props,
                                         bitmap.GetAddressOf()));

m_target->SetTarget(bitmap.Get());
```

Finally, you need to tell Direct2D how to scale its logical coordinate system to the physical display embodied by the DXGI surface:

```
float dpiX, dpiY;
m_factory->GetDesktopDpi(&dpiX, &dpiY);
m_target->SetDpi(dpiX, dpiY);
```

Rendering

The `DesktopWindow.Render` method acts as the catalyst for much of the Direct2D device management. **Figure 4** provides the basic outline of the `Render` method.

As with the HWND render target, the `Render` method first checks whether the render target needs to be created. The `CreateDevice` method contains the Direct3D device, DXGI swap chain and Direct2D device context creation. The `CreateDeviceSwapChainBitmap` method contains the code to connect the swap chain to the device context by means of a DXGI surface-backed Direct2D bitmap. The latter is kept separate because it's needed during window resizing.

The `Render` method then follows the usual pattern of bracketing draw commands with the `BeginDraw` and `EndDraw` methods. Notice that I don't bother to check the result of the `EndDraw` method. Unlike the HWND render target, the device context's `EndDraw` method doesn't actually present the newly drawn frame to the screen. Instead, it merely concludes rendering to the target bitmap. It's the job of the swap chain's `Present` method to present this to the screen, and it's at this point that any rendering and presentation issues can be handled.

Because I'm only using a simple event-driven rendering model for this window, the presentation is straightforward. If I were using an animation loop for synchronized rendering at the display's refresh frequency, things would get a lot more complicated, but I'll cover that in a future column. In this case, there are three scenarios to deal with. Ideally, `Present` returns `S_OK` and all is well. Alternatively, `Present` returns `DXGI_STATUS_OCCLUDED` indicating the window is occluded, meaning it's invisible. This is increasingly rare, as desktop composition relies on the window's presentation to remain active. One source of occlusion, however, is when the active desktop is switched. This happens most often if

a User Account Control (UAC) prompt appears or the user presses `Ctrl+Alt+Del` to switch users or lock the computer. At any rate, occlusion doesn't mean failure, so there's nothing you need to do except perhaps avoid extra rendering calls. If `Present` fails for any other reason, then it's safe to assume that the underlying Direct3D device has been lost and must be re-created. The `DesktopWindow.ReleaseDevice` method might look like this:

```
void ReleaseDevice()
{
    m_target.Reset();
    m_swapChain.Reset();
    static_cast<T*>(this)->ReleaseDeviceResources();
}
```

Here's where you can start to understand why I avoid holding on to any unnecessary interface pointers. Every resource pointer represents a reference directly or indirectly to the underlying device. Each one must be released in order for the device to be properly re-created. At a minimum, you need to hold on to the render target (so that you can actually issue drawing commands) and the swap chain (so that you can present). Related to this—and the final piece of the puzzle—is the `ResizeSwapChainBitmap` method I alluded to inside the `WM_SIZE` message handler.

Every resource pointer represents a reference directly or indirectly to the underlying device.

The HWND render target made this simple with its `Resize` method. Because you're now in charge of the swap chain, it's your responsibility to resize its buffers. Of course, this will fail unless all references to these buffers have been released. Assuming you aren't directly holding on to any, this is simple enough. **Figure 5** shows you how.

In this case, the only reference to the swap chain's back buffer that the `DesktopWindow` holds is the one held indirectly by the device context render target. Setting this to a nullptr value releases that final reference so that the `ResizeBuffers` method will succeed. The various parameters just tell the swap chain to resize the buffers based on the window's new size and keep everything else as it was. Assuming `ResizeBuffers` succeeds, I simply call the `CreateDeviceSwapChainBitmap` method to create a new Direct2D bitmap for the swap chain and hook it up to the Direct2D device context. If anything goes wrong, I simply release the device and all of its resources; the `Render` method will take care of re-creating it all when the time comes.

You now have everything you need to render in a desktop window with Direct2D 1.1! And that's all I have room for this month. Join me next time as I continue to explore Direct2D. ■

KENNY KERR is a computer programmer based in Canada, an author for *Pluralsight* and a Microsoft MVP. He blogs at kennykerr.ca and you can follow him on Twitter at twitter.com/kennykerr.

THANKS to the following technical expert for reviewing this article:
Worachai Chaoweeraprasit (Microsoft)

DirectShow and Media Foundation SDK for .NET, Win32/64



MPEG-2 TRANSPORT STREAM, DVR, KLV, RTSP

MPEG4, MPEG2, H.264, H.263, AAC, AC3 AND 100+CODECS

MULTIPLEXERS AND DEMULTIPLEXERS

MKV, MXF, MP4, AVI, WMV AND MPG

IIS SMOOTH STREAMING, UDP, RTP, ENCRYPTION, WMV

CAPTURE, PLAYBACK, CONVERSION, COMPRESSION

CLOUD SDK FOR DISTRIBUTED PROCESSING

PLAY, CONVERT, AUTHOR AND BURN DVDs AND CDs



TECHMENTOR

CONFERENCES

YEARS OF IT

15

EDUCATION

DEEP DIVE LAS VEGAS

Sept 30-Oct 4, 2013

The Tropicana, Las Vegas

Redefining

~~GET INSIDE~~ THE IT CLASSROOM

NEW FORMAT! Every session dives deep.

- ✦ New format based on your feedback.
- ✦ Each session will last at least 3 hours.
- ✦ 🖱️ Featuring "hands-on" training with your own laptop.
- ✦ You won't leave this conference wanting more!





NEW FORMAT! Every session dives deep.

Register today and get inside the redefined IT classroom! More information, more detail, and more knowledge is waiting for you – TechMentor **DEEP DIVE** is a conference you won't want to miss.

Save \$300!



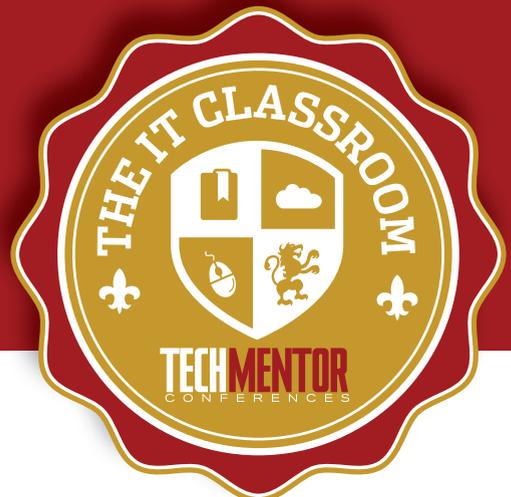
Register before July 31
Use Promo Code TMLVMAY2

TechMentor tracks include:

- ✦ Security
- ✦ PowerShell
- ✦ System Center
- ✦ Exchange
- ✦ Active Directory & Identity
- ✦ Cisco + Networking
- ✦ MCSE / MCSA Certification Prep
- ✦ SharePoint

Connect with TechMentor:

-  @techmentorevent
-  Search "TechMentor"
-  JOIN "TechMentor" GROUP





Geo-Protection for Video Blobs Using a Node.js Media Proxy

Windows Azure provides three useful options for its cloud storage system: tables, blobs, and queues, all backed up by a redundant and resilient global infrastructure. We looked at tables in our last column (msdn.microsoft.com/magazine/dn166928); now we'll focus on blobs, in particular on how to resolve certain security issues that can arise when you use them.

Blobs are used to store binary data, and Windows Azure gives you two types to choose from: page blobs, which are used for random data access and can be up to 1TB; and block blobs, which are optimized for uploading and streaming purposes and can contain up to 200GB of data. Blobs are used as the foundation for many services in Windows Azure, including OS-formatted disks and video assets. In terms of performance, the throughput target for each blob is up to 60 MB/s. **Figure 1** shows how blob storage is structured, partitioned and accessed.

Blobs are used as the foundation for many services in Windows Azure, including OS-formatted disks and video assets.

Blobs are exposed to the world via the HTTP and HTTPS protocols and can be either public or private. When a blob is marked as private, a policy-based Shared Access Signature (a URL) is used to grant access to a specific blob, or to any blob within a specified container for a specific period of time. Once the URL for a specific blob is known, however, the only way to prevent access to it is by modifying, expiring or deleting the corresponding security policy.

This condition presents a challenge for some use cases, such as restricting access to video assets when the URL of the blob needs

to be publicly shared for streaming purposes. Two common levels of protection are often required: geographical access, which restricts the countries or regions where the video can be played; and referrer access, which restricts the domains or Web sites where the video can be embedded. This is particularly important for media corporations that acquire digital rights to broadcast events (such as the Olympics, or the soccer World Cup) in specific countries, as well as for marketing agencies creating location-based advertising.

In this article, we'll show you how to create a reverse proxy server in Windows Azure that offers a solution to these security requirements, as shown in **Figure 2**.

For our solution, we'll use Node.js, a powerful server-side JavaScript-based platform we discussed in a previous column (see "Real-World Scenarios for Node.js in Windows Azure" at msdn.microsoft.com/magazine/jj991974). The reason for using Node.js is simple: It requires minimal memory and CPU use while supporting thousands of connections per server. And by deploying our application to Windows Azure, we can scale as needed, based on traffic and demand. Here's the basic flow we'll follow:

1. Capture the original request to embed a video from the Web server that was generated by a remote client.
2. Identify the referrer page and validate that the domain is authorized to embed the video.
3. Identify the country from which the request originated and verify that it's authorized.
4. If all the criteria are met, stream the video to the client.
5. If not all criteria are met, stream a video showing an error message.

We'll be using the following cloud resources to make our solution work:

- Windows Azure Media Services to generate the video assets to be used in code.
- Windows Azure Tables to store the list of authorized countries and referrers.
- MongoDB (mongodb.org) to host the IP geolocation database.
- Windows Azure Web Sites to host the Node.js reverse proxy and the sample Web page.

Hosting Video Content in Windows Azure

Before you start, you need to upload and encode some video content in the cloud for testing purposes. You can do this easily by signing in to the Windows Azure portal at manage.windowsazure.com and clicking Media Services. (For more information on Media

TRY OUT WINDOWS AZURE FOR FREE FOR 90 DAYS

Experience Windows Azure for free for three months without any obligation. Get 750 small compute hours, a 1GB SQL Azure database and more.

bit.ly/VzrCq0

Code download available at archive.msdn.microsoft.com/mag201305AzureInsider.



 **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

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

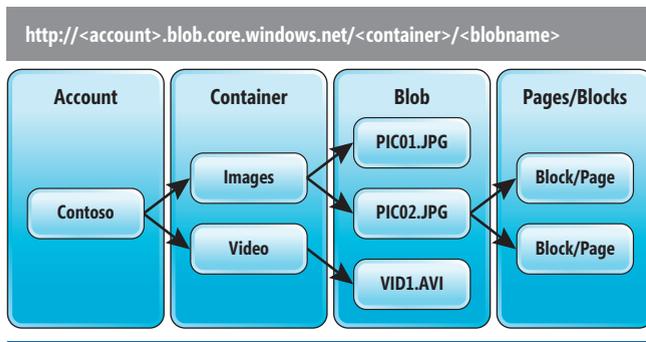


Figure 1 Windows Azure Blob Storage Concepts

Services, please refer to our June 2012 article, “Democratizing Video Content with Windows Azure Media Services,” at msdn.microsoft.com/magazine/jj133821.) If you don’t have a subscription, you can request a trial account at bit.ly/YCNEd3. For this example, we uploaded and encoded three videos, as shown in Figure 3.

Blobs are exposed to the world via the HTTP and HTTPS protocols and can be either public or private.

The Publish URL link associated with each video allows them to be played and embedded in other pages. Our proxy server will use these links to stream the media content to viewers around the world, after validating the location and page from which the request originated.

Storing Validation Information in Windows Azure Table Storage

The next step is to create a few tables in Windows Azure Storage that will help with the validation process. As we explained in our previous column, these tables are based strictly on key-value pairs, but they serve our purpose for this project. Figure 4 describes the structure of the tables.

You can create these tables in Windows Azure Storage using one of the free tools available for download. We particularly like Azure Storage Explorer from Neudesic, which can be downloaded

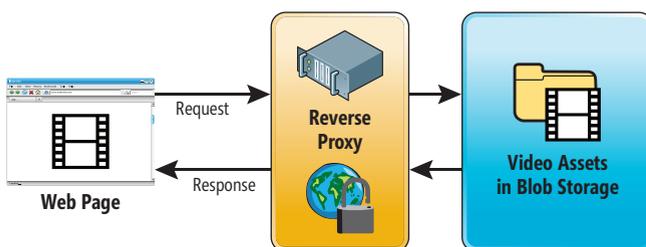


Figure 2 Providing Geo-Protection for Video Assets Using a Reverse Proxy Server

from CodePlex at bit.ly/H3r0C. For basic functionality, you’ll need to insert at least one entity into the proxycountries and proxyreferers tables, with PartitionKey=“undefined” and RowKey=“true.” This lets you test your Node.js media proxy locally. In last month’s article, we discussed the importance of selecting the correct partition and row keys for the best query performance.

Preparing the Geolocation Database

There are a few companies offering databases and services for geolocation purposes. MaxMind is one of them, and because the company provides a GeoLite version that can be used under the Creative Commons license, we decided to include it in our project. The CSV file can be downloaded from bit.ly/W5Z7qA. This database allows us to identify the country where the video request is coming from, based on the IP address.

Our next decision involved where to host this database in the cloud. Because we’ll need to perform a range search on this table (not natively supported by Windows Azure Table Storage), we opted to use MongoDB, an open source document-oriented (JSON) database engine developed and supported by 10gen. MongoDB supports multiple indexes and complex queries, which is ideal for our solution. The good news is that there are a few companies offering this database as a service, including MongoLab, available in the Windows Azure Store. You can also sign up for an account at mongolab.com, selecting Windows Azure as your hosting provider. Make sure to select the same datacenter where you created the Windows Azure Storage tables.

When you’ve created your MongoDB database, you can access it using a URL similar to this:

```
mongodb://{username}:[password]!@{server_name}.mongolab.com:{port_number}/{database_name}
```

Once the URL for a specific blob is known, however, the only way to prevent access to it is by modifying, expiring or deleting the corresponding security policy.

In order to import the MaxMind CSV file into your database, simply download the MongoDB tools from mongodb.org/downloads. Once you’ve installed them on your computer, run the following command:

```
mongoimport -h {servername}.mongolab.com:{port_number} -d {database_name} -c {collection_name} -u {username} -p {password} --file (MaxMind CSV file) --type csv --headerline
```

Now you can run queries against the geolocation database.

Reviewing the Source Code

If you haven’t already, please download the source code for this article from archive.msdn.microsoft.com/mag201305AzureInsider. The code consists of three different files: server.js, config.json and package.json.



garden care cost calculator

Equipment	Maker/supplier	Quantity	Cost/tool	Cost/yr
Shears		4	\$12.51	\$50.04
		4	\$7.89	\$31.56
		3	\$15.88	\$47.64
		6	\$10.00	\$60.00
		4	\$22.00	\$88.00
		8	\$81.00	\$648.00
		22	\$14.99	\$329.78
			\$164.27	\$1,255.02

Grand total for this year	
Total	\$3,774.17

Total costs by area	
Tools	\$1,255.02
Ornaments & misc	\$469.90
Total	\$1,724.92
Lawn care	\$90.00
Plant and bed care	\$161.00
Shrubs and tree care	\$215.00
Total	\$466.00
Vegetables	\$193.00
Flowers	\$661.50
Seeds and bulbs	\$90.25
Trees and shrubs	\$638.50
Total	\$1,583.25
Grand total	\$3,774.17



Item	Company or Service	Times/mth	Cost/yr
Lawn care		2	\$40.00
Fertilizers		1	\$35.00
Aeration		2	\$15.00
Total lawn care			\$90.00
Mowing/raking		2	\$25.00
Total lawn care		1	\$60.00
Plant and bed care		1	\$12.50
Mulch		3	\$14.00
Tilling by machine			
Topsoil			
Weeding/spray			

Garden plants and seeding plan		
Item	Supplier	When to plant
Vegetables		
Green Beans		40
Corn		35
Flowers		
Roses		50

Available for



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

Spread

ComponentOne®
 a division of GrapeCity®

Download your free trial @
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.

The main part of the code is in `server.js`, where you'll see a few modules defined in the first lines (they'll be automatically downloaded and installed later from `npmjs.org`):

1. `Request`: Simplifies the process of sending and streaming requests to external sites.
2. `azure`: Provides access to Windows Azure Storage, including tables.
3. `url`: Facilitates the parsing of URL strings.
4. `mongodb`: Provides access to MongoDB databases.
5. `nconf`: Simplifies the process of setting and retrieving application settings.

Also, a few variables are set in the first portion of the code, following this format:

```
var port = process.env.PORT || nconf.get("PORT_NUMBER");
```

This allows their values to be retrieved either from the Windows Azure Web Sites configuration parameters (once deployed to the cloud), or from the local `config.json` file (in case the parameter can't be found in the Windows Azure environment). Finally, a client is created for the Windows Azure Table, a default agent for the subsequent requests is defined and a placeholder for the error-logging object is instantiated to null, as shown in the following initialization code:

```
// Create Windows Azure Storage Table client
console.log('Connecting to Windows Azure Table Service');

var tableService = azure.createTableService(storageServiceName,
    storageKey, storageServiceUrl);
// Create custom agent for requests; number of sockets can be tweaked

var agent = new http.Agent();
agent.maxSockets = 1000;

// Placeholder for errorEntity object
var errorEntity = null;
```

Once the initialization process has been performed, it's time to create an HTTP server in Node.js, with its corresponding callback function. The basic waterfall structure in high-level pseudocode—derived from the asynchronous, event-based nature of Node.js—looks like this:

```
Create HTTP Server (request, response)
(callback to)
Find the page where the video is hosted, using the http-referer header value
Find the origin IP address by using the x-forwarded-for header value
Split the request URL to find the video-friendly name and encoding
(callback to)
    Query the proxyreferrers table for validation
    (callback to)
        Query the MongoDB database to find the request country
        (callback to)
            Query the proxycountries table for validation
            (callback to)
                Stream the video using the request function
```

You'll find details in the included source code, but we want to highlight a few aspects:

1. The origin IP address is converted to an integer number using a formula defined by MaxMind (bit.ly/15xuuJE). This number is then used in the following query to find the corresponding country in the MongoDB database:

```
{ convstart: { $lte: ipValue }, convend: { $gte: ipValue } }
```

The reason for using Node.js is simple: it requires minimal memory and CPU use while supporting thousands of connections per server.

The `ipValue` is compared against the range defined between the `convstart` (less than or equal to) and the `convend` (greater than or equal to) columns provided in the MaxMind table. If no value is found, the country is set to "undefined."

2. The main streaming process occurs in the request command inside the function `streamVideo`, which looks like this:

```
request({options}, {callback_function}).pipe(resp);
```

Thanks to the `request` module, which simplifies this process, data is sent back to the client as it's received, being piped directly back to the client. This turns our reverse proxy into a fast, efficient application. The keep-alive header (found in the full code among the options) is extremely important, improving server performance by keeping the client/server connection open across multiple requests to the server.

Testing the Reverse Proxy Server Locally

You can test the application locally by downloading the Node.js installer from nodejs.org. Open a command prompt window

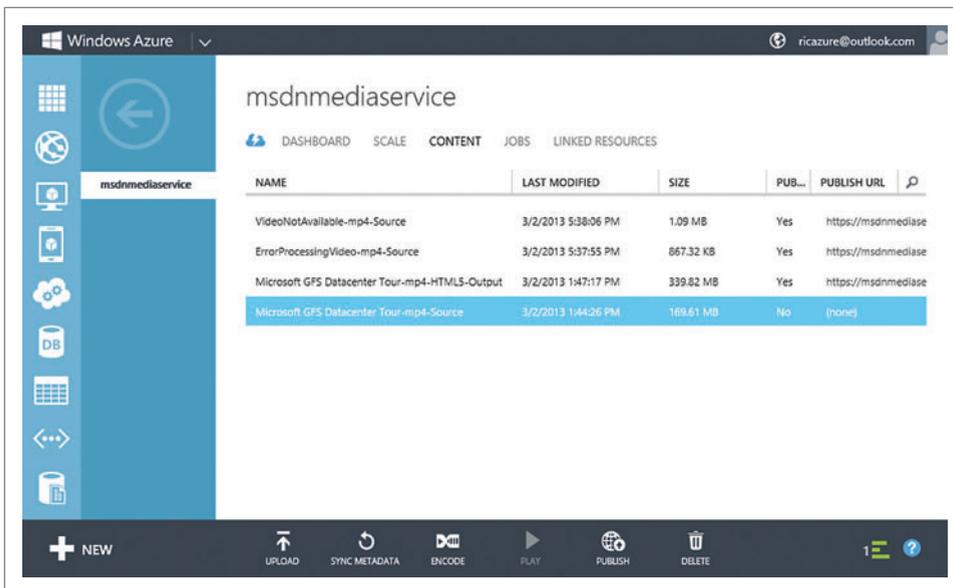


Figure 3 Video Assets Used in the Example

DEVELOPED FOR INTUITIVE USE

DynamicPDF—Comprehensive PDF Solutions for .NET Developers

ceTe Software's DynamicPDF products provide real-time PDF generation, manipulation, conversion, printing, viewing, and much more. Providing the best of both worlds, the object models are extremely flexible but still supply the rich features you need as a developer. Reliable and efficient, the high-performance software is easy to learn and use. If you do encounter a question with any of our components, simply contact ceTe Software's readily available, industry-leading support team.



DynamicPDF

WWW.DYNAMICPDF.COM



TRY OUR PDF SOLUTIONS FREE TODAY!

www.DynamicPDF.com/eval or call 800.631.5006 | +1 410.772.8620

ceTe software

Figure 4 Storage Table Structure for the Video Content

proxycountries	PartitionKey: country abbreviation RowKey: "true" or "false" (for access)
proxyreferrers	PartitionKey: domain where the video is hosted RowKey: "true" or "false" (for access)
proxyvideos	PartitionKey: friendly name for the video asset RowKey: encoding format URL: published URL for the video asset in Media Services
proxyrejects	PartitionKey: "error" or "reject" RowKey: category for the error or reject Description: details about the error or reject

in the folder where you copied the source files and execute the command "npm install," which will install the required modules. Next, set the different options inside the config.json file, including the Windows Azure Table Storage information, the MongoDB connection string and the location of the videos containing the error messages. You can now start the solution in the same command prompt windows by typing "node server.js."

Deploying the Reverse Proxy to Windows Azure Web Sites

In order to deploy the reverse proxy to Windows Azure, create a new Web site in the management portal and enable Git deployment, which lets you check in code from any Git repository (including local ones). You'll find specific instructions on how to accomplish this at bit.ly/KCQo9V. The source code files already include package.json, which defines the required modules and engine version for the solution to work in Windows Azure. Once the Web site is created, you can deploy directly from the command prompt window by navigating to the folder where the files are located and executing the following commands:

```
> git init
> git add .
> git commit -m "Initial commit"
> git push (git URL from Windows Azure portal) master
```

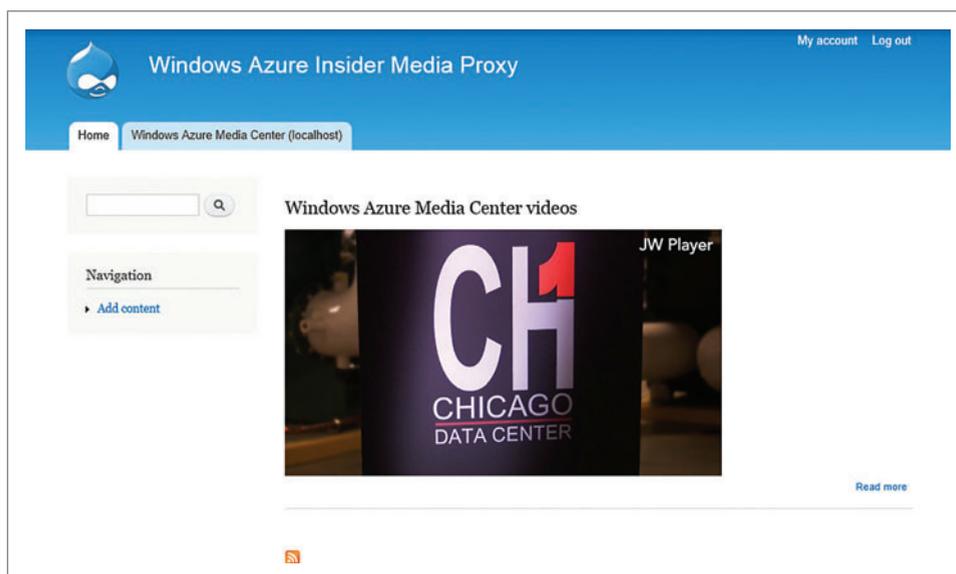


Figure 5 Sample Web Site Pointing to the Node.js Proxy Server

After you provide the required credentials, the code will be deployed to Windows Azure.

We've created a Web site at rvideo.azurewebsites.net where you can see a sample page that includes the popular HTML5-compatible JW Player, pointing to our reverse proxy (Figure 5). If you're located in the United States, you'll be able to watch the video; otherwise, you'll get a warning about the video not being available in your country or region. There's also a tab on the Web site pointing to a localhost server, in order to facilitate local debugging.

By deploying our application to Windows Azure, we can scale as needed, based on traffic and demand.

Wrapping Up

In this example, we've shown how to combine multiple cloud components to add an extra layer of geolocation security to videos stored in Windows Azure blob storage. Intercepting, routing and manipulating HTTP requests is one of the use cases where Node.js shines. We've also shown how simple it is to interact with Windows Azure services such as Table Storage, as well as third-party providers in the Windows Azure Store such as MongoLab. Finally, by deploying our solution to Windows Azure Web Sites, we can scale its capacity based on demand and traffic. ■

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. You can read his blog at blogs.msdn.com/b/brunoterkaly.

RICARDO VILLALOBOS is a seasoned software architect with more than 15 years of experience designing and creating applications for companies in the supply chain management industry. Holding different technical certifications, as well as 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. You can read his blog at blog.ricardovillalobos.com.

BRUNO AND RICARDO jointly present at large industry conferences. They encourage readers of Windows Azure Insider to contact them for availability. Bruno can be reached at bterkaly@microsoft.com and Ricardo can be reached at Ricardo.Villalobos@microsoft.com.

THANKS to the following technical expert for reviewing this article: David Makogon (Microsoft)

Parse JSON Strings in Windows Runtime Components

Craig Shoemaker

Windows Store apps built with JavaScript let anyone with HTML and JavaScript skills build native Windows apps, but JavaScript isn't always the best choice to solve every problem. Some behavior in your apps might be better implemented in a more object-oriented fashion using C#, Visual Basic or C++. Also, certain aspects of your code might be a candidate for reuse among multiple Windows Runtime (WinRT) components that require data from the UI layer. Passing data from JavaScript into WinRT components and back to the UI is important to understand in either of these situations.

On the Web, data is often passed from the client to the server and back in the form of JSON objects. In different contexts, frameworks like ASP.NET Web Forms and ASP.NET MVC include fea-

tures such as model binders or at least some sort of "auto-magic" handling on the server side for parsing JSON objects. WinRT components have objects with affordances for parsing JSON, but the support is low-level, and more streamlined interaction requires some explicit handling on your part.

This article demonstrates how to reliably parse JSON strings passed into WinRT components in order to hydrate strongly typed objects and return a result back up to the UI.

Restrictions for Interacting with WinRT components

Before discussing the particulars of parsing JSON objects, you must first get acquainted with requirements and restrictions for interacting with WinRT components. The MSDN help topic, "Creating Windows Runtime Components in C# and Visual Basic" (bit.ly/WgBBai), details what's needed for WinRT components in regard to declaring method parameter and return types. Allowed types are largely composed of primitive types and a handful of collection types, so attempting to pass a raw JSON object into a component isn't allowed. The best way to pass a JSON object into a managed component is to first serialize the JSON object (using the `JSON.stringify` method), as strings are fully supported in these classes.

This article discusses:

- Restrictions when using WinRT components
- Parsing JSON objects in managed code
- Extending `JsonObject`
- Adding support for factories and arrays
- Supporting asynchronous interaction
- Using JavaScript promises

Technologies discussed:

Windows Runtime components, JavaScript, JSON

Code download available at:

archive.msdn.microsoft.com/mag201305JSON

GET HELP BUILDING YOUR WINDOWS STORE APP!

Receive the tools, help and support you need to develop your Windows Store apps.

bit.ly/XLjOrx

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

Figure 1 GetStringValue Extension Method Implementation

```
public static string GetStringValue(this JsonObject jsonObject, string key)
{
    IJsonValue value;
    string returnValue = string.Empty;

    if (jsonObject.ContainsKey(key))
    {
        if (jsonObject.TryGetValue(key, out value))
        {
            if (value.ValueType == JsonValueType.String)
            {
                returnValue = jsonObject.GetNamedString(key);
            }
            else if (value.ValueType == JsonValueType.Number)
            {
                returnValue = jsonObject.GetNamedNumber(key).ToString();
            }
            else if (value.ValueType == JsonValueType.Boolean)
            {
                returnValue = jsonObject.GetNamedBoolean(key).ToString();
            }
        }
    }

    return returnValue;
}
```

Parsing JSON Objects in Managed Code

The Windows.Data.Json namespace includes a number of different classes designed to work with JSON objects in a strongly typed manner, including the JsonValue, JsonArray and JsonObject classes. The JsonValue class represents a JSON value that's exposed in the form of a string, number, Boolean, array or object (see more on this at bit.ly/14AcTmF). Parsing a JSON string requires that you pass the raw string to JsonValue, which then is able to return an instance of JsonObject.

The JsonObject class represents a full JSON object and includes methods to manipulate the source object. Through the JsonObject class, you can add and remove members, extract data from members, iterate over each member and even serialize the object again. More details about JsonObject are available at bit.ly/WDWZkG.

The JsonArray class represents a JSON array, which, again, includes a host of methods for controlling the array, such as iteration and addition and removal of array elements. More information about the interface of the JsonArray class is available at bit.ly/XVUZo1.

As an example to see how to begin using these classes, consider the following JSON object in JavaScript:

```
{
  firstName: "Craig"
}
```

Before attempting to pass this object to a WinRT component, you must serialize the object into a string using the JSON.stringify function. Notice what happens after the object is serialized—the very same object is represented as follows:

```
"{
  '_backingData': {
    'firstName': 'Craig'
  },
  'firstName': 'Craig',
  'backingData': {
    'firstName': 'Craig'
  }
}"
```

This might come as a surprise to you, as the very same function call in a Web browser simply serializes the object into a string without

adding any members to the object. This change in the structure of the JSON string affects how you extract data from the object.

The first step in reading this data in a WinRT component is to attempt to parse the incoming string as a JsonValue instance. If that parse succeeds, then you can request the JsonObject from the root JsonValue instance. In this case the JsonValue is the root object as created by the call to the stringify function, and JsonObject grants you access to the original object you started with in JavaScript.

The following code depicts how, once the JsonObject is available, you can use the GetNamedString method to extract the value of the "firstName" member into a variable:

```
JsonValue root;
JsonObject jsonObject;
string firstName;

if (JsonValue.TryParse(jsonString, out root))
{
    jsonObject = root.GetObject();

    if (jsonObject.ContainsKey("firstName"))
    {
        firstName = jsonObject.GetNamedString("firstName");
    }
}
```

A similar approach is used to access Boolean and numeric members—where GetNamedBoolean and GetNamedNumber methods are available. The next step is to implement extension methods of the JsonObject in order to make it easy to access JSON data.

Figure 2 GetBooleanValue Extension Method Implementation

```
public static bool? GetBooleanValue(this JsonObject jsonObject, string key)
{
    IJsonValue value;
    bool? returnValue = null;

    if (jsonObject.ContainsKey(key))
    {
        if (jsonObject.TryGetValue(key, out value))
        {
            if (value.ValueType == JsonValueType.String)
            {
                string v = jsonObject.GetNamedString(key).ToLower();
                if (v == "1" || v == "true")
                {
                    returnValue = true;
                }
                else if (v == "0" || v == "false")
                {
                    returnValue = false;
                }
            }
            else if (value.ValueType == JsonValueType.Number)
            {
                int v = Convert.ToInt32(jsonObject.GetNamedNumber(key));
                if (v == 1)
                {
                    returnValue = true;
                }
                else if (v == 0)
                {
                    returnValue = false;
                }
            }
            else if (value.ValueType == JsonValueType.Boolean)
            {
                returnValue = value.GetBoolean();
            }
        }
    }

    return returnValue;
}
```

SQL SERVER + FATDB



SOME THINGS JUST GO WELL TOGETHER...

Like FatDB and SQL Server. Create stunning new applications that leverage legacy data from SQL Server, and add new capabilities by integrating unstructured data from FatDB, building complex queries with Map Reduce, or using LINQ as a common query engine, all wrapped up using .NET as a common development framework. Improve performance. Lower TCO. Build powerful applications.

Sandwich anyone? Try out FatDB for free at FATCLOUD.COM/DOWNLOAD



Figure 3 GetDoubleValue Extension Method Implementation

```
public static double? GetDoubleValue(this JsonObject jsonObject, string key)
{
    IJsonValue value;
    double? returnValue = null;
    double parsedValue;

    if (jsonObject.ContainsKey(key))
    {
        if (jsonObject.TryGetValue(key, out value))
        {
            if (value.ValueType == JsonValueType.String)
            {
                if (double.TryParse(jsonObject.GetNamedString(key), out parsedValue))
                {
                    returnValue = parsedValue;
                }
            }
            else if (value.ValueType == JsonValueType.Number)
            {
                returnValue = jsonObject.GetNamedNumber(key);
            }
        }
    }

    return returnValue;
}
```

Extension Methods for JsonObject

The default implementation of the `JsonObject` class provides some low-level behavior that's greatly enhanced with some simple methods that can handle imperfect formatting and can avoid exceptions if members don't exist in the source. In other words, objects created in JavaScript are bound to encounter formatting or structural problems that could cause exceptions. Adding the following extension methods to the `JsonObject` class will help mitigate these problems.

The first extension method to add is called `GetStringValue`. **Figure 1** shows the implementation, which first checks to make sure the member exists on the object. In this use, the `key` parameter is the name of the JSON object property. After the member is known to exist, then the method `TryGetValue` is used to attempt to access the data from the `JsonObject` instance. If the value is successfully found, it's returned as an object implementing the `IJsonValue` interface.

Figure 4 PersonFactory Create Method That Accepts a JsonValue

```
public static Person Create(JsonValue personValue)
{
    Person person = new Person();

    JsonObject jsonObject = personValue.GetObject();

    int? id = jsonObject.GetIntegerValue("id");
    if (id.HasValue)
    {
        person.Id = id.Value;
    }

    person.FirstName = jsonObject.GetStringValue("firstName");
    person.LastName = jsonObject.GetStringValue("lastName");

    bool? isOnWestCoast = jsonObject.GetBooleanValue("isOnWestCoast");
    if (isOnWestCoast.HasValue)
    {
        person.IsOnWestCoast = isOnWestCoast.Value;
    }

    return person;
}
```

The `IJsonValue` interface includes the read-only `ValueType` property, which exposes the given value of the `JsonValueType` enumeration that signifies the object's data type. After interrogating `ValueType`, the appropriately typed method is used to extract the data from the object.

The `GetStringValue` method includes awareness of Booleans and numeric values in order to guard against malformed JSON objects. You might choose to make your implementation stricter and forgo parsing or throw an error if the JSON object isn't strictly formatted for the expected type, but the code in my example makes the parse operation flexible and guards against errors.

The next extension method, seen in **Figure 2**, is the implementation to extract Boolean values. In this case, Boolean values expressed as strings (for example, "1" or "true" for a true value, and so on) and numbers (for example, "1" for true and "0" or false) are supported in the `GetBooleanValue` method.

The numeric-based extension methods are set up to return nullable types, so in this case, `GetDoubleValue` returns a nullable double. The corrective behavior in this case attempts to convert strings into the possible corresponding numeric values (see **Figure 3**).

Because the built-in method to extract numbers in the `JsonObject` class returns a double, and often data values are expressed as integers, the following code shows how the `GetIntegerValue` method wraps the `GetDoubleValue` method and converts the result into an integer:

```
public static int? GetIntegerValue(this JsonObject jsonObject, string key)
{
    double? value = jsonObject.GetDoubleValue(key);
    int? returnValue = null;

    if (value.HasValue)
    {
        returnValue = Convert.ToInt32(value.Value);
    }

    return returnValue;
}
```

Adding Factory Support

Now that the `JsonObject` class is extended to include some higher-level support for data extraction into primitive types, the next step is to use this support in factory classes that are responsible for taking incoming JSON strings and returning an instance of a hydrated domain object.

Figure 5 PersonFactory CreateList Method

```
public static IList<Person> CreateList(string peopleJson)
{
    List<Person> people = new List<Person>();
    JsonArray array = new JsonArray();

    if (JsonArray.TryParse(peopleJson, out array))
    {
        if (array.Count > 0)
        {
            foreach (JsonValue value in array)
            {
                people.Add(PersonFactory.Create(value));
            }
        }
    }

    return people;
}
```

Does your Team do more than just track bugs?

Free Trial and Single User FreePack™ available at www.alexcorp.com

Alexsys Team® does! Alexsys Team 2 is a multi-user Team management system that provides a powerful yet easy way to manage all the members of your team and their tasks - including defect tracking. Use Team right out of the box or tailor it to your needs.



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

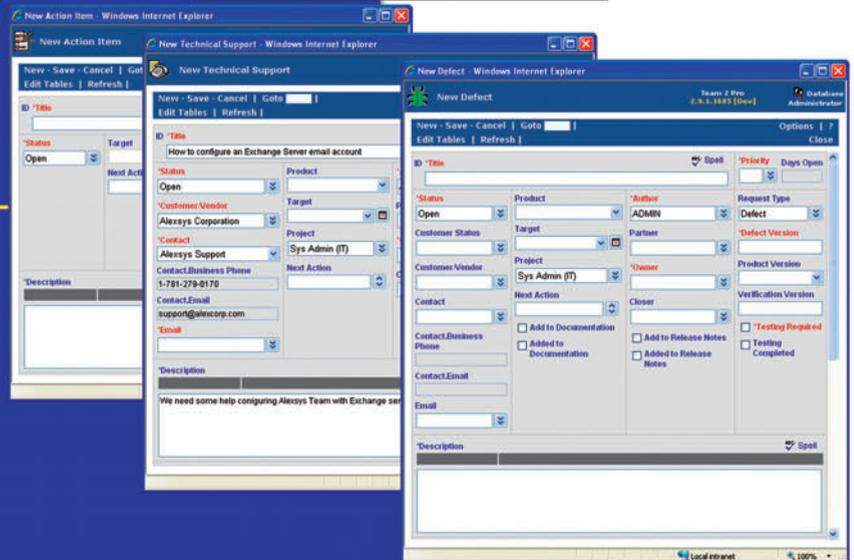
- Full Windows 7 Support
- Windows Single Sign-on
- System Audit Log
- Trend Analysis
- Alternate Display Fields for Data Normalization
- Lookup Table Filters
- XML Export
- Network Optimized for Enterprise Deployment

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. FreePack™ includes a free single user Team Pro and Team-Web license. Need more help? Give us a call at 1-888-880-ALEX (2539).

Team 2 works with its own standard database, while Team Pro works with Microsoft SQL, MySQL, and Oracle Servers. Team 2 works with Windows 7/2008/2003/Vista/XP. Team-Web works with Internet Explorer, Firefox, Netscape, Safari, and Chrome.

Figure 6 ContactsManager Implementation (Without Async Support)

```
using System.Collections.Generic;

public sealed class ContactsManager
{
    private string AddContact(string personJson)
    {
        Person person = PersonFactory.Create(personJson);

        return string.Format("{0} {1} is added to the system.",
            person.FirstName,
            person.LastName);
    }

    private string AddContacts(string personJson)
    {
        IList<Person> people = PersonFactory.CreateList(personJson);

        return string.Format("{0} {1} and {2} {3} are added to the system.",
            people[0].FirstName,
            people[0].LastName,
            people[1].FirstName,
            people[1].LastName);
    }
}
```

Figure 7 JSON Data Source

```
var _model = {
    contact: {
        id: 1000,
        firstName: "Craig",
        lastName: "Shoemaker"
    },
    contacts: [
        {
            id: 1001,
            firstName: "Craig",
            lastName: "Shoemaker",
            isOnWestCoast: "true"
        },
        {
            id: 1002,
            firstName: "Jason",
            lastName: "Beres",
            isOnWestCoast: "0"
        }
    ]
}
```

Figure 8 View Model That Uses ContactsManager

```
var _vm = {
    ViewModel: WinJS.Binding.as({
        model: _model,
        contactMsg: "",
        contactsMsg: ""
    }),

    addContact: function () {
        var mgr = ParseJSON.Utility.ContactsManager();
        var jsonString = JSON.stringify(_vm.ViewModel.model.contact);
        mgr.addContactAsync(jsonString).done(function (response) {
            _vm.ViewModel.contactMsg = response;
        });
    },

    addContacts: function () {
        var mgr = ParseJSON.Utility.ContactsManager();
        var jsonString = JSON.stringify(_vm.ViewModel.model.contacts);
        mgr.addContactsAsync(jsonString).done(function (response) {
            _vm.ViewModel.contactsMsg = response;
        });
    }
});
```

The following code depicts how a person is modeled in the system:

```
internal class Person
{
    public int Id { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public bool? IsOnWestCoast { get; set; }
}
```

The following code shows the Create method in the PersonFactory class that accepts a string:

```
public static Person Create(string jsonString)
{
    JsonValue json;
    Person person = new Person();

    if (JsonValue.TryParse(jsonString, out json))
    {
        person = PersonFactory.Create(json);
    }

    return person;
}
```

Figure 4 shows a Create method that accepts a JsonValue. These Create methods used together are responsible for taking in a raw string and returning an instance of the Person class with the expected data in each member. The methods are separated and overloaded in order to provide support for JSON arrays, which is explained in the next section.

Adding Array Support

Sometimes your data comes in the form of object arrays rather than just single objects. In this case you must attempt to parse the string as an array by leveraging the JsonArray class. **Figure 5** shows how the incoming string is parsed into an array and then each item is passed to the Create method for final parsing into the model. Notice that a new instance of the Person list is first created so that in case the string doesn't parse into an object array, the result is an empty array, which helps avoid unexpected exceptions.

Adding Support Classes

The next step is to create an object that's responsible for using the factory class and doing something interesting with the resulting model instances. **Figure 6** demonstrates how both individual and JSON array strings are consumed and then manipulated as strongly typed objects.

Supporting Asynchronous Interaction

Calls of this nature into methods of WinRT components should be done asynchronously, as the JSON messages have the potential to grow to an arbitrary size, which might introduce latency into your application.

The following code includes the method added to the ContactsManager to support asynchronous access to the AddContact method:

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Windows.Foundation;

public IAsyncOperation<string> AddContactAsync(string personJson)
{
    return Task.Run<string>(() =>
    {
        return this.AddContact(personJson);
    }).AsAsyncOperation();
}
```

The AddContactAsync method accepts the JSON string and then starts a Task, which does the work of running the AddContact

method. Once the Task is complete, a response is sent up to the JavaScript promise as facilitated by the IAsyncOperation interface support. The full source of the ContactsManager class with asynchronous support for both AddContact and AddContacts can be found in the accompanying code download.

Keeping Promises in JavaScript

The final piece of the puzzle is to use the ContactsManager class in JavaScript and invoke calls to the class using the promise pattern. The approach used in this example is to implement a view model that passes the modeled data to the WinRT component and then waits for a response. The data used to pass to the component is defined in **Figure 7**, which includes a single JSON object as well as an array.

The view model, as seen in **Figure 8**, includes a member for the model and members for messages that are returned from the WinRT component. The Windows Library for JavaScript (WinJS) binding framework is used to bind messages returned from the response to the HTML elements. A full listing of the page module is available in the accompanying code download so you can see how all the individual parts fit together.

Note that if you want to bind the addContact or addContacts functions to a button during data binding, you must run the WinJS.Utilities.requireSupportedForProcessing function, passing in a reference to the function on your view model.

The final step is to add the appropriate elements and attributes to the HTML to support binding. A div element acts as the main binding container for the binding elements and is marked by setting data-win-bind-source="Application.Pages.Home.ViewModel." Then the header elements are bound to their data members by supplying the appropriate values to the data-win-bind attributes:

```
<section aria-label="Main content" role="main">
  <div data-win-bind-source=
    "Application.Pages.Home.ViewModel">
    <h2 data-win-bind="innerText: contactMsg"></h2>
    <hr />
    <h2 data-win-bind="innerText: contactsMsg"></h2>
  </div>
</section>
```

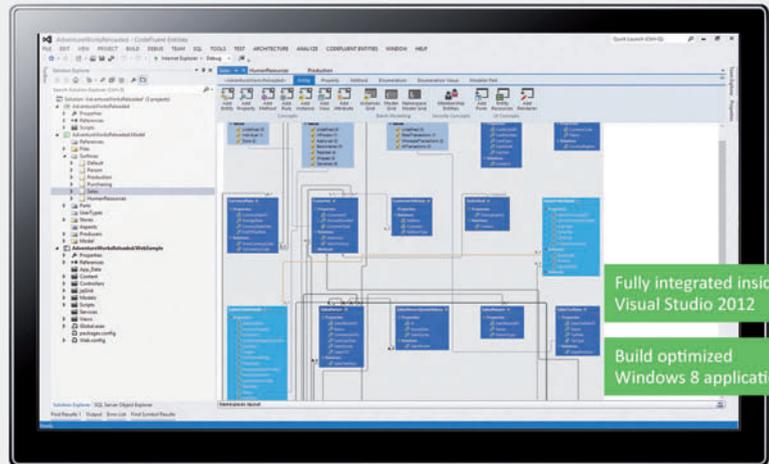
So there you have it! Building Windows Store apps with JavaScript gives you an opportunity to leverage your existing skills from the Web to build native, modern UI apps, but there are a number of distinguishing factors between the two platforms. Low-level support for parsing JSON data is available through the Windows.Data.Json namespace, but you can add richer support with a few extensions to existing objects. ■

CRAIG SHOEMAKER is a software developer, podcaster, blogger and technical evangelist. He's also a Code Magazine, MSDN and Pluralsight author. In his spare time, he enjoys looking for a haystack in which to hide his prized needle collection. You can reach him on Twitter at twitter.com/craigshoemaker.

THANKS to the following technical experts for reviewing this article: Christopher Bennage (Microsoft), Kraig Brockschmidt (Microsoft) and Richard Fricks (Microsoft)

Save your time!

Stop writing repetitive code and focus on what matters



Fully integrated inside Visual Studio 2012

Build optimized Windows 8 applications

Generate rock-solid foundations for your .NET applications



Benefit from out-of-the-box advanced features

“ A remarkable product that adds features where all the ‘junior’ equivalents fall short. Things like hassle-free schema updates, up-casting, enum & null management, security, full data-binding including grids with pagination, performance, interface support, custom stored procedures within a wide range of architectures.

Basically all the ‘add-ons’ you discover you need when you start developing a real world app based on any code generator. ”

Boris Bosnjak, Developer, Dreamquest, Canada

Get a license worth \$1499 for free until June 30th!

Go to www.softfluent.com/forms/msdn-special-offer

Tools for developers, by developers.

More information at www.softfluent.com
Contact us at info@softfluent.com



Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS



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 65 sessions and workshops, you'll get an all-access look at the Microsoft Platform and practical, unbiased, developer training at Visual Studio Live! Redmond.

EVENT SPONSOR
Microsoft

PLATINUM SPONSOR
 esri

SUPPORTED BY
 Visual Studio

msdn
magazine

Visual Studio
MAGAZINE

PRODUCED BY
 1105 MEDIA

REDMOND, WA | AUGUST 19-23, 2013

MICROSOFT CAMPUS

REGISTER TODAY AND SAVE \$400

USE PROMO CODE REDMAY2



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

TOPICS WILL INCLUDE:

- Web Development
- Azure / Cloud Computing
- Cross-Platform Mobile
- Data Management
- SharePoint / Office
- Windows 8 / WinRT
- Visual Studio 2012 / .NET 4.5
- SQL Server

CONNECT WITH VISUAL STUDIO LIVE!

 twitter.com/vslive – @VSLive

 facebook.com – Search “VSLive”

 linkedin.com – Join the “Visual Studio Live” group!

vslive.com/redmond

Leverage Multiple Code Frameworks with One ASP.NET

Jeffrey T. Fritz

In 2001, when Microsoft introduced the Microsoft .NET Framework and, with it, a new technology called ASP.NET, Web developers embraced it for building sites using a forms-based framework. This framework, known as Web Forms, stood the test of time for eight years, with enhancements and changes to support an evolving Web environment. Creating a Web application during that time was a simple choice, with a New Project dialog that presented four ASP.NET options, as shown in **Figure 1**. Most of us ignored the ASP.NET Mobile Web Site and Web Control Library projects and built only ASP.NET Web Application projects. If you needed Web services, you'd add a SOAP-based service to an existing Web site with an .asmx file.

This article discusses:

- The four component frameworks of ASP.NET
- The "One ASP.NET" concept
- Setting up the sample project and shared layout
- Configuring MVC to use a Web Forms master page
- Configuring the Web Forms-based search page
- Adding a Web API controller
- Adding a SignalR hub

Technologies discussed:

ASP.NET, Visual Studio 2012, Telerik RadControls for ASP.NET AJAX

Code download available at:

archive.msdn.microsoft.com/mag201305OneASP

In early 2009, the ASP.NET landscape changed dramatically with the introduction of Model-View-Controller (MVC). With the promise of no more viewstate, page event lifecycle or postback events to handle, developers flocked to the new framework. I was one of them, intrigued by the potential of this more-testable Web technology. We had to find ways to justify to our managers and cost centers the budget for switching applications to MVC, and many developers worked through the steps to get MVC content presented in the same application as an existing Web Forms app. Things worked very well with MVC for several years, and then the Web grew up a little. ASP.NET needed to evolve again.

In 2012, Microsoft delivered two new frameworks to add to the ASP.NET toolkit: Web API and SignalR. Both of these frameworks bring something special to the environment, and each is unique in its own way:

- Web API provides an MVC-like experience for developers to deliver content intended for machine interpretation. There's no UI, and transactions occur in a RESTful manner. Content types are negotiated, and Web API can automatically format content as JSON or XML, based on the HTTP headers submitted to a Web API endpoint.
- SignalR is the new "real-time Web" delivery model from Microsoft. This technology opens up the client-server communications channel to allow for immediate, rich communications from the server to the client. The content delivery model in SignalR reverses our normal expectations, as the server calls the client to interact with content.

Consider the trade-offs already seen between Web Forms and MVC with those of Web API and MVC, as shown in **Figure 2**.

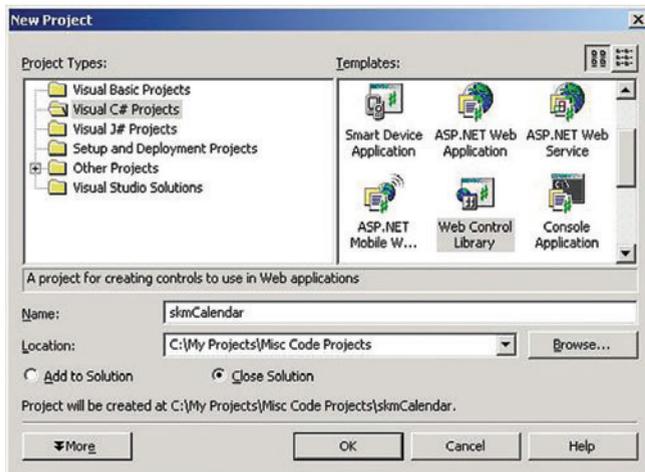


Figure 1 The Original New Project ASP.NET Choices in Visual C#

Productivity involves features that let you develop and deliver a solution quickly. Control is the extent to which you can affect the bits being transmitted over the network to your connected users. UI indicates whether you can use the framework to deliver a complete UI. Finally, Real Time suggests how well the framework presents content in a timely fashion that could be perceived as an immediate update.

Now, in 2013, when I open my copy of Visual Studio and attempt to start an ASP.NET project, I'm faced with the dialogs shown in Figure 3 and Figure 4.

There are some tricky questions in those windows. What type of project should I start with? What template is going to get me closest to my solution, fastest? And what if I want to include some components of each template? Can I build a mobile application with some server controls and a Web API?

Do I Have to Choose Just One Approach?

Do I have to choose just one approach? The short answer is no, you don't have to select only one of these frameworks to build a Web application. There are available techniques that allow you to use Web Forms and MVC together, and contrary to the dialog windows presented, Web API and SignalR can easily be added as features to a Web application. Remember, all ASP.NET content is rendered through a series of HttpHandlers and HttpModules. As long as the correct handlers and modules are referenced, you can build a solution with any of these frameworks.

This is the heart of the "One ASP.NET" concept: Don't choose just one of these frameworks—

Figure 2 Benefits of Each ASP.NET Component Framework

Framework	Productivity	Control	UI	Real Time
Web Forms	●		●	
MVC		●	●	
Web API	●	●		
SignalR				●

build your solution with the parts of each that best suit your needs. You've got a number of choices; don't limit yourself to just one.

So you can see this in action, I'm going to put together a small Web application that will have a unified layout, a search screen and a create screen for a list of products. The search screen will be powered by Web Forms and Web API, and show live updates from SignalR. The create screen will be generated automatically by MVC templates. I'm also going to make the Web Forms look great by using a third-party control library, the Telerik RadControls for ASP.NET AJAX. A trial version of these controls is available at bit.ly/15o20ab.

Setting up the Sample Project and Shared Layout

To get started, I need to create a project using the dialog shown in Figure 3. While I could choose an empty or Web Forms application, the most encompassing solution to choose is the MVC application. Starting with an MVC project is a great choice because you get all of the tooling from Visual Studio to help you configure your models, views and controllers, as well as the ability to add Web Forms objects anywhere in the project file structure. It's possible to add the MVC tooling back into an existing Web application by changing some of the XML content in the .csproj file. This process can be automated by installing the NuGet package called AddMvc3ToWebForms.

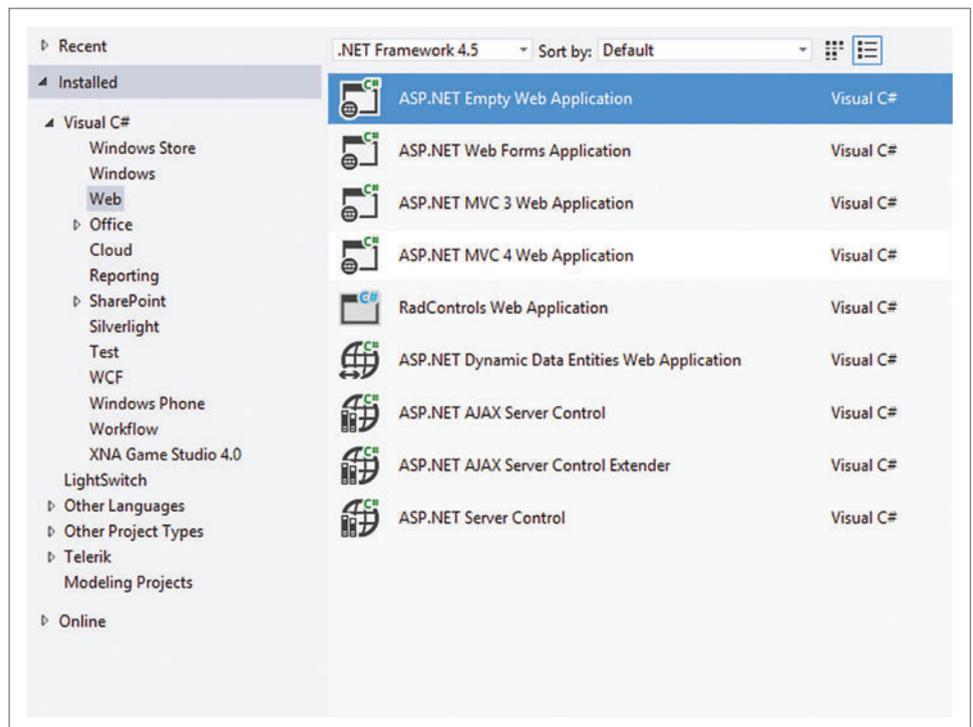


Figure 3 New Web Project in Visual Studio 2012

To configure the Telerik controls for use in this project, I need to do some hacking in Web.config to add the HttpHandlers and HttpModules that would normally be configured in a standard Telerik RadControls project. First, I'll add a couple of lines to define the Telerik AJAX controls UI skin:

```
<add key="Telerik.Skin" value="WebBlue" />
</appSettings>
```

Next, I'll add the Telerik tag prefix:

```
<add tagPrefix="telerik" namespace="Telerik.Web.UI" assembly="Telerik.Web.UI" />
</controls>
```

I'll make the minimal additions needed to the Web.config Http-Handlers for the Telerik controls:

```
<add path="Telerik.Web.UI.WebResource.axd" type="Telerik.Web.UI.WebResource"
verb="*" validate="false" />
</httpHandlers>
```

And, finally, I'll make the additions to the Web.config Handlers for the Telerik controls:

```
<system.WebServer>
<validation validateIntegratedModeConfiguration="false" />
<handlers>
<remove name="Telerik_Web_UI_WebResource_axd" />
<add name="Telerik_Web_UI_WebResource_axd"
path="Telerik.Web.UI.WebResource.axd"
type="Telerik.Web.UI.WebResource" verb="*" preCondition="integratedMode" />
</handlers>
```

Now I want to create a layout page for this project, so I'll create a Web Forms site.master page in the Views | Shared folder. For this site layout, I want to add a standard logo and menu to all pages. I'll add a logo image by simply dragging the image onto my layout. Next, to add a great cascading menu to the layout, I'll drag a RadMenu from my controls toolbox onto the designer, just below the image. From the designer surface, I can quickly build out my menu by right-clicking on the menu control and selecting Edit Items to get the window shown in **Figure 5**.

The two menu items I want to focus on are under Products—Search and New. For each of these items, I've set the NavigateUrl property and text as follows:

```
<telerik:RadMenuItem Text="Products">
<Items>
<telerik:RadMenuItem Text="Search" NavigateUrl="~/Product/Search" />
<telerik:RadMenuItem Text="New" NavigateUrl="~/Product/New" />
</Items>
</telerik:RadMenuItem>
```

With the menu configured, I now have a problem where I've defined my layout using Web Forms, and need to host MVC content. This is not a trivial problem, but it is a problem that can be solved.

Bridging the Divide—Configuring MVC to Use a Web Forms Master Page

Like most of you, I prefer to keep things simple. I want to share my defined layout for this project between Web Forms and MVC. There's a well-documented technique devised by Matt Hawley that demonstrates how to use a Web Forms master page with MVC Razor-based views (bit.ly/ehWY3H). I'm going to use that technique in this project. To create this bridge, I'll configure a simple Web Forms view called RazorView.aspx that references the master page:

```
<%@ Page Language="C#" AutoEventWireup="true"
MasterPageFile="~/Views/Shared/Site.Master"
Inherits="System.Web.Mvc.ViewPage<dynamic>" %>
<%@ Import Namespace="System.Web.Mvc" %>

<asp:Content id="bodyContent" runat="server" ContentPlaceHolderID="body">
<% Html.RenderPartial((string)ViewBag._ViewName); %>
</asp:Content>
```

In order for my MVC controllers to use this view and allow their Razor-based views to be executed, I need to extend each control-

ler to route the view content appropriately. This is accomplished through an extension method that reroutes the model, ViewData and TempData appropriately through RazorView.aspx, as shown in **Figure 6**.

With this method constructed, I can easily route all of the MVC actions through the master page. The next step is to set up the ProductsController so that products can be created.

MVC and the Create Product Screen

The MVC part of this solution is a fairly standard MVC approach. I defined a simple model object called BoardGame in the Models folder of my project, as shown in **Figure 7**.

Next, using the standard MVC tooling in Visual Studio, I create an empty ProductController. I'll add a Views | Product folder, then right-click on the Product folder and choose View from the Add menu. This view will support the creation of new board games, so I'll create it with the options shown in **Figure 8**.

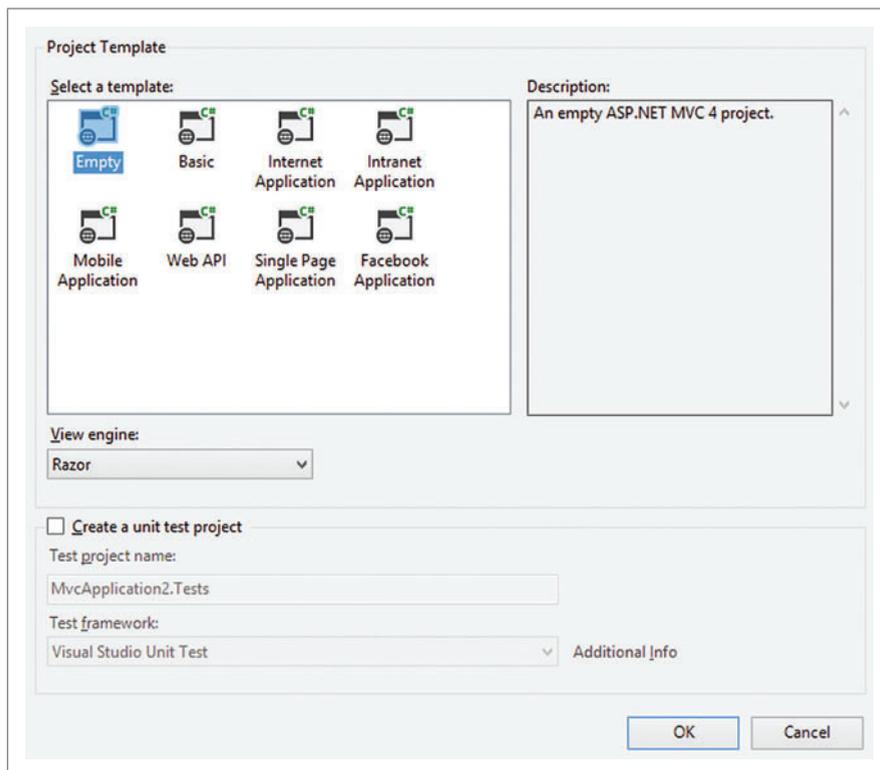


Figure 4 New Project Template Dialog in Visual Studio 2012

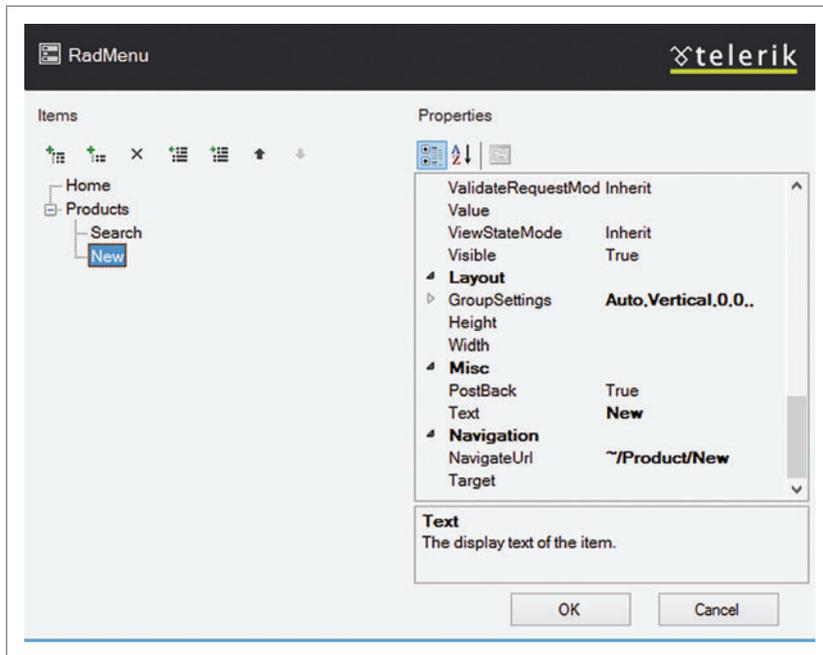


Figure 5 Telerik RadMenu Configuration Window

Thanks to the MVC tooling and templates, I don't need to change a thing. The view created has labels and validation, and can use my master page. **Figure 9** shows how to define the New action in the ProductController.

This syntax should be familiar to MVC developers, as the only change is to return a RazorView instead of a View. The `_Products` object is a static, read-only collection of dummy products that are defined in this controller (instead of using a database in this sample):

```
public static readonly List<BoardGame> _Products = new List<BoardGame>()
{
    new BoardGame() {Id=1, Name="Chess", Price=9.99M},
    new BoardGame() {Id=2, Name="Checkers", Price=7.99M},
    new BoardGame() {Id=3, Name="Battleship", Price=8.99M},
    new BoardGame() {Id=4, Name="Backgammon", Price= 12.99M}
};
```

Configuring the Web Forms-Based Search Page

I want my users to be able to access the product search page with a URL that doesn't look like it's a Web Forms URL, and is friendly

Figure 6 The RazorView Extension Method to Reroute MVC Views Through a Web Forms Master Page

```
public static ActionResult RazorView(this Controller controller,
string viewName = null, object model = null)
{
    if (model != null)
        controller.ViewData.Model = model;

    controller.ViewBag._ViewName = !string.IsNullOrEmpty(viewName)
        ? viewName
        : controller.RouteData.GetRequiredString("action");

    return new ViewResult
    {
        ViewName = "RazorView",
        ViewData = controller.ViewData,
        TempData = controller.TempData
    };
}
```

to search. With the release of ASP.NET 2012.2, this can now be configured easily. Simply open the `App_Start/RouteConfig.cs` file and call `EnableFriendlyUrls` to switch on this capability:

```
public static void RegisterRoutes(
RouteCollection routes)
{
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

    routes.EnableFriendlyUrls();

    routes.MapRoute(
        name: "Default",
        url: "{controller}/{action}/{id}",
        defaults: new { controller = "Home", action =
            "Index", id = UrlParameter.Optional }
    );
}
```

With this line added, ASP.NET will route requests for `/Product/Search` to the physical file residing at `/Product/Search.aspx`

Next, I want to configure a search page that shows a grid of the current products and their stock levels. I'll create a `Product` folder in my project and add a new Web Form to it named `Search.aspx`. In this file, I'll remove all markup except the `@Page` directive and set

the `MasterPageFile` to the `Site.Master` file defined previously. To display my results, I'll choose the Telerik RadGrid so I can quickly configure and display the result data:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Search.aspx.cs"
    Inherits="MvcApplication1.Product.Search"
    MasterPageFile="~/Views/Shared/Site.Master" %>

<asp:Content runat="server" id="main" ContentPlaceHolderID="body">

    <telerik:RadGrid ID="searchProducts" runat="server" width="500"
        AllowFilteringByColumn="True" CellSpacing="0" GridLines="None"
        AllowSorting="True">
```

I want to share my defined layout for this project between Web Forms and MVC.

The grid will automatically render columns bound to it on the server side and provide sorting and filtering capabilities. However, I'd like to make this more dynamic. I want to see data delivered and managed on the client side. In this model, data could be transmitted and bound with no server-side code in the Web Forms. To accomplish this, I'm going to add a Web API that will deliver and perform the data operations.

Adding Web API to the Mix

I'll use the standard Project | Add New menu to add a Web API controller named `ProductController` to a folder named `api` in my project. This helps me keep clear the differences between the MVC controllers and the API controllers. This API is going to do one thing—deliver data for my grid in JSON format and support

Figure 7 The BoardGame Object

```
public class BoardGame
{
    public int Id { get; set; }

    public string Name { get; set; }

    [DisplayFormat(DataFormatString="$0.00")]
    public decimal Price { get; set; }

    [Display(Name="Number of items in stock"), Range(0,10000)]
    public int NumInStock { get; set; }
}
```

OData queries. To accomplish this in Web API, I'm going to write a single Get method and decorate it with the Queryable attribute:

```
[Queryable]
public IQueryable<dynamic> Get(ODataQueryOptions options)
{
    return Controllers.ProductController._Products.Select(b => new
    {
        Id = b.Id,
        Name = b.Name,
        NumInStock = b.NumInStock,
        Price = b.Price.ToString("$0.00")
    }).AsQueryable();
}
```

This code returns my collection of BoardGame objects in the static list with a small bit of formatting. By decorating the method with [Queryable] and returning a queryable collection, the Web API framework will automatically handle and process OData filter and sort commands. The method also needs to be configured with the input parameter ODataQueryOptions in order to handle the filter data submitted by the grid.

To configure the grid on Search.aspx to consume this new API, I need to add some client settings to the page markup. In this grid control, I define the client databinding with a ClientSettings element and a DataBinding setting. The DataBinding setting lists

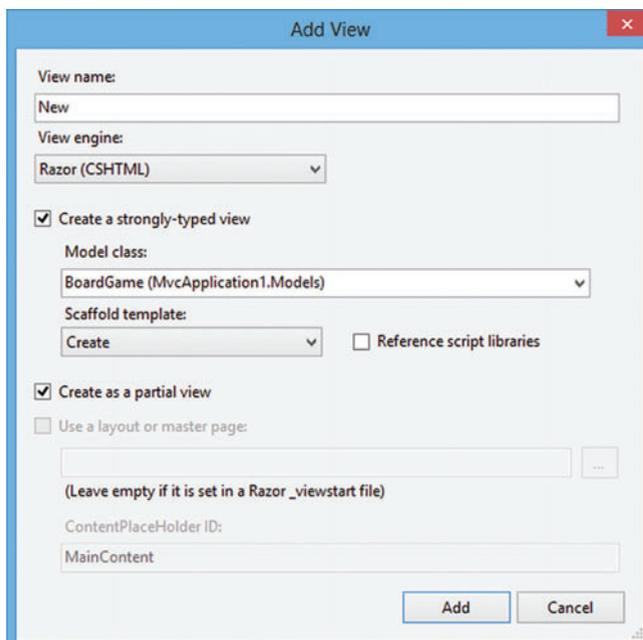


Figure 8 Creating the "New" View

Figure 9 ProductController Routing Through the RazorView

```
public ActionResult New()
{
    return this.RazorView();
}

[HttpPost]
public ActionResult New(BoardGame newGame)
{
    if (!ModelState.IsValid)
    {
        return this.RazorView();
    }

    newGame.Id = _Products.Count + 1;
    _Products.Add(newGame);

    return Redirect("~/Product/Search");
}
```

the location of the API, the response format type and the name of the controller to query, as well as the OData query format. With these settings, and a definition of the columns to present in the grid, I can run the project and see the grid bound to the data in the _Products dummy list of data, as Figure 10 shows.

Activating the Grid with Real-Time Data

The last piece of the puzzle is the ability to show real-time changes of the stock levels as products are shipped and received. I'm going to add a SignalR hub to transmit updates and present those new values on the search grid. To add SignalR to my project, I need to issue the following two NuGet commands:

```
Install-Package -pre Microsoft.AspNet.SignalR.SystemWeb
Install-Package -pre Microsoft.AspNet.SignalR.JS
```

These commands will install the ASP.NET server components for hosting within the IIS Web server and make the JavaScript client libraries available to the Web Forms.

The SignalR server-side component is called a Hub, and I'll define mine by adding a class called StockHub to a folder called

Figure 10 Complete Formatting Source of the Grid

```
<telerik:RadGrid ID="searchProducts" runat="server" width="500"
    AllowFilteringByColumn="True" CellSpacing="0" GridLines="None"
    AllowSorting="True" AutoGenerateColumns="false"
>
    <ClientSettings AllowColumnsReorder="True"
        ReorderColumnsOnClient="True"
        ClientEvents-OnGridCreated="GridCreated">
        <Scrolling AllowScroll="True" UseStaticHeaders="True"></Scrolling>
        <DataBinding Location="/api" ResponseType="JSON">
            <DataService TableName="Product" Type="OData" />
        </DataBinding>
    </ClientSettings>
    <MasterTableView ClientDataKeyNames="Id" DataKeyNames="Id">
        <Columns>
            <telerik:GridBoundColumn DataField="Id" HeaderStyle-Width="0"
                ItemStyle-Width="0"></telerik:GridBoundColumn>
            <telerik:GridBoundColumn DataField="Name" HeaderText="Name"
                HeaderStyle-Width="150" ItemStyle-Width="150">
            </telerik:GridBoundColumn>
            <telerik:GridBoundColumn ItemStyle-CssClass="gridPrice"
                DataField="Price"
                HeaderText="Price" ItemStyle-HorizontalAlign="Right">
            </telerik:GridBoundColumn>
            <telerik:GridBoundColumn DataField="NumInStock"
                ItemStyle-CssClass="numInStock"
                HeaderText="# in Stock"></telerik:GridBoundColumn>
        </Columns>
    </MasterTableView>
</telerik:RadGrid>
```

FLOW TYPE LAYOUT REPORTING



Visit us: June 3-6, New Orleans
TechEd
North America 2013



Reuse MS Word documents or templates as your reporting templates.



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



Easy database connection with master-detail nested blocks.



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



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



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

TX
TEXT CONTROL®
word processing components

 **Visual Studio**
Microsoft

Partner

US +1 855-533-8398

EU +49 421-4270671-0

WWW.TEXTCONTROL.COM

Figure 11 The SignalR Hub Server-Side Component

```
public class StockHub : Hub
{
    public static readonly Timer _Timer = new Timer();
    private static readonly Random _Rdm = new Random();

    static StockHub()
    {
        _Timer.Interval = 2000;
        _Timer.Elapsed += _Timer_Elapsed;
        _Timer.Start();
    }

    static void _Timer_Elapsed(object sender, ElapsedEventArgs e)
    {
        var products = ProductController._Products;
        var p = products.Skip(_Rdm.Next(0, products.Count())).First();

        var newStockLevel = p.NumInStock + _Rdm.Next(-1 * p.NumInStock, 100);
        p.NumInStock = newStockLevel;

        var hub = GlobalHost.ConnectionManager.GetHubContext<StockHub>();
        hub.Clients.All.setNewStockLevel(p.Id, newStockLevel);
    }
}
```

Hubs in my Web project. The StockHub is required to descend from the Microsoft.AspNet.SignalR.Hub class. I defined a static System.Timers.Timer to allow the application to simulate the changing of stock levels. For this simulation, every 2 seconds (when the timer Elapsed event handler triggers), I'll randomly set the stock level of a randomly chosen product. Once the product stock level is set, I'll notify all attached clients by executing a method on the client called setNewStockLevel, which is shown in Figure 11.

For this hub's data to be accessible from the server, I need to add a line to RouteConfig indicating the presence of the hub. By calling

Figure 12 SignalR Client-Side Code to Activate the Grid

```
<script src="/Scripts/jquery.signalR-1.0.0-rc2.min.js"></script>
<script src="/signalr/hubs"></script>
<script type="text/javascript">

    var grid;

    $(document).ready(function() {

        var stockWatcher = $.connection.stockHub;

        stockWatcher.client.setNewStockLevel = function(id, newValue) {

            var row = GetRow(id);

            var orgColor = row.css("background-color");

            row.find("td.numInStock").animate({
                backgroundColor: "#FFFD5"
            }, 1000, "swing", function () {
                row.find("td.numInStock").html(newValue).animate({
                    backgroundColor: orgColor
                }, 1000);
            });
        };

        $.connection.hub.start();

    })

</script>
```



Figure 13 The Search Interface with Grid Generated by Web API and Maintained by SignalR

routes.MapHubs in the RegisterRoutes method of RouteConfig, I complete the server-side configuration of SignalR.

Next, the grid needs to listen for these events from the server. To accomplish this, I need to add some JavaScript references to the SignalR client library installed from NuGet and the code generated from the MapHubs command. The SignalR service connects and exposes the setNewStockLevel method on the client using the code shown in Figure 12.

In the jQuery ready event handler, I establish a reference called stockWatcher to the StockHub using the \$.connection.stockHub syntax. I then define the setNewStockLevel method on the client property of the stockWatcher. This method uses some other JavaScript helper methods to traverse the grid, find the row with the appropriate product and change the stock level with a fancy color animation provided by the jQuery UI, as shown in Figure 13.

Wrapping Up

I've demonstrated how to build an ASP.NET MVC project and add a Web Forms layout, third-party AJAX controls and Web Forms routing to it. I generated the UI with MVC tooling and activated the content with Web API and SignalR. This project used features from all four of the ASP.NET frameworks to present a cohesive interface, taking advantage of the best features of each component. You can do the same. Don't choose just one ASP.NET framework for your next project. Instead, choose to use them all. ■

JEFFREY T. FRITZ is a developer evangelist for Telerik with more than 15 years of experience building large-scale multi-tenant Web applications in the Software as a Service model. He's an INETA speaker and maintains a blog at csharpfritz.com. You can find him on Twitter at twitter.com/csharpfritz and can reach him at jeff.fritz@telerik.com.

THANKS to the following technical experts for reviewing this article: Scott Hanselman (Microsoft) and Scott Hunter (Microsoft)

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



SpreadsheetGear

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

Migrating Legacy .NET Libraries to Target Modern Platforms

Josh Lane

One of the greatest strengths of the Microsoft .NET Framework is the wide variety of third-party open source and commercial libraries that target the platform. It's a testament to the maturity of the .NET development ecosystem that you not only have great APIs to choose from within the .NET Framework itself, but also thousands of non-Microsoft libraries for serving HTTP requests, drawing grids in a desktop application, storing structured data on the file system, and everything in between. Indeed, a quick perusal of popular .NET code repositories shows more than 32,000 projects on CodePlex, more than 5,000 code samples on code.msdn.microsoft.com and more than 10,000 unique packages on the NuGet Gallery!

The emergence of new software platforms such as Windows Phone 8 and Windows 8 has the potential to breathe new life into these tried-and-true codebases. .NET libraries that have served you well for years on the desktop and server can prove equally (and sometimes more) useful in these new environments—provided you're willing to expend the migration effort needed to target these new

platforms. Traditionally such tasks can be difficult and tedious, but while care and explicit planning are still needed to ensure success, Visual Studio 2012 has several features that minimize potential difficulties and maximize opportunities for reuse across platforms.

In this article I'll explore the challenges found during a real-world forward migration of the open source Sterling NoSQL object-oriented database (OODB) project. I'll walk you through a brief overview of the library, share the migration obstacles encountered and the solutions to overcome them, and then wrap up by considering some patterns and best practices advice you can exploit in your own library migration efforts.

What Is Sterling?

Sterling is a lightweight NoSQL data storage library that provides fast, indexed retrieval of .NET class instances. Updates, deletes, backup and restore, truncation, and the like are also supported, though like other NoSQL technologies it doesn't provide a general-purpose, SQL language-based query facility. Instead, the notion of “query” consists of a set of distinct, ordered operations:

- First, retrieve a collection of predefined keys or indexes mapped to lazy-loaded class instances.
- Next, index into the key collection to do initial fast filtering of the entire possible result set.
- Finally, use standard LINQ to Objects queries against the now-filtered key-value pairs to further refine the results.

Clearly the usage model for Sterling (and for similar NoSQL databases) differs from the one offered by traditional relational databases like SQL Server. The lack of a formal, distinct query language seems particularly strange to newcomers. In fact, this is

This article discusses:

- The Sterling NoSQL data storage library
- Converting synchronous APIs to async versions
- Easing cross-platform development with Portable Class Libraries
- Using familiar patterns such as Inversion of Control to take advantage of platform-specific APIs

Technologies discussed:

Microsoft .NET Framework, Windows Phone 8, Windows 8, Visual Studio 2012, C#

presented by NoSQL proponents as a strength, given the potential complexity and overhead associated with mapping inputs and outputs between the world of query and the world of code. In a NoSQL solution such as Sterling, there is no mapping because query and code are one and the same.

A full treatment of the ins and outs of Sterling is beyond the scope of this article (see Jeremy Likness' article, "Sterling for Isolated Storage on Windows Phone 7," at msdn.microsoft.com/magazine/hh205658 for further details), but I'll highlight some key advantages and trade-offs to keep in mind:

- It has a small footprint (around 150K on disk) and is well-suited for in-process hosting.
- It works out-of-the-box with the standard range of serializable .NET types.
- The set of concepts needed for basic functionality is small; you can be up and running with Sterling in as little as five lines of C# code.
- Traditional database features such as granular security, cascading updates and deletes, configurable locking semantics, Atomicity, Consistency, Isolation, Durability (ACID) guarantees, and so on are not supported in Sterling. If you need these features, you should consider a full relational engine like SQL Server.

The creator of Sterling (my Wintellect colleague, Jeremy Likness) intended from the outset that it target multiple platforms; he created binaries for the .NET Framework 4, Silverlight 4 and 5, and Windows Phone 7. So when considering the work needed to update Sterling to target the .NET Framework 4.5, Windows Phone 8 and Windows Store apps, I knew the architecture would lend itself to such an effort, but I didn't know exactly what the project would entail.

The advice I outline in this article is a direct result of my experience updating Sterling to target the .NET Framework 4.5, Windows Phone 8 and Windows Store apps. While some of the details of my journey are specific to the Sterling project, many others are relevant to a wide range of projects and porting efforts in the Microsoft ecosystem.

Challenges and Solutions

Reflecting on the obstacles I faced as I ported Sterling to the new target platforms, a few broad categories emerged under which I can both lump the problems I encountered and provide more general guidance for anyone undertaking this kind of project.

Accommodate Divergent Design Philosophies The first set of potential problems is somewhat philosophical in nature, though it has a real impact on the overall migration effort you'll expend. Ask yourself: "To what extent does the architecture and design of the library I want to migrate align with the common patterns and usage models of the new target platforms?"

This isn't an easy question to resolve, and neat, one-size-fits-all answers are elusive. Your clever, custom Windows Forms layout manager might be difficult or impossible to port to Windows Presentation Foundation (WPF). The APIs are certainly distinct, but it's the very different core design philosophies and notions of control management and positioning of those two worlds that are likely to trip you up in the long run. As another example, custom UI

input controls that work well for the classic keyboard-and-mouse input style might yield a poor UX for touch-based environments like Windows Phone or Windows 8. The mere desire to migrate a codebase forward isn't enough; there must be an underlying design compatibility between old and new platforms, plus a willingness to reconcile whatever minor differences that do exist. In the case of Sterling, I had a few such dilemmas to work through.

The most prominent design issue was the mismatch between the synchronous Sterling data update API and the expected asynchronous nature of such behavior in libraries that target Windows Phone 8 and Windows Store apps. Sterling was designed several years ago in a world where asynchronous APIs were rare, and the tools and techniques for creating them were crude at best.

User expectations with regard to responsive software design have increased dramatically in the last several years.

Here's a typical Save method signature in pre-migration Sterling:

```
object Save<T>(T instance) where T : class, new()
```

What's important to note here is that this method executes synchronously; that is, no matter how long it takes to save the instance argument, the caller is blocked, waiting for the method to complete. This can result in the usual array of thread-blocking issues: unresponsive UIs, dramatically reduced server scalability and so forth.

User expectations with regard to responsive software design have increased dramatically in the last several years; none of us want to put up with a UI frozen for several seconds while waiting for a save operation to complete. In response, API design guidelines for new platforms like Windows Phone 8 and Windows 8 mandate that public library methods such as Save be asynchronous, non-blocking operations. Thankfully, features like the .NET Task-based Asynchronous Pattern (TAP) programming model and C# async and await keywords make this easier now. Here's the updated signature for Save:

```
Task<Object> SaveAsync<T>(T instance) where T : class, new()
```

Now Save returns immediately, and the caller has an awaitable object (the Task) to use for eventual harvesting of the result (in this case, the unique key of the newly saved instance). The caller isn't blocked from doing other work while the save operation completes in the background.

To be clear, all I've shown here are the method signatures; the conversion from synchronous to async implementation under the hood required additional refactoring and a switch to the asynchronous file APIs for each target platform. For instance, the synchronous implementation of Save used a BinaryWriter to write to the file system:

```
using ( BinaryWriter instanceFile = _fileHelper.GetWriter( instancePath ) )  
{  
    instanceFile.Write( bytes );  
}
```

But because `BinaryWriter` doesn't support async semantics, I've refactored this to use asynchronous APIs appropriate to each target platform. For example, **Figure 1** shows how `SaveAsync` looks for the Sterling Windows Azure Table Storage driver.

I still use `BinaryWriter` to write discrete values to an in-memory stream, but then use the Windows Azure `DynamicTableEntity` and `CloudTable.BeginExecute` and `.EndExecute` to asynchronously store the stream's byte array content in the Windows Azure Table Storage service. There were several other, similar changes necessary to achieve the async data-update behavior of Sterling. The key point: surface-level API refactoring may be only the first of several steps necessary to achieve a migration redesign goal like this. Plan your work tasks and effort estimates accordingly, and be realistic about whether such a change is even a reasonable goal in the first place.

In fact, my experience with Sterling surfaced just such an unrealistic goal. A core design characteristic of Sterling is that all storage operations work against strongly typed data, using standard .NET data contract serialization APIs and extensions. This works well for Windows Phone and .NET 4.5 clients, as well as C#-based Windows Store apps. However, there's no notion of strong typing in the world of Windows Store HTML5 and JavaScript clients. After some research and discussion with Likness, I determined there was no easy way to make Sterling available to these clients, so I chose to omit them as supported options. Such potential mismatches must of course be considered case-by-case, but know that they can arise and be realistic about your options.

Share Code Across Target Platforms The next big challenge I faced was one we've all encountered at one time or another: how to share common code across multiple projects?

Identifying and sharing common code across projects is a proven strategy for minimizing time-to-market and downstream maintenance headaches. We've done this for years in .NET; a typical pattern is to define a Common assembly and reference that from multiple consumer projects. Another favorite technique is the "Add As Link" functionality in Visual Studio, which grants the ability to share a single master source file among multiple projects, demonstrated in **Figure 2**.

Even today, these options work well if the consumer projects all target the same underlying platform. However, when you want to expose common functionality across multiple platforms (as in my case with Sterling), creating a single Common assembly for such code becomes a significant development burden. Creation and maintenance of multiple build targets becomes a necessity, which increases the complexity of the project configuration and build process. Use of preprocessor directives (`#if`, `#endif` and so on) to conditionally include platform-specific behavior for certain build configurations is a virtual necessity, which makes the code more difficult to read, navigate and reason about. Energy wasted on such configuration burdens distracts from the primary goal of solving real problems through code.

Happily, Microsoft anticipated the need for easier cross-platform development and, beginning in the .NET Framework 4, added a new feature called Portable Class Libraries (PCLs). PCLs allow you to selectively target multiple versions of the .NET Framework, Silverlight and Windows Phone, as well as Windows Store and

Xbox 360, all from a single Visual Studio .NET project. When you choose a PCL project template, Visual Studio automatically ensures that your code uses only libraries that exist on each chosen target platform. This eliminates the need for clumsy preprocessor directives and multiple build targets. On the other hand, it does place some restrictions on which APIs you can call from your library; I'll explain more on how to work around such restrictions in a moment. See "Cross-Platform Development with the .NET Framework" (msdn.microsoft.com/library/gg597391) for further details on PCL features and use.

PCLs were a natural fit for achieving my cross-platform goals with Sterling. I was able to refactor more than 90 percent of the Sterling codebase into a single common PCL usable without modification from the .NET Framework 4.5, Windows Phone 8 and Windows 8. This is a huge advantage for the long-term viability of the project.

A brief note about unit test projects: As of today, there's no PCL equivalent for unit test code. The primary obstacle to creating one is the lack of a single unified unit test framework that works across multiple platforms. Given that reality, for Sterling unit tests I defined separate test projects for .NET 4.5, Windows Phone 8 and Windows 8; the .NET 4.5 project contains the sole copy of the test code, while the other projects share the test code using the Add As Link technique mentioned earlier. Each platform project does reference the test framework assemblies unique to that platform; fortunately, the namespace and type names are identical in each, so the same code compiles unmodified in all the test projects. See the updated Sterling codebase on GitHub (bit.ly/YdUNRN) for examples of how this works.

Leverage Platform-Specific APIs While PCLs are a huge help in creating unified cross-platform codebases, they do pose a bit of a dilemma: How do you use platform-specific APIs that aren't callable from PCL code? A perfect example is the asynchronous code refactoring I mentioned; while .NET 4.5 and Windows Store apps in particular have a wealth of powerful async APIs to choose from, none of these is callable from within a PCL. Can you have your cake and eat it too?

Figure 1 How SaveAsync Looks for the Sterling Windows Azure Table Storage Driver

```
using ( var stream = new MemoryStream() )
{
    using ( var writer = new BinaryWriter( stream ) )
    {
        action( writer );
    }

    stream.Position = 0;

    var entity = new DynamicTableEntity( partitionKey, rowKey )
    {
        Properties = new Dictionary<string, EntityProperty>
        {
            { "blob", new EntityProperty( stream.GetBuffer() ) }
        }
    };

    var operation = TableOperation.InsertOrReplace( entity );

    await Task<TableResult>.Factory.FromAsync(
        table.BeginExecute, table.EndExecute, operation, null );
}
```



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!



www.alachisoft.com

1-800-253-8195



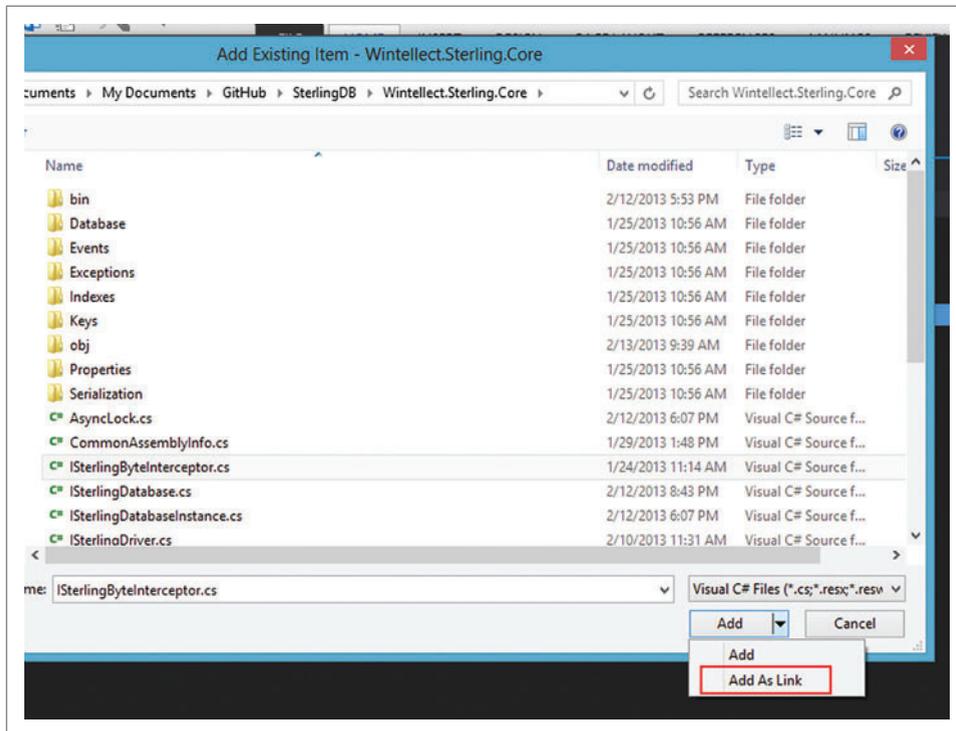


Figure 2 The Visual Studio 2012 Add As Link Feature

It turns out you can—with a bit of work. The idea is to define inside your PCL one or more interfaces that model the platform-specific behaviors you're unable to call directly, and then implement your PCL-based code in terms of those interface abstractions. Then, in separate platform-specific libraries, you provide implementations for each interface. Finally, at run time you create instances of PCL types to accomplish some task, plugging in the specific interface implementation appropriate for the current target platform. The abstraction allows the PCL code to remain decoupled from platform details.

If this all sounds vaguely familiar, it should: the pattern I've described here is known as Inversion of Control (IoC), a tried-and-true software design technique for achieving modular decoupling and isolation. You can read more about IoC at bit.ly/13VBTpQ.

While porting Sterling, I resolved several API incompatibility issues using this approach. Most of the problem APIs came from the System.Reflection namespace. The irony is that while each target platform exposed all of the reflection functionality I needed for Sterling, each had its own minor quirks and nuances that made it impossible to support them uniformly in PCLs. Hence the need for this IoC-based technique. You'll find the resulting C# interface abstraction I defined for Sterling to work around these problems at bit.ly/13FtG0.

A Bit of General Advice

Now that I've outlined my migration strategy for Sterling, I'll take a small step back and consider how the lessons from the experience might apply in the general case.

First—and I can't stress this enough—use the PCL feature. PCLs are a huge win for cross-platform development, and they offer enough configuration flexibility to suit most any need. If you're migrating an existing library forward (or even writing a new one)

and it targets more than one platform, you should be using a PCL.

Next, anticipate some refactoring effort in order to accommodate updated design goals. In other words, don't expect your code migration to be a simple mechanical process, replacing one API call with another. It's entirely possible that the changes you'll need to make go deeper than surface level and might require changing one or more core assumptions made when the original codebase was written. There's a practical limit to the total churn you can impose upon the existing code without major downstream impact; you'll have to decide for yourself where that line is, and when and if to cross it. One person's migration is another's source code fork into an entirely new project.

Finally, don't abandon your existing toolbox of patterns and design techniques. I've demon-

strated how I used the IoC principle and dependency injection with Sterling to take advantage of platform-specific APIs. Other, similar approaches will undoubtedly serve you well. Classic software design patterns such as strategy (bit.ly/Hhms), adapter (bit.ly/xRM3i), template method (bit.ly/OrfyT) and facade (bit.ly/gYAK9) can be very useful when refactoring existing code for new purposes.

Brave New Worlds

The end result of my work is a fully functional Sterling NoSQL implementation on the three target platforms of the .NET Framework 4.5, Windows 8 and Windows Phone 8. It's gratifying to see Sterling run on the latest Windows-based devices like my Surface tablet and my Nokia Lumia 920 phone.

The Sterling project is hosted on the Wintellect GitHub site (bit.ly/X5jmUh) and contains the full migrated source code as well as unit tests and sample projects for each platform. It also includes an implementation of the Sterling driver model that uses Windows Azure Table Storage. I invite you to clone the GitHub repository and explore the patterns and design choices I've outlined in this article; I hope they serve as a useful starting point for your own similar efforts.

And remember, don't throw out that old code ... migrate it! ■

JOSH LANE is a senior consultant for Wintellect LLC in Atlanta. He's spent 15 years architecting, designing and building software on Microsoft platforms, and has successfully delivered a wide range of technology solutions, from call center Web sites to custom JavaScript compilers and many others in between. He enjoys the challenge of deriving meaningful business value through software. Reach him at jlane@wintellect.com.

THANKS to the following technical expert for reviewing this article:
Jeremy Likness (Wintellect)

INSTALLAWARE ON ...MARS!

We're not on this red, dusty planet yet... but we might as well have been!

- > *Industry leading designs copied by InstallShield (Partial Web Deploy, Shell to MSI)*
- > *Advanced artificial intelligence with MSIcode scripting (un-copy-able by InstallShield)*
- > *Successful delivery to the most hostile of environments with Native Code Setup Engine*
- > *Bullet-proof, dependency free; run the same setup on Windows 95 - Server 2012 x64!*
- > *Scalable and secure delivery of tens of gigabytes from redundant source URLs.*

VERSION
15

 InstallAware



Windows Phone Video Capture: A Best-of-Breed Approach

Chris Barker

The Windows Phone 7.1 SDK opened up a number of new development scenarios, including the ability to access the camera of Windows Phone 7.5 devices. When Windows Phone 8 was released, it brought new capabilities such as being able to capture high-definition 1080p videos on supported hardware. While the Windows Phone 7.1 SDK could've been extended to support the new device features, the Windows Phone 8 release also coincided with a more architectural change to the OS—it shares a kernel with Windows 8. A further significant change was made to move a number of APIs to the Windows 8 model and use a developer surface known as the Windows Runtime (WinRT).

This article discusses the prerelease “Async for .NET Framework 4, Silverlight 4 and 5, and Windows Phone 7.5” NuGet package. All related information is subject to change.

This article discusses:

- Strategy for the sample app
- Decoupling and abstraction
- Targeting Windows Phone 8 APIs
- The Model-View-ViewModel option

Technologies discussed:

Windows Phone

Code download available at:

archive.msdn.microsoft.com/mag201305Phone

The Windows Phone 7.1 SDK is the version that lets developers target features of Windows Phone 7.5. From here on, I'll refer to Windows Phone 7.5 to encompass both the OS version and the SDK. (Incidentally, the code name for Windows Phone 7.5 was “Mango,” and the code name for Windows Phone 8 was “Apollo”—you might see these names documented elsewhere.)

The Windows Runtime not only introduced a number of consistent APIs across Windows Phone and Windows 8, but it also provided a much more efficient, native runtime that allowed new Windows Phone-specific APIs to be provided using the same model. I won't go deeply into the new changes brought about by the Windows Runtime, but be aware it did impact APIs such as those enabling video capture.

I'm going to give you an overview of what has changed between versions, but the real takeaway is that you're going to learn how to maintain your Windows Phone 7.5 project while at the same time giving a richer experience to your Windows Phone 8 users. A huge benefit here is that the techniques discussed won't just apply to video capture, but to any of those APIs that have been reinvented for Windows Phone 8. I'm going to use an existing public code sample to help explain the techniques you can use to introduce code reuse across your Windows Phone 7.5 and 8 projects—and even Windows 8 projects, too.

Getting Started

One of the main goals I want to illustrate is how to port a solution to Windows Phone 8 while not neglecting Windows Phone 7.5

devices. Before I get into that, it's worth taking a step back and noting that in many cases you won't need to do anything at all. Your existing Windows Phone 7.5 app will continue to run on Windows Phone 8, although you should test it to make sure there's no unexpected behavior. Visual Studio 2012 will only allow you to use the supported APIs when targeting Windows Phone 7.5. And remember, you only need to worry about writing code for Windows Phone 8 if you want to take advantage of any of the new APIs and functionality it offers.

I'm going to assume you want to continue supporting your Windows Phone 7.5 users, while giving your Windows Phone 8 users access to the rich new features of their devices.

To help explain how you can get a best-of-breed solution, I'm going to use the Windows Phone (7.5) Video Recorder Sample (bit.ly/16tM2c1) as a starting point. Remember, although I happen to be using the video recorder code sample here, the approaches I discuss could apply to any number of platform features that you want to span across versions of the platform.

There are a couple of points to call out about this solution:

- It doesn't use the Model-View-ViewModel (MVVM) pattern.
 - The majority of the UI is in the MainPage.xaml file.
 - The majority of the logic is driven out of the MainPage.xaml.cs codebehind file.
- It uses the Windows Phone 7.5 API to manipulate the camera—namely the System.Windows.Media.VideoCaptureDevice class. Windows Phone 8 introduces a new API in this area that I'll talk about more, later on.

The fact that this solution isn't using the MVVM pattern isn't a big deal—it would be nice to have (and generally recommended for production code) because it saves you some refactoring work later on, but in itself, it's not a compatibility issue.

The VideoCaptureDevice class, however, is going to restrict you when moving to Windows Phone 8. It's going to run just fine against the Windows Phone 8 runtime, but it's not going to give you maximum performance and it's not going to allow you to target the full range of resolutions supported by the hardware (among other things).

In subsequent sections I'm going to take the following approach:

- Create a common abstraction for functionality that will be implemented differently between platforms. This will enable you to share the code that consumes that functionality.
- Put the abstraction for a video recorder in a Portable Class Library (PCL). This will make it easier to move more code into the PCL later on if needed.
- Share the codebehind for MainPage via file linking (a pragmatic decision because the existing app doesn't use

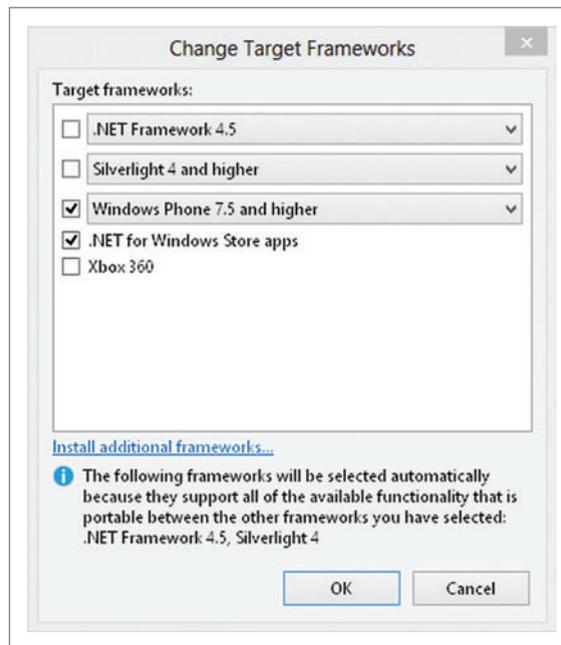


Figure 1 Configuring Your Portable Class Library

the MVVM pattern—otherwise I'd recommend moving the view models to a PCL).

So, given I want to be greedy and have a best-of-breed approach, I need to think about what I can do to abstract out the Windows Phone 7.5 camera logic from the MainPage.xaml.cs file. I'll look to resolve this problem in the next section.

Decoupling and Abstracting Your Implementation

First, you need a place into which to factor your code. A class library would make sense, but you can use a PCL. A PCL lets you specify the platforms you wish to target and provides a nice way of restricting you to the APIs provided by the intersection of those platforms—ultimately giving you the ability to make binary refer-

ences across projects (rather than simply code/project linking and rebuilding to the target platform at compile time). In this case you can create a PCL project that targets Windows Phone 7.5 or greater and Windows Store apps (that is, Windows 8 “modern apps”) and call it VideoRecorder.Shared (see **Figure 1**).

Following the abstraction pattern (bit.ly/YQwsVD), you can create a VideoRecorderCore abstract class, which will let you target platform-specific code where required. The abstract class will look something like **Figure 2**.

Note: In the example shown in **Figure 2** you could just as easily use an interface, but chances are you're going to need some common base functionality in many scenarios.

In the sdkVideoRecorderCS project you'd ideally spend some time implementing the MVVM pattern, but that can be left for another day. The refactoring for now will focus on providing a concrete implementation of the abstract class. Fortunately, you already have the concrete implementation—it's just a little too tightly coupled in the MainPage.xaml.cs right now. To address this, you can create a WP7VideoRecorder class in the sdkVideoRecorderCS

Figure 2 Creating a VideoRecorderCore Abstract Class

```
namespace VideoRecorder.Shared
{
    public abstract class VideoRecorderCore
    {
        public abstract void InitializeVideoRecorder();
        public abstract void StartVideoRecording();
        public abstract void StopVideoRecording();

        public abstract void StartVideoPreview();

        public abstract void DisposeVideoPlayer();
        public abstract void DisposeVideoRecorder();

        public static VideoRecorderCore Instance { get; set; }
    }
}
```

Figure 3 The InitializeVideoRecorder Method

```
public override void InitializeVideoRecorder()
{
    if (captureSource == null)
    {
        captureSource = new CaptureSource();
        fileSink = new FileSink();

        videoCaptureDevice =
            CaptureDeviceConfiguration.GetDefaultVideoCaptureDevice();

        captureSource.CaptureFailed +=
            new EventHandler<ExceptionRoutedEventArgs>(OnCaptureFailed);

        if (videoCaptureDevice != null)
        {
            videoRecorderBrush = new VideoBrush();
            videoRecorderBrush.SetSource(captureSource);

            viewfinderRectangle.Fill = videoRecorderBrush;

            captureSource.Start();

            UpdateUI(ButtonState.Initialized, "Tap record to start recording...");
        }
        else
        {
            UpdateUI(ButtonState.CameraNotSupported,
                "A camera is not supported on this device.");
        }
    }
}
```

project and inherit it from VideoRecorderCore in the PCL project. All you need to do next is move the implementation out of MainPage.xaml.cs and into the appropriate overridden method. As an example, the InitializeVideoRecorder method would initially look something like **Figure 3**.

I won't discuss every line of code in **Figure 3** here—that's done comprehensively in the documentation (bit.ly/WVf0I)—however, in summary, the code is initializing the VideoCaptureDevice instance and then setting up the video preview in the UI. There are a couple of issues I've introduced here by simply doing a copy and paste from the codebehind into the concrete implementation. The code is making references to UI elements and methods (for example, viewfinderRectangle and the UpdateUI method). You don't want

Figure 4 Codebehind That Works in Windows Phone 7.5 and Windows Phone 8

```
public void InitializeVideoRecorder()
{
    _videoRecorder.InitializeVideoRecorder();
    _videoRecorder.CaptureFailed += OnCaptureFailed;

    if (_videoRecorder.VideoCaptureDevice != null)
    {
        videoRecorderBrush = new VideoBrush();
        videoRecorderBrush.SetSource(
            _videoRecorder.VideoSource as CaptureSource);

        viewfinderRectangle.Fill = videoRecorderBrush;

        _videoRecorder.StartVideoSource();

        UpdateUI(ButtonState.Initialized, "Tap record to start recording...");
    }
    else
    {
        UpdateUI(ButtonState.CameraNotSupported,
            "A camera is not supported on this device.");
    }
}
```

these in your concrete implementation, and if you had already introduced a view model, these would be more easily factored out. So, there's a little bit of cleanup work required here:

1. Move the UI code back into the respective UI method (InitializeVideoRecorder in the MainPage.xaml.cs file, in this case).
2. Create a new abstract method for initializing the VideoRecorderBrush, as this will be required across Windows Phone 7.5 and Windows Phone 8—but the implementations might differ.

Once you've done some of this housekeeping, your method will look something like this:

```
public override void InitializeVideoRecorder()
{
    if (_captureSource == null)
    {
        _captureSource = new CaptureSource();
        _fileSink = new FileSink();

        _videoCaptureDevice =
            CaptureDeviceConfiguration.GetDefaultVideoCaptureDevice();

        _captureSource.CaptureFailed +=
            new EventHandler<ExceptionRoutedEventArgs>(OnCaptureFailed);
    }
}
```

You're now able to get the benefit of a Windows Phone code-behind (MainPage.xaml.cs) that works across versions Windows Phone 7.5 and Windows Phone 8, as shown in **Figure 4**.

All you need to do is initialize the VideoRecorder instance and you're ready to begin consuming a new platform-specific implementation:

```
_videoRecorder = VideoRecorderCore.Instance = new WP7VideoRecorder();
```

Aside from a few minor changes, you can simply go ahead and refactor the remaining methods in a similar way.

Targeting the New Windows Phone 8 APIs

As summarized earlier, porting your Windows Phone 7.5 code to Windows Phone 8 is pretty trivial—all you need to do is open the project and select "Upgrade to Windows Phone 8.0" (see **Figure 5**)—but what if you want to maintain both versions? Following the steps that you've just taken, the code is now structured in such a way as to easily support code reuse. Not only that, but because you've now decoupled the video recorder, you can make use of the new, more powerful WinRT API for video-camera capture on Windows Phone 8.

The next step is to create the Windows Phone 8 project, reusing as much code as possible and introducing any new functionality that you'd like.

You have two options for creating your Windows Phone 8 version at this point. You can either make a copy of your existing project and upgrade it, or you can create a new Windows Phone 8 project from scratch and add in the duplicate code files and references. The decision really comes down to how big your project is and how much code you're likely going to want to reuse. In this example you'll make a copy of the project and upgrade—this means you get to cherry-pick the pieces you want to reuse, and it reduces the chance of missing some key files. Here's what you'll need to do:

- Copy the sdkVideoRecorderCS project and rename the sdkVideoRecorderCS.cspj to sdkVideoRecorderCS.WP8.cspj. Next, add the new project to the existing solution to help maintain both versions.

- Upgrade the new project to Windows Phone 8. You'll find that the upgraded project runs just fine on Windows Phone 8 without any further modification—this is thanks to the backward compatibility of the API. The advantages of this are that no code changes are required, but the downside can be poorer performance—and you aren't going to be able to use the latest and greatest features of the platform.
- You can now pick and choose the elements you'd like to reuse. For example, MainPage.xaml and MainPage.xaml.cs are largely going to remain the same across versions (at least in this solution), so delete

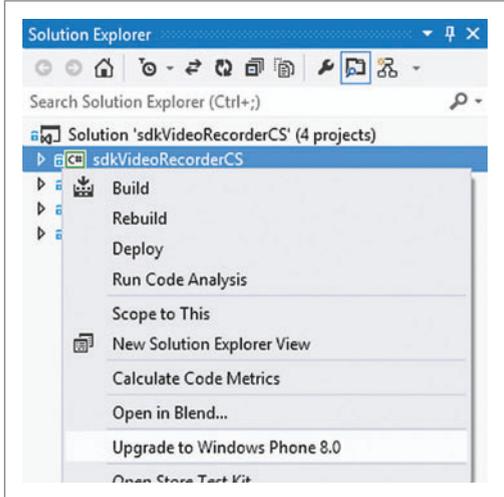


Figure 5 Upgrading Your Project to Target the Windows Phone 8 Runtime

the existing version of these files in the Windows Phone 8 project and add a file link to the versions in the Windows Phone 7.5 project by right-clicking on your project, selecting “Add Existing Item” and pointing at the MainPage.xaml file with “Add As Link” selected (see Figure 6).

Note: Instead of going through the menus as just described, you can simply drag the MainPage.xaml file from one project into the target project while holding Ctrl+Shift. This will create a file link and also move the codebehind file automatically.

You'll need to repeat the preceding steps for all of the relevant files, but for the purpose of demonstration, this is the only file link you'll need to add and make reuse of for now.

As a brief aside, there's often a utopian dream when it comes to code reuse, but in the real world there are going to be times when you want to use the same code across platforms and there might be just one or two subtle differences. In these circumstances it can be impractical to maintain two sets of code, and it's overkill to abstract the platform specifics out. In this scenario it's often better to use preprocessor directives to target specific lines of code at a specific platform. An example will be shown of this a little later, but it's important not to get too carried away with this technique or your code can quickly become spaghetti.

A good example of when to use platform abstraction is when you have an isolated piece of functionality that's platform-specific. In the current example, this will be the actual video recorder logic. Currently in your Windows Phone 8 project you have a WP7VideoRecorder.cs file that works perfectly fine, but it isn't using the new Windows Phone 8 API, so that's what you're going to change.

Delete the WP7VideoRecorder.cs file from your Windows Phone 8 project and create a new one called WP8VideoRecorder.cs that also inherits from VideoRecorderCore.

As before, implement each of the methods. The key difference

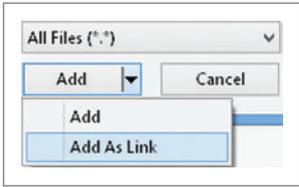


Figure 6 Adding a File Link

this time around is that you're going to use the Windows.Phone.Media.Capture namespace rather than the System.Windows.Media namespace. The former is the newer WinRT namespace, which introduces a class called AudioVideoCaptureDevice. This serves a similar purpose to the VideoCaptureDevice class used previously, but it's much richer in functionality.

There are a couple of approaches that WinRT APIs take differently from their predecessors. One such change you'll come across here is that many of the APIs are asynchronous (more on that later). Another difference is that you deal with storage using the Windows.Storage namespace rather than using the familiar System.IO.IsolatedFileStream (although

the latter will still be used to support media playback).

I'll walk through a few of the bigger changes now to show some differences in the video recorder scenario and how you more generally approach resolving these types of differences while maintaining an element of code reuse.

In the Windows Phone 7.5 version of the video recorder code, there were several private variables defined:

```
private CaptureSource _captureSource;
private VideoCaptureDevice _videoCaptureDevice;

private IsolatedStorageFileStream _isoVideoFile;
private FileSink _fileSink;
private string _isoVideoFileName = "CameraMovie.mp4";
```

Ideally you want to keep the two codebases as similar as possible, but there's no need for a CaptureSource or FileSink in the new API—the CaptureSource is replaced by the actual VideoCaptureDevice instance, which can act as a source, and you get the FileSink functionality for free so you just need a StorageFile for it to save to:

```
private AudioVideoCaptureDevice _videoCaptureDevice;

private IsolatedStorageFileStream _isoVideoFile;
private StorageFile sfVideoFile;
private string _isoVideoFileName = "CameraMovie.mp4";
```

The logical steps that follow are to work through the concrete implementation of the methods. I'm going to start with InitializeVideoRecorder again. You've seen how this method previously looked, and you need to simply initialize the AudioVideoCaptureDevice instance in a similar way here:

```
public async override void InitializeVideoRecorder()
{
    CameraSensorLocation location = CameraSensorLocation.Back;
    var captureResolutions =
        AudioVideoCaptureDevice.GetAvailableCaptureResolutions(location);

    _videoCaptureDevice =
        await AudioVideoCaptureDevice.OpenAsync(location, captureResolutions[0]);

    _videoCaptureDevice.RecordingFailed += OnCaptureFailed;
}
```

As you can see, the syntax is different, but the code is serving the same purpose. This part of the code also allows you to configure additional camera properties such as specifying the capture resolution and which camera (if there are multiple ones available) you'd

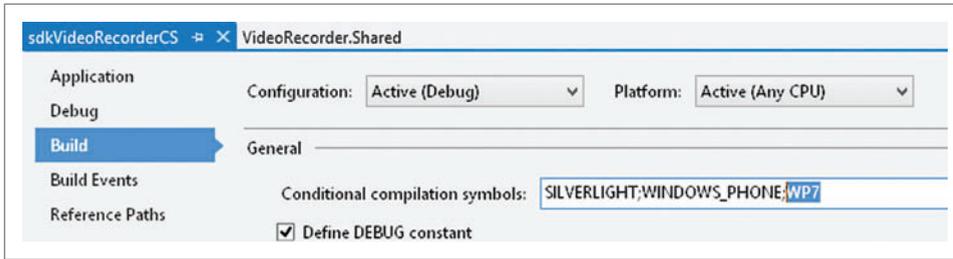


Figure 7 Adding a Custom Conditional Compilation Symbol to Your Windows Phone 7.5 Project

like to use. The control you get over the resolutions here is already an advantage over what you get with the Windows Phone 7.5 APIs, as this will allow you to use full-scale, high-definition 1080p if your phone camera supports it. As with many of the WinRT APIs, the other welcome difference is that the methods are asynchronous. To aid manageability you can introduce the C# 5 keywords `async` and `await`. I won't go into detail on these features here as they've been covered elsewhere, but they help you gain the benefits of asynchronous code while reducing the complexity of doing so.

The next method to take a look at is the `InitializeFileSink` method. Although you no longer have a `FileSink` per se, you do need to initialize the files in which you're storing the video capture. This also happens to be where you begin using the `Windows.Storage` namespace with classes such as `StorageFolder` and `StorageFile`:

```
public async override void InitializeFileSink()
{
    StorageFolder isoStore = ApplicationData.Current.LocalFolder;
    sfVideoFile = await isoStore.CreateFileAsync(
        _isoVideoFileName, CreationCollisionOption.ReplaceExisting);
}
```

There's no need to go through every method as these are really the biggest differences to highlight. Where methods remain the same across both versions, you can move them into the `VideoRecorderCore` base class to ensure a common behavior across the subclasses.

Once the remaining changes have been made, you should notice a build error relating to the following line:

```
_videoRecorder = VideoRecorderCore.Instance = new WP7VideoRecorder();
```

The cause here is pretty obvious—it's due to the Windows Phone 8 code referencing a class that no longer exists: `WP7VideoRecorder`. Of course, you need to reference the new `WP8VideoRecorder` class, but making that change in the `MainPage.xaml.cs` file would break the Windows Phone 7.5 code because you're linking to the same file.

You have a few options here. You could use preprocessor directives like so:

```
#if WP7
_videoRecorder = VideoRecorderCore.Instance = new WP7VideoRecorder();
#else
_videoRecorder = VideoRecorderCore.Instance = new WP8VideoRecorder();
#endif
```

If you were to use preprocessor directives, a tidier approach would be to move these inside the wrapper class—for example, introduce a `CreateVideoRecorder` method into the `VideoRecorderCore` abstract base class and place the preceding `#if` declaration inside of this.

The route I decided to take here was to avoid using the preprocessor directives and abstract the concrete class lookup into a separate factory class—this means the code consuming the recorder is consistent across both Windows Phone 7.5 and Windows Phone 8:

```
_videoRecorder =
    VideoRecorderCore.Instance = VideoRecorderFactory.CreateVideoRecorder();
```

An alternative approach would be to use a Dependency Injection (DI) or Service Locator pattern and leave the instance resolution as a responsibility of the container or service, respectively.

A more discreet issue involves setting the `VideoBrush` for the camera preview. In this case you can again consider the preprocessor `#if`

directive technique to differentiate the minor implementation details:

```
#if WP7
    videoRecorderBrush.SetSource(_videoRecorder.VideoSource as CaptureSource);
#else
    videoRecorderBrush.SetSource(_videoRecorder.VideoSource);
#endif
```

This is required because Windows Phone 8 overloads the `SetSource` method to allow an object to be specified as the `Source` object. This is why you're able to return the `AudioVideoCaptureDevice` instance in the `VideoSource` property for Windows Phone 8. To support this approach, you'll need to add a new conditional "WP7" compilation symbol to the Windows Phone 7.5 project under `Project Properties | Build` (see **Figure 7**).

The Windows Runtime relies heavily on asynchronous methods in order to improve the responsiveness of an app.

As mentioned previously, the Windows Runtime relies heavily on asynchronous methods in order to improve the responsiveness of an app, but if you're sharing code across versions of the platform, what impact does this have? Although you can write C# 5 code in a project targeting Windows Phone 7.5, there are certain implementation details that aren't available when you want to use the `async` and `await` keywords. Consider the following method:

```
public async override void InitializeVideoRecorder()
{
    CameraSensorLocation location = CameraSensorLocation.Back;
    var captureResolutions =
        AudioVideoCaptureDevice.GetAvailableCaptureResolutions(location);

    _videoCaptureDevice =
        await AudioVideoCaptureDevice.OpenAsync(location, captureResolutions[0]);

    _videoCaptureDevice.RecordingFailed += OnCaptureFailed;
}
```

This is effectively setting the `VideoCaptureDevice` property, but the consuming code looks like this:

```
_videoRecorder.InitializeVideoRecorder();

if (_videoRecorder.VideoCaptureDevice != null)
{
    ...
}
else
{
    UpdateUI(ButtonState.CameraNotSupported,
        "A camera is not supported on this device.");
}
```

What might not be immediately obvious is that while you're awaiting the call to the `AudioVideoCaptureDevice.OpenAsync` method, the check on `_videoRecorder.VideoCaptureDevice` might already have happened. In other words, the `VideoCaptureDevice` check might evaluate to null because it hasn't had a chance to initialize yet.

Again, you have a few choices at this stage. By making the consuming code asynchronous, it does mean retrofitting your code a little bit—and depending on your solution this could be a lot of work. If you want to avoid this, you could merely wrap up the asynchronous code inside a synchronous wrapper method. Another alternative might be to have an event-based mechanism for notifying the consumer when the initialization is complete.

Perhaps you want to use this opportunity to move to the new async model and therefore await the `InitializeVideoRecorder` call, but how can you do this if the Windows Phone 7.5 SDK doesn't support the required constructs? One solution is to download the "Async for .NET Framework 4, Silverlight 4 and 5, and Windows Phone 7.5" NuGet package within Visual Studio, as shown in **Figure 8**.

At the time of writing this package was still in prerelease, but a stable release is expected soon. If you want to go a more tried-and-tested route, you'll need to rework the code as described earlier and introduce an event pattern or synchronous wrapper.

Note: Make sure you have at least version 2.1 of the NuGet Package Manager installed or you'll get unexpected error messages trying to use this package. You can download the latest version from bit.ly/dUeqlu.

Once you've downloaded this package, you can align the code in each project. For example, you can bake async support into your abstract base class by returning a `Task` type rather than a void—this will allow you to await the result rather than use the current "fire-and-forget" mechanism. It's also a good convention to append "Async" to the names of the asynchronous method names, which makes consuming the methods more intuitive to a developer. For example, some of the methods will now have a signature like this:

```
public abstract Task InitializeVideoRecorderAsync();
public abstract Task InitializeFileSinkAsync();
```

You will, of course, then need to refactor the code in the subclasses. For example:

```
public override async Task InitializeVideoRecorderAsync()
{
```

And finally, you'll be able to update your `MainPage.xaml.cs` code to support the ability to await on asynchronous operations. An example of this is:

```
public async void InitializeVideoRecorder()
{
    await _videoRecorder.InitializeVideoRecorderAsync();
```

Having made these final changes, you should now find that you can run both the Windows Phone 7.5 and Windows Phone 8 projects successfully. You have a strong element of code reuse and you're using the latest features where they're supported. You have a best-of-breed solution!

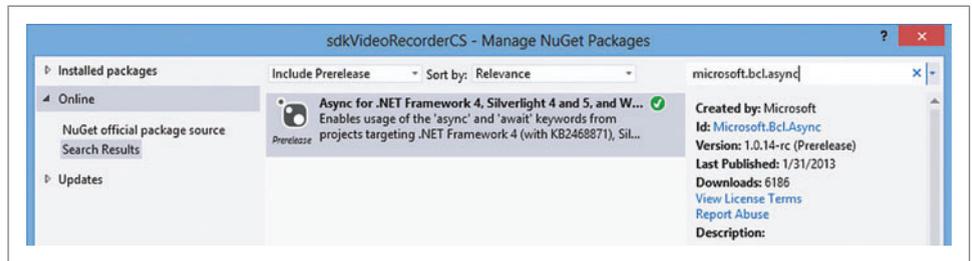


Figure 8 Find the Async NuGet Package for Windows Phone 7.5 by Searching for "microsoft.bcl.async"

What About Windows 8?

Many of the partners who I work with have developed a Windows Phone app using XAML and C#. When considering developing a Windows Store app for Windows 8, one of their first technical questions is whether they should go the HTML5/JavaScript route or the XAML/C# route. A well-architected Windows Phone solution can offer a great amount of code reuse with Windows 8, so this can be the decisive factor when choosing which language you'll use to develop your Windows Store app.

Although you'll be able to reuse code between Windows Phone 7.5 and Windows 8, there are still a number of scenarios where you need to branch between the two. The video recorder is one such scenario, but the good news here is that the approaches you've taken earlier save you a lot of time and effort.

Implementing a Windows 8 version of this code is left as an exercise to the reader, or if you so choose you can download the accompanying sample that includes all of the code discussed along with a Windows 8 solution.

Note: For a Windows 8 implementation you'll need to utilize the `Windows.Media.Capture.MediaCapture` class.

Going Further with MVVM

What you've just implemented gives you 100 percent reuse across `MainPage.xaml` for the two versions of Windows Phone. Given the abstraction and refactoring steps you went through, you were also able to achieve near-100 percent reuse across the accompanying codebehind file as well. You could take these techniques even further by introducing an MVVM pattern, and this would help to provide a great deal of reuse not just over Windows Phone versions, but across your Windows 8 codebase, too.

These techniques can be used to gain reuse while abstracting out platform-specific features such as the video capture I described, but also application lifecycle management (ALM), file storage and lots more. ■

CHRIS BARKER is a technical evangelist for Microsoft based in Derbyshire, United Kingdom. He has spent many years developing code for both independent software vendors and enterprise organizations, running performance and scalability labs, and taking a keen interest in low-level debugging techniques. In recent years he has focused more on client development using XAML and C# and now works with global consumer partners developing for the Windows 8 and Windows Phone platforms.

THANKS to the following technical expert for reviewing this article:
Daniel Plaisted (Microsoft)

Understanding and Using the SharePoint 2013 REST Interface

Jim Crowley and Ricky Kirkham

SharePoint 2013 provides a Representational State Transfer (REST) interface that opens the SharePoint 2013 development platform to standard Web technologies and languages. SharePoint capabilities have long been available to Web applications through the client object model, but those APIs are available only to .NET applications and languages. The extensive REST interface in SharePoint 2013 makes it possible to access most SharePoint capabilities and entities with standard Web languages such as JavaScript or PHP, and any technology stack or language that supports REST.

Because the REST interface doesn't require referencing client assemblies, it also allows you to limit the footprint of your Web applications—an especially important consideration for mobile applications. REST has obvious benefits for mobile applications

that are written for non-Microsoft platforms, such as iOS and Android devices, but it's also a useful resource for Windows 8 app developers. A Windows 8 app written in HTML5 and JavaScript would require the REST interface for any SharePoint operations, and C# developers who are concerned about the size of their apps would also want to consider using REST. This article will demonstrate how to use this interface to incorporate the power of the SharePoint platform into your applications and perform advanced operations with SharePoint entities.

The extensive REST interface in SharePoint 2013 makes it possible to access most SharePoint capabilities and entities with standard Web languages.

The Basics

Before it can do anything with SharePoint, your remote Web or mobile application must obtain authorized access. In SharePoint 2013 there are two general approaches to authorizing access to a

This article discusses:

- Obtaining authorized access to a SharePoint site
- Constructing RESTful URLs
- Writing to SharePoint
- Sorting, filtering and ordering data
- Working with files and folders
- Using change queries
- Debugging your application
- A sample PHP application

Technologies discussed:

SharePoint 2013, HTML5, JavaScript, PHP, Windows 8

Figure 1 Endpoints for Read Operations

URL	Returns
<code>_api/web/title</code>	The title of the current site
<code>_api/web/lists(guid '<list id>')</code>	A list
<code>_api/web/lists/getByTitle('Announcements')/fields</code>	The columns in the Announcements list
<code>_api/web/lists/getByTitle('Task')/items</code>	The items in the Task list
<code>_api/web/siteusers</code>	The users in the site
<code>_api/web/sitegroups</code>	The user groups in the site
<code>_api/web/sitegroups(3)/users</code>	The users in group 3
<code>_api/web/GetFolderByServerRelativeUrl('/Shared Documents')</code>	The root folder of the Shared Documents library
<code>_api/web/GetFolderByServerRelativeUrl('/Plans')/Files('a.txt')/\$value</code>	The file a.txt from the Plans library

SharePoint site (regardless of whether you're using REST). The first approach involves authenticating a SharePoint user, in which case your application has the same access to SharePoint data and capabilities as the user. The different ways you can authenticate a user from a Web or mobile application are outside the scope of this article, but we'll briefly mention two new options in SharePoint 2013 because they affect the way you construct your REST requests:

- If you're calling into SharePoint from a remotely hosted application that can't use client-side code (HTML and JavaScript) exclusively, and there's no firewall between SharePoint and your application, you can use OAuth 2.0 tokens with Microsoft Access Control Service (ACS) as the secure token server.
- If client-side code and the permissions of a user who's logged in to SharePoint are sufficient, then the JavaScript cross-domain library (bit.ly/12kFwSP) is a good alternative to OAuth. The cross-domain library is also a convenient alternative to OAuth whenever you're making remote calls through a firewall. The MSDN Library article, "Data access options for SharePoint 2013" (bit.ly/WGvt9L), describes these options in detail.

Using OAuth The MSDN Library article, "Authorization and authentication for apps in SharePoint 2013" (bit.ly/XAyv28), covers the details of how OAuth works in SharePoint 2013 and how you can get an access token for your application. Once you have a token, you need to pass it with each REST request. You do this by adding an Authorization header that passes the access token as its value, preceded by "Bearer":

```
Authorization: Bearer access_token
```

For examples of C# and JavaScript code that do this, see the "Reading data with the SharePoint 2013 REST interface" section of the MSDN Library article, "How to: Complete basic operations using SharePoint 2013 REST endpoints" (bit.ly/13fjqFn). This example, which retrieves all of the lists from a SharePoint site, shows you what a basic REST request that passes an OAuth token looks like:

```
HttpWebRequest endpointRequest =
    (HttpWebRequest)HttpWebRequest.Create(
        "http:// <http://> <site url>/_api/web/lists");
endpointRequest.Method = "GET";
endpointRequest.Accept = "application/json;odata=verbose";
endpointRequest.Headers.Add("Authorization", "Bearer " + accessToken);
HttpWebResponse endpointResponse =
    (HttpWebResponse)endpointRequest.GetResponse();
```

Using the Cross-Domain JavaScript Library

The SharePoint cross-domain library is contained in the file `SP.RequestExecutor.js`, which is located in the virtual `/_layouts/15/` directory on SharePoint servers. To use it, load this file on a remote Web page. In JavaScript, create an `SP.RequestExecutor` object and then call its `executeAsync` method. As a parameter to this method you pass the information it needs to construct an HTTP request to the REST service. The main differences between REST calls using OAuth and REST calls using the cross-domain library are that for the latter, you don't need to pass an access token in the request, but you do need to specify (in the RESTful URL) the SharePoint Web site that will serve as the client

context site. The MSDN Library article, "How to: Access SharePoint 2013 data from remote apps using the cross-domain library" (bit.ly/12kFwSP), discusses these details (such as the difference between an app Web and a host Web) more thoroughly. This example, which retrieves all of the lists from a SharePoint site, shows you what a REST request that uses the cross-domain library looks like:

```
var executor = new SP.RequestExecutor(appweburl);
executor.executeAsync(
    {
        url:
            appweburl +
            "/_api/SP.AppContextSite(@target)/web/lists?@target='" +
            hostweburl + "'",
        method: "GET",
        headers: { "Accept": "application/json; odata=verbose" },
        success: successHandler,
        error: errorHandler
    }
);
```

Constructing RESTful URLs The SharePoint REST service is implemented in a `client.svc` file in the virtual folder `/_vti_bin` on the SharePoint Web site, but SharePoint 2013 supports the abbreviation `"_api"` as a substitute for `"_vti_bin/client.svc"`. This is the base URL for every endpoint:

```
http://<domain>/<site url>/_api/
```

Before it can do anything with SharePoint, your remote Web or mobile application must obtain authorized access.

The service-relative URLs of specific endpoints are appended to this base; for example, you can retrieve the lists from a SharePoint site with this URL:

```
http://<domain>/<site url>/_api/web/lists
```

You can get a reference to a particular list by specifying its ID or, as in the following example, by its title:

```
_api/web/lists/getByTitle('samplelist')/
```

The "web" in these examples is not a placeholder—it's the name of an object of the Web class in the SharePoint client object model;

Figure 2 Options for Filtering and Sorting Data

Option	Purpose
\$select	Specifies which fields are included in the returned data.
\$filter	Specifies which members of a collection, such as the items in a list, are returned.
\$expand	Specifies which projected fields from a joined list are returned.
\$top	Returns only the first <i>n</i> items of a collection or list.
\$skip	Skips the first <i>n</i> items of a collection or list and returns the rest.
\$orderby	Specifies the field that's used to sort the data before it's returned.

"lists" is the name of a collection property and "getByTitle" is a method of that collection object. This paradigm enables Microsoft to combine the API reference for the endpoints and the JavaScript object model; for examples, see SPWeb.lists and SPListCollection.getByTitle at bit.ly/14a38wZ and bit.ly/WNtRMO, respectively. Also, the syntax roughly mirrors the structure of a SharePoint tenancy. You get information about a site collection at `_api/site`; information about a SharePoint Web site at `_api/web`; and information about all of the lists in a Web site at `_api/web/lists`. The last URL delivers a list collection that consists of all the lists on the SharePoint site. You can also take a look at how these objects are represented in XML by navigating to those URLs on your development site collection.

Every major class of the SharePoint content object model is available, including site collection, Web sites, lists, folders, fields and list items. You can get information about users through the SP.User (bit.ly/15M4fzo), SP.UserCollection (bit.ly/16TQtnW), SP.Group (bit.ly/X55Pga) and SP.GroupCollection (bit.ly/ZnFhbG) classes. The table in **Figure 1** shows examples of various endpoints for read operations.

By default, the data is returned as XML in AtomPub format as extended by the OData format, but you can retrieve the data in JSON format by adding the following accept header to the HTTP request:

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

Whether to use JSON or Atom (XML) depends on your skill set, the programming platform you're using and whether network latency will be an issue for your app. JSON uses far fewer characters, so it makes sense for smaller payloads over the network. On the other hand, most major platforms, including the Microsoft .NET Framework, have rich XML-parsing libraries.

We'll describe later how you can use OData query operators to select, filter and order the data.

Writing to SharePoint All of the previous requests use the HTTP verb GET. When you write to SharePoint, your requests use the POST verb—though in some cases you'll override this verb by adding an X-HTTP-Method header to the request and specifying a value of PUT, MERGE or DELETE. In general, POST is used when you're creating an object such as a site, list or list item. MERGE is used when you're updating certain properties of an object and you want the other properties to keep their current values. PUT is used when you want to replace an item; properties not specifically mentioned in the request are set to their default values. DELETE is used when you want to remove an item.

Every request that writes to SharePoint must include a form digest. Your code gets the digest as part of a set of information returned by the following endpoint:

```
_api/contextinfo
```

You must use the POST verb in this request (with an empty body) and include the Authorization header as described earlier. In some frameworks you'll have to specify that the length of the POST request is 0. In the structure that's returned there's a property named FormDigestValue that contains the form digest. In all subsequent POST requests, you add an X-Request-Digest header with the digest as its value.

Note that if you're working with a SharePoint-hosted app and a page that uses the default master page for SharePoint, the digest is already on the page in an element with the ID "__REQUESTDIGEST" (with two underscore characters). So, instead of calling the contextinfo endpoint, you can simply read the value with script such as this jQuery code:

```
var formDigestValue = $("#__REQUESTDIGEST").val();
```

Of course, you need to add to the body of the request the data you want to write, or an identification of the data you want to delete. You can use either AtomPub/OData format or JSON format. If you choose the latter, you must add a content-type header to the request like the following:

```
content-type: application/json;odata=verbose
```

For a full set of concrete examples of Create, Read, Update and Delete (CRUD) operations on SharePoint objects, see "How to: Complete basic operations using SharePoint 2013 REST endpoints" at bit.ly/13fjqFn.

A certain degree of complexity comes along with the power of the SharePoint 2013 REST interface.

Advanced Operations

A certain degree of complexity comes along with the power of the SharePoint 2013 REST interface. The interface supports operations for sorting, filtering and ordering the data that it returns. It also supports a large number of SharePoint-specific operations. These additional capabilities add features and benefits that you don't always see in a standard REST implementation. The next sections discuss some of the most important factors you'll encounter when working with REST and SharePoint.

Filtering, Selecting and Sorting You can use OData system query options to control what data is returned and how it's sorted. **Figure 2** shows the supported options.

To return the author, title and ISBN from a list called Books, for example, you'd use the following:

```
_api/web/lists/getByTitle('Books')/items?$select=Author,Title,ISBN
```

If the \$select option isn't used, all fields are returned except fields that would be resource-intensive for the server to return. If you need these fields, you need to use the \$select option and specify them by name. To get all fields, use \$select='*'. To get all the books by Mark Twain, use:

```
_api/web/lists/getByTitle('Books')/items?$filter=Author eq 'Mark Twain'
```

For a list of all the operators supported for the \$filter option, see the MSDN Library article, “Programming using the SharePoint 2013 REST service,” at bit.ly/Zlqf3e.

To sort the books by title in ascending order, use:

```
_api/web/lists/getByTitle('Books')/items?$orderby=Title asc
```

Use “desc” in place of “asc” to specify descending order. To sort by multiple fields, specify a comma-separated list of fields.

An important consideration when updating files is that you can only use the PUT HTTP method.

You can conjoin multiple options using the “&” operator. To get only the Title of the first two books by Mark Twain, use:

```
_api/web/lists/getByTitle('Books')/items?$select=Title&$filter=Author eq 'Mark Twain'&$top=2
```

The service will completely resolve each option before it applies the next one. So each option only applies to the data set that’s produced by the options to its left in the URL. Thus, the order in which you apply the options matters. For example, the following URL returns items 3-10:

```
_api/web/lists/getByTitle('Books')/items?$top=10&$skip=2
```

But reversing the two options returns items 3-12:

```
_api/web/lists/getByTitle('Books')/items?$skip=2&$top=10
```

You can get the bottom *n* items by using a descending \$orderby and a \$top option (in that order). The following URL gets the bottom two items:

```
_api/web/lists/getByTitle('Books')/items?$orderby=ID desc&$top=2
```

When a SharePoint list has a lookup field to another list, this effectively serves as a join of the two lists. You can use the \$expand option to return projected fields from the joined list. For example, if the Books list has a PublishedBy field that looks up to the Name field of a Publisher list, you can return those names with this URL:

```
_api/web/lists/getByTitle('Books')/items?$select=Title,PublishedBy/Name&$expand=PublishedBy
```

Notice that you reference the column in the foreign list by using the syntax lookup_column_display_name/foreign_column_name, not foreign_list_name/foreign_column_name. It’s also important to note that you can’t select a lookup field name without also expanding it.

Figure 3 Uploading a Binary File Using the Cross-Domain Library

```
function uploadFileBinary() {
    var executor = new SP.RequestExecutor(appweburl);
    var body = "";
    for (var i = 0; i < 1000; i++) {
        var ch = i % 256;
        body = body + String.fromCharCode(ch);
    }
    var info = {
        url: "_api/web/lists/getByTitle(
            'Shared Documents')/RootFolder/Files/Add(url='a.dat', overwrite=true)",
        method: "POST",
        binaryStringRequestBody: true,
        body: body,
        success: success,
        error: fail,
        state: "Update"
    };
    executor.executeAsync(info);
}
```

Working with Files and Folders The best way to reach a document library is by taking advantage of the GetFolderByServerRelativeUrl method that’s available at /_api/web. When you add a file to a document library, you send the contents of your file in the request body, and you pass the name of the file in the URL:

```
http://<site url>/_api/web/GetFolderByServerRelativeUrl(
    '/Shared Documents')/Files/add(url='a.txt', overwrite=true)
```

An important consideration when updating files is that you can only use the PUT HTTP method. You can’t, therefore, merge the contents of a file into one that’s already stored in a document library. Instead, you replace one version of a file with another. You also need to make sure to use the \$value operator in your URL so you get access to the contents of the file itself, instead of the metadata associated with the file:

```
http://<site url>/_api/web/GetFileByServerRelativeUrl(
    '/Shared Documents/a.txt')/$value
```

It’s a best practice in SharePoint to check out files before making any changes to them, so you should check a file out before updating it, and then check it back in when you’re done. The following operations require you to make POST requests to these URLs, with empty request bodies:

```
http://<site url>/_api/web/GetFileByServerRelativeUrl(
    '/Shared Documents/a.txt')/CheckOut()
http://<site url>/_api/web/GetFileByServerRelativeUrl(
    '/Shared Documents/a.txt')/CheckIn(comment='Comment', checkintype=0)
```

The CheckIn method takes two parameters. The comment parameter enables you to add a comment to the check-in, and the checkintype parameter specifies whether this is a minor (0) or major (1) check-in.

It’s a best practice in SharePoint to check out files before making any changes to them.

One final consideration is that if you’re working with code (such as JavaScript) that runs in your browser client and you want to upload a file larger than 1.5MB, REST is your only option. This sort of operation with large files (larger than 1.5MB) is available only for Internet Explorer 10 (and later) and other modern browsers of equivalent vintage. The sample in **Figure 3** shows how you might upload a binary file using the cross-domain library.

Change Queries So far we’ve been describing how REST works with SharePoint entities you can reach with URLs that mimic the structure of a SharePoint site. Some SharePoint types can’t be reached or represented this way, however. In the context of REST, three of the most important of these are the ChangeQuery, ChangeLogItemQuery and ChangeToken types.

ChangeQuery objects make it possible for you to query the SharePoint change log for any updates that have been made to a SharePoint site collection, site or list. The REST interface exposes a getchanges method at each of these three locations:

- /_api/site (for site collections)
- /_api/web (for sites)
- /_api/web/lists/list(guid '<list id>') OR /_api/web/lists/getByTitle('list title') (for lists)

Figure 4 Requesting the Form Digest

```
$opts = array (
  'Authorization: Bearer ' . $accToken
);
$ch = curl_init();
$url = $appweburl . '/_api/contextinfo';
curl_setopt($ch, CURLOPT_URL, $url);
curl_setopt($ch, CURLOPT_HTTPHEADER, $opts);
curl_setopt($ch, CURLOPT_HEADER, 0);
curl_setopt($ch, CURLOPT_POST, 1);
curl_setopt($ch, CURLOPT_POSTFIELDS, '');
curl_setopt($ch, CURLOPT_SSL_VERIFYHOST, 0);
curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, 0);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
$result = curl_exec($ch);
```

You pass a query to any of these locations by adding `/getchanges` to the relevant URL path, and then sending a `ChangeQuery` object through the POST body of your request. A simple change query that asks for all items that have been added to a list would look like this (in JSON):

```
{'query': { '__metadata': {'type': 'SP.ChangeQuery'}, 'Add':
  'true', 'Item': 'true'}}
```

The `getchanges` method expects a request body that contains a representation of the `ChangeQuery` object inside the query parameter. You send this request to the URL for a specific list:

```
/_api/web/lists/list(guid '<list id>')/getchanges
or
/_api/web/lists/getByTitle('<list title>')/getchanges
```

The response returns a result that contains a collection of changes that matches the request. If the list has only one item, the response body would look like this:

```
{'d': {'results': [{'__metadata': {'id': 'https://<site url>/_api/
  SP.ChangeItem7e7c6e9-2c41-47c3-aae9-2b4a63b7a087', 'uri': 'https://
  site url/_api/SP.ChangeItem', 'type': 'SP.ChangeItem'}, 'ChangeToken': {'__
  metadata': {'type': 'SP.ChangeToken'}, 'StringValue': '1;3;482e41
  8a-0900-414b-8902-02248c2e44e8;634955266749500000;5749111'}, 'ChangeT
  ype': 1, 'SiteId': 'ce11bfbb-cf9d-4b2b-a642-8673bd48cceb', 'Time': '2013-
  02-03T22:17:54Z', 'ItemId': 1, 'ListId': '482e418a-0900-414b-8902-
  02248c2e44e8', 'WebId': 'a975b994-fc67-4203-a519-b160175ca967'}]}}
```

This response tells you that a single list item (with an `ItemId` value of 1) was added at a time that's represented in the change log by a `ChangeToken`. You can use the string value of this object to make your queries more precise. You can, for example, specify values for

Figure 5 Executing the Request that Uploads the File

```
"/_api/web/GetFolderByServerRelativeUrl('Lists/SharedDoc')/Files/
  add(url=' ' . $_FILES['file']['name'] . ".", 'overwrite=true");
$opts = array (
  'X-RequestDigest: ' . $_REQUEST['digest'],
  'Authorization: Bearer ' . $accToken);
// Initialize cURL
$ch = curl_init();
curl_setopt($ch, CURLOPT_HTTPHEADER, $opts);
// Set URL on which you want to post the Form and/or data
curl_setopt($ch, CURLOPT_URL, $url);
// Data+Files to be posted
curl_setopt($ch, CURLOPT_POST, 1);
curl_setopt($ch, CURLOPT_POSTFIELDS, $post_data);
// Pass TRUE or 1 if you want to wait for and catch the
// response against the request made
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
// For Debug mode; shows up any error encountered during the operation
curl_setopt($ch, CURLOPT_VERBOSE, 1);
curl_setopt($ch, CURLOPT_HEADER, 0);
curl_setopt($ch, CURLOPT_SSL_VERIFYHOST, 0);
curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, 0);
// Execute the request
$response = curl_exec($ch);
```

the `ChangeTokenStart` and `ChangeTokenEnd` properties of your `ChangeQuery` object to make sure you get changes that occurred before or after a single point in time, or between two points in time.

You can also use the value of the `ChangeToken` object when you use the `getListItemsChangesSinceToken` method:

```
/_api/web/lists/list(guid '<list id>')/getListChangesSinceToken
```

This method exists only in the REST interface. If you want to know all of the changes to items in this list since the addition of the first item, you construct a `ChangeLogItemQuery` object that contains the change token:

```
{'query': { '__metadata': {'type': 'SP.ChangeLogItemQuery'},
  'ChangeToken': '1;3;482e418a-0900-414b-8902-02248c2e4
  4e8;634955266749500000;5749111'}}
```

SharePoint Server 2013 Feature Areas All of the operations we discuss in this article apply to both SharePoint Foundation 2013 and SharePoint Server 2013, because they involve core SharePoint capabilities. The SharePoint REST interface also exposes many of the capabilities of SharePoint Server 2013 feature areas. These feature areas lie outside the scope of this article, but you can refer to the following resources in the SDK for more information about using REST to work with them:

- “SharePoint 2013: Using the search REST service from an app for SharePoint” (bit.ly/Mt4szN)
- “Get started developing with social features in SharePoint 2013” (bit.ly/102qlGM)
- “BCS REST API reference for SharePoint 2013” (bit.ly/10FFMMu)

The most important piece of information you need in order to perform a REST operation is obviously the correct URL.

Debugging

The most important piece of information you need in order to perform a REST operation is obviously the correct URL. We've mentioned a lot of the most important ones, and for many of the others, you can refer to the SharePoint SDK. Because the REST interface is modeled on the client object model, you can refer to the JavaScript object model reference for information about REST endpoint URLs. For example, if you want to see what URLs are available for working with list collections, you can refer to the reference documentation for the `SPListCollection` object at bit.ly/108h1e.

You can also navigate to a REST URL as a logged-in user and view the XML output of any GET request in order to see the available data at each endpoint and how it's structured. This won't help you with POST requests, but it can help you familiarize yourself with the different SharePoint entities and the information available at each endpoint.

It's important that the HTTP requests you send from your code contain correctly encoded URLs. When you launch an app for SharePoint from SharePoint, you can retrieve an encoded URL

from the SPHostUrl query string argument, but in other contexts you might have to encode the URL yourself.

When you're doing more complex operations—especially when you're performing operations that require the POST HTTP verb—you'll need to use an HTTP tracing utility in order to debug your HTTP requests. SharePoint returns error messages whenever you make a bad request, and those messages can tell you a lot about what's going wrong with your requests. For example, your application or user may simply not be authorized to get certain kinds of information from SharePoint. At other times, you may have constructed an invalid JSON object, or you might have assigned an invalid value to a property.

It's important that the HTTP requests you send from your code contain correctly encoded URLs.

Some frameworks provide HTTP tracing utilities for you. You can use trace.axd (bit.ly/8bnst4) when you're working with ASP.NET applications. If you're sending requests directly from your browser, as with JavaScript, you can use Fiddler (fiddler2.com/fiddler2). We used Fiddler to generate the sample HTTP responses we included in this article.

Using REST to Talk to SharePoint in a PHP Application

As we said in our introduction, the REST interface allows you to interact with SharePoint from any of the standard languages and frameworks Web developers commonly use. In order to demonstrate this, we've published a sample PHP application that demonstrates how you can interact with a SharePoint site from a remote Web application written in PHP. This particular application is an app for SharePoint that's designed to be launched from an Office 365 SharePoint site and run from a Windows Azure Web Site. This architecture simplifies some steps, such as publishing the Web site, but the PHP code in the sample can run on any architecture that supports PHP.

You can view and download the sample from the code gallery page at bit.ly/ZQsmvP. The sample demonstrates a number of things, including how to work with files and folders with REST, how to obtain a valid OAuth access token from a PHP application, and how to use the JavaScript cross-domain library. Most important for this context, it demonstrates how to retrieve files from and upload files to a SharePoint document library using REST and PHP.

Because the application writes data to a SharePoint site, one of the first things the application needs to do (after obtaining an access token) is request the form digest from `_api/contextinfo`. The request passes the access token in the headers and sets up a POST request to an SSL URL. The code, shown in **Figure 4**, will be familiar to anyone who has worked with PHP client URL objects.

After executing the request, the application parses the XML and stores the form digest value:

```
$root = new SimpleXmlElement($result);
$ns = $root->getNamespaces(TRUE);
$childrenNodes = $root->children($ns['d']);
$formValue = $childrenNodes->FormDigestValue;
```

It also stores the access token in a cookie. When the user chooses to upload a file, the application passes those values to an HTML file that constructs the HTTP request for uploading files to a document library. Before setting up the request, it reads the data from the locally stored file into a string object that will be passed as the body of the POST request:

```
$accessToken = $_COOKIE["moss_access_token"];
$url = $_REQUEST["target"];
$furl = $_FILES["file"]["tmp_name"];
$file = fopen($furl, "r");
$post_data = fread($file, filesize($furl));
fclose($file);
```

The lines in **Figure 5** retrieve the form digest and access token values and set up and execute the HTTP request that uploads the file.

What Next?

Although it doesn't have complete parity with the client object model, the SharePoint 2013 REST interface is extensive and powerful enough to provide most of what Web and mobile app developers will want to do, especially if they're working with frameworks other than .NET. We've looked at many of the most important ways in which you can integrate SharePoint into your applications by using the REST interface, but there are many more possibilities.

The REST interface allows you to interact with SharePoint from any of the standard languages and frameworks Web developers commonly use.

The SharePoint 2013 SDK contains a collection of resources for REST development, and you can find links to all of them on the MSDN Library page, "SharePoint 2013 REST API, endpoints and samples" (bit.ly/137q9yk). This collection will grow, and it will contain an especially wide range of samples because, as the PHP sample demonstrates, the REST interface significantly expands the world of SharePoint development. ■

JIM CROWLEY is a senior programming writer in the Office Division. He writes developer documentation for SharePoint 2013. He also wrote the SharePoint Developer Documentation RSS Reader app (bit.ly/YIVbvY) for Windows Phone 7.

RICKY KIRKHAM is a senior programming writer in the Office Division and a Microsoft Certified Professional Developer (SharePoint 2010). Kirkham has been on the Microsoft developer documentation team for SharePoint since 2006.

THANKS to the following technical expert for reviewing this article:
Jyoti Jacob (Microsoft)

Shortest-Path Graph Analysis Using a CLR Stored Procedure

James McCaffrey

Graph analysis is becoming increasingly important in software applications. Here a graph is a collection of nodes and edges, not a data visualization such as a bar chart. This article presents a demonstration of how to perform shortest-path analysis using a SQL CLR stored procedure. The techniques presented here can also be used for many other data-access programming tasks.

Shortest-path graph analysis really involves two closely related problems. The first is to determine the shortest path from a specified graph start node to an end node in terms of number of hops. The second problem is to determine the length of the shortest path if graph connections have some kind of weight. For example, suppose the nodes in a graph represent people and the edges between

nodes represent an e-mail communication. You might be interested in the shortest number of hops between two people, where you're implicitly assuming that each hop has a weight of 1. This is similar to the "six degrees of Kevin Bacon" game or the Erdos number for a researcher. If each graph edge has a weight—for example, representing some measure of importance of a relationship—you might want to determine the shortest path between two people taking the importance of weight into account.

But why use a CLR stored procedure? Traditional shortest-path algorithms assume that the problem graph representation can be completely stored in machine memory, typically in a matrix or adjacency list. For large graphs—for example, graphs representing social networks—this approach often isn't feasible. Large graphs can be conveniently stored in a SQL database. One approach for performing shortest-path analysis of graphs stored in a SQL database is to write a native SQL language stored procedure. The *MSDN Magazine* article, "Graph-Based Shortest-Path Analysis with SQL" (msdn.microsoft.com/magazine/jj863138), explains that approach in detail. However, using a CLR stored procedure instead of a pure SQL approach can provide dramatically better performance and much greater flexibility for customization.

Take a look at the dummy graph representation in **Figure 1**. The graph has eight nodes. The numbers next to each arrow represent a weight. The shortest path between node 222 and node 444 is 222 -> 555 -> 666 -> 777 -> 444, which has a weighted distance $1.0 + 1.0 + 1.0 + 2.0 = 5.0$. Notice that 222 -> 333 -> 666 -> 777 -> 444 is also a shortest path from 222 to 444.

This article discusses:

- Creating a sample database
- Creating a CLR stored procedure
- Coding your own priority queue
- Creating a shortest-path algorithm stored procedure
- Building, deploying and calling the stored procedure

Technologies discussed:

CLR, C#, SQL Server

Code download available at:

archive.msdn.microsoft.com/mag201305Graph

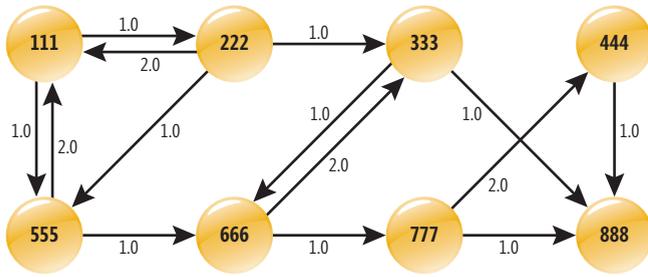


Figure 1 Dummy Graph for Shortest-Path Analysis

Figure 2 shows a screenshot of a call to a CLR stored procedure named `csp_ShortestPath` that determines the shortest path between node 222 and node 444. In this case the shortest path is displayed as a semicolon-delimited string in reverse order. The output is at the bottom of the image. The top part of the image contains SQL statements that create a graph corresponding to the one in Figure 1.

This article assumes you have advanced C# programming skills and a basic familiarity with SQL. I present all the SQL code to create the dummy graph and all the C# code to create the CLR stored procedure, and I also describe several ways to call the CLR stored procedure. All the code for this article is available at archive.msdn.microsoft.com/mag201305Graph.

Creating the Database

To create the dummy SQL-based graph, I launched Microsoft SQL Server Management Studio (SSMS) and connected to an instance of a SQL Server 2008 database. CLR stored procedures are supported by SQL Server 2005 and later. First, I created a database named `dbShortPathWithCLR` using the following commands:

```
use master
go

if exists(select * from sys.sysdatabases where name='dbShortPathWithCLR')
drop database dbShortPathWithCLR
go

create database dbShortPathWithCLR
go

use dbShortPathWithCLR
go
```

I strongly recommend experimenting with a dummy database rather than with a production database. The commands to create a table that hold node and edge data are:

```
create table tblGraph
(
    fromNode bigint not null,
    toNode bigint not null,
    edgeWeight real not null
)
```

Node IDs are stored as SQL type `bigint`, which roughly corresponds to C# type `long`. The edge weights are stored as type `real`, which is a synonym for SQL type `float(24)`, which roughly corresponds to C# type `double`. In many situations you won't be concerned with an edge weight and the `edgeWeight` column can be omitted.

The 14 statements in Figure 3 define the graph.

If you compare the statements in Figure 3 with the image in Figure 1 you'll see that each statement corresponds to an edge between nodes.

Next, the database is prepared for shortest-path analysis:

```
create nonclustered index idxTblGraphFromNode on tblGraph(fromNode)
go
create nonclustered index idxTblGraphToNode on tblGraph(toNode)
go
create nonclustered index idxTblGraphEdgeWeight on tblGraph(edgeWeight)
go

alter database dbShortPathWithCLR set trustworthy on
go
select is_trustworthy_on from sys.databases
where name = 'dbShortPathWithCLR'
go
```

The first three statements create indexes on the `fromNode`, `toNode` and `edgeWeight` columns. When working with large graphs, indexes are almost always necessary to give reasonable performance. The next two statements alter the database so the `trustworthy` property is set to "on." The default value of the property is "off." I'll explain why the `trustworthy` property must be set to "on" shortly.

At this point the dummy SQL-based graph is created. The next step is to use Visual Studio to create a CLR stored procedure to perform shortest-path analysis.

Creating the CLR Stored Procedure

To create the CLR shortest-path stored procedure, I launched Visual Studio 2010. To create CLR stored procedures your development environment must include the Microsoft .NET Framework 3.5 or later. From the File | New | Project menu I selected the Database | SQL Server project template group and then selected the Visual C# SQL CLR Database Project option as shown in Figure 4. I named the project `CreateStoredProc`.

One approach for performing shortest-path analysis of graphs stored in a SQL database is to write a native SQL language stored procedure.

Notice that the .NET Framework 3.5 is selected in the New Project dialog dropdown control. The version of the framework to target must match the version of the framework on the SQL Server hosting the database. Because the dummy database is on an instance of SQL Server 2008, I targeted .NET Framework 3.5. If the dummy database had been on an instance of SQL Server 2005, I would've targeted .NET Framework 2.0. The CLR stored procedure documentation is a bit murky in describing the correlations between the framework versions on the development environment, on the SQL Server and in the C# stored procedure creation project. You might have to resort to a bit of trial and error.

After clicking OK to create the CLR stored procedure creation project, Visual Studio prompts with a request for information about the name and authentication model of the target database (see Figure 5). After clicking OK on the New Database Reference dialog, Visual Studio loads a new project but doesn't directly

create a template. To generate a template, right-click on the project name—CreateStoredProc in this case—and select Add | New Item from the context menu (see **Figure 6**).

I selected the Stored Procedure item and named it `csp_ShortestPath.cs`. This name, without the `.cs` extension, will become the name of the stored procedure in the target database. As a matter of style, I generally prepend CLR stored procedure names with `csp` to distinguish them from system stored procedures (`sp`), extended stored procedures (`xp`) and user-defined stored procedures (`usp`). After clicking the Add button, Visual Studio generated a lightweight template of a partial class named `StoredProcedures`:

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Data.SqlTypes;
using Microsoft.SqlServer.Server;

public partial class StoredProcedures
{
    [Microsoft.SqlServer.Server.SqlProcedure]
    public static void csp_ShortestPath()
    {
        // Put your code here
    }
};
```

A Poor Man's Priority Queue

The shortest-path algorithm presented in this article requires a random-access priority queue data structure. The .NET Framework doesn't supply a built-in priority queue that will exactly meet the needs of the shortest-path algorithm, so you must code your own priority queue.

A priority queue is similar to a normal first-in-first-out (FIFO) queue with a few differences. The items in a priority queue are assumed to have some sort of priority field in addition to a data field. For example, a priority queue item for a group of customers waiting for technical support might consist of a customer's name and the length of time the customer has been waiting for service.

Priority queues support a Dequeue operation that always removes the item with the highest priority. Here, the meaning of highest depends on the problem context and can be a largest value or a smallest value. A random-access priority queue supports an operation that modifies the priority field of an item with a specified ID.

This article presents a poor man's priority queue—one that gets the job done but has much room for improvement. The shortest-path algorithm operates on a priority queue where items in the queue have two fields: a node ID (such as 111 or 222) and a distance field. The distance field is used by the algorithm to store the best (shortest) known distance between the start node and the item node at any given time during

the algorithm's execution. The distance field acts as the item's priority, and smaller values for distance represent higher priority.

So, to support a random-access priority queue, the C# stored procedure template needs an additional using statement that references the `System.Collections.Generic` namespace and two additional program-defined class definitions placed below the partial `StoredProcedures` class:

```
public class NodeDistance
{
    // Definition here
}

public class MyPriorityQueue
{
    // Definition here
}

Class NodeDistance is simple:

public class NodeDistance
{
    public long nodeID;
    public double distance;

    public NodeDistance(long nodeID, double distance)
    {
        this.nodeID = nodeID;
        this.distance = distance;
    }
}
```

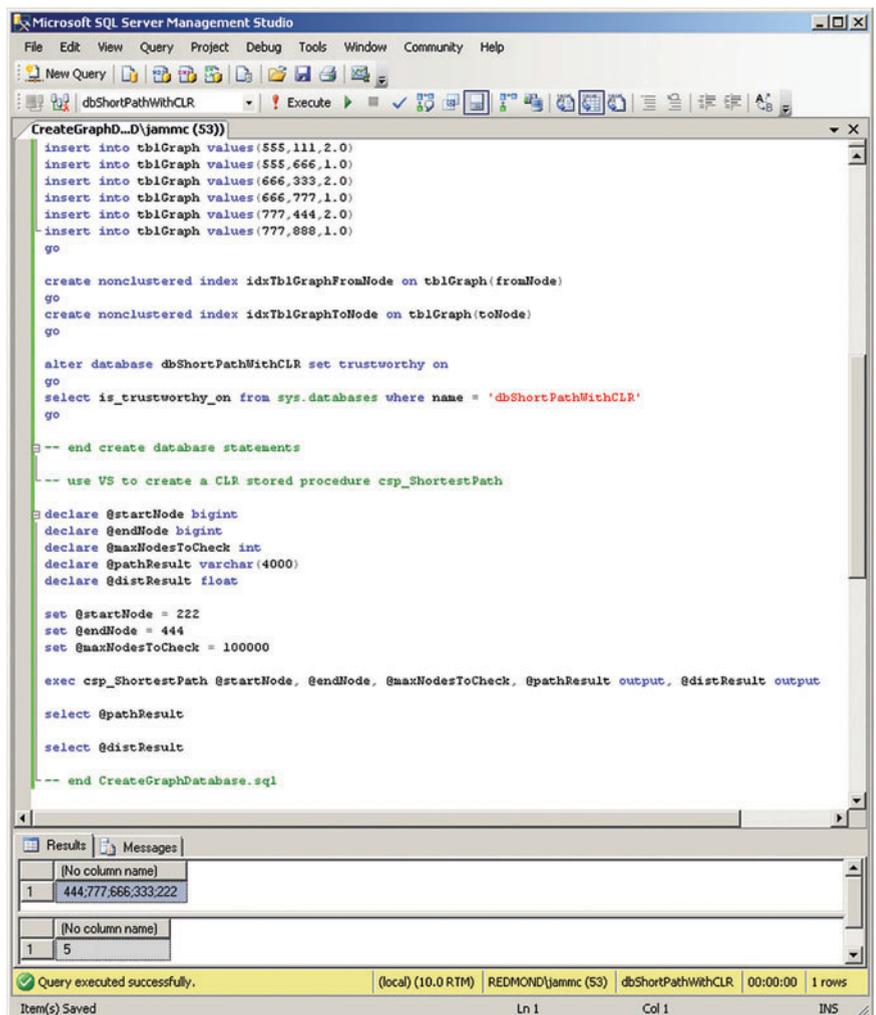


Figure 2 Calling a Shortest-Path CLR Stored Procedure

Figure 3 Defining the Graph

```
insert into tblGraph values(111,222,1.0)
insert into tblGraph values(111,555,1.0)
insert into tblGraph values(222,111,2.0)
insert into tblGraph values(222,333,1.0)
insert into tblGraph values(222,555,1.0)
insert into tblGraph values(333,666,1.0)
insert into tblGraph values(333,888,1.0)
insert into tblGraph values(444,888,1.0)
insert into tblGraph values(555,111,2.0)
insert into tblGraph values(555,666,1.0)
insert into tblGraph values(666,333,2.0)
insert into tblGraph values(666,777,1.0)
insert into tblGraph values(777,444,2.0)
insert into tblGraph values(777,888,1.0)
go
```

I use public scope for the class fields for simplicity. The priority queue is essentially a List of NodeDistance items:

```
public class MyPriorityQueue
{
    public List<NodeDistance> list;

    public MyPriorityQueue()
    {
        this.list = new List<NodeDistance>();
    }
}
```

Again, I use public scope for simplicity only. The Dequeue operation removes the NodeDistance item with the smallest distance value:

```
public NodeDistance Dequeue()
{
    int i = IndexOfSmallestDist();
    NodeDistance result = this.list[i];
    this.list.RemoveAt(i);
    return result;
}
```

Method Dequeue calls a helper method to find the location of the item that has the smallest distance, saves a reference to that item and then uses the built-in List.RemoveAt method to remove the item.

Helper method IndexOfSmallestDist is defined as:

```
private int IndexOfSmallestDist() {
    double smallDist = this.list[0].distance;
    int smallIndex = 0;
    for (int i = 0; i < list.Count; ++i) {
        if (list[i].distance < smallDist) {
            smallDist = list[i].distance;
            smallIndex = i;
        }
    }
    return smallIndex;
}
```

The method does a simple linear search through the underlying List collection. Note that this approach means that Dequeue has O(n) performance.

The Enqueue operation is defined as:

```
public void Enqueue(NodeDistance nd)
{
    list.Add(nd);
}
```

The change-priority operation is defined as:

```
public void ChangePriority(long nodeID, double newDist)
{
    int i = IndexOf(nodeID);
    list[i].distance = newDist;
}
```

Method ChangePriority calls helper method IndexOf that locates the position of a NodeDistance item, given the item's ID, and is defined as:

```
private int IndexOf(long nodeID)
{
    for (int i = 0; i < list.Count; ++i)
        if (list[i].nodeID == nodeID)
            return i;
    return -1;
}
```

Again, note that because the IndexOf method performs a linear search, its performance is O(n).

The shortest-path algorithm needs to know the number of items in the priority queue at any given time:

```
public int Count()
{
    return this.list.Count;
}
```

To summarize, the poor man's random-access priority queue defined here supports an Enqueue operation; a Count property; a Dequeue operation that removes the NodeDistance item with the smallest distance; and a ChangePriority operation that modifies the distance of a specified item. Operations Dequeue and ChangePriority have O(n) performance.

For large graphs, the performance of the shortest-path algorithm is highly dependent on the efficiency of the priority queue used. While O(n) performance is often acceptable, it's possible to implement a priority queue that has much better O(log n) performance. Such implementations typically use a binary heap data structure and are quite tricky. I describe an efficient priority queue in my *Visual Studio Magazine* article, "Priority Queues with C#," available at bit.ly/QWU1Hv.

Creating the Stored Procedure

With the custom priority queue defined, the shortest-path algorithm can be implemented. The method signature is:

```
public static void csp_ShortestPath(System.Data.SqlTypes.SqlInt64
    startNode, SqlInt64 endNode, SqlInt32 maxNodesToCheck,
    out SqlString pathResult, out SqlDouble distResult)
{
}
```

Method csp_ShortestPath accepts three input parameters and has two out result parameters. Parameter startNode holds the node ID of the node to begin the search. Recall that in the SQL table definition, columns fromNode and toNode are defined as SQL type bigint, which corresponds to C# type long. When defining a CLR stored

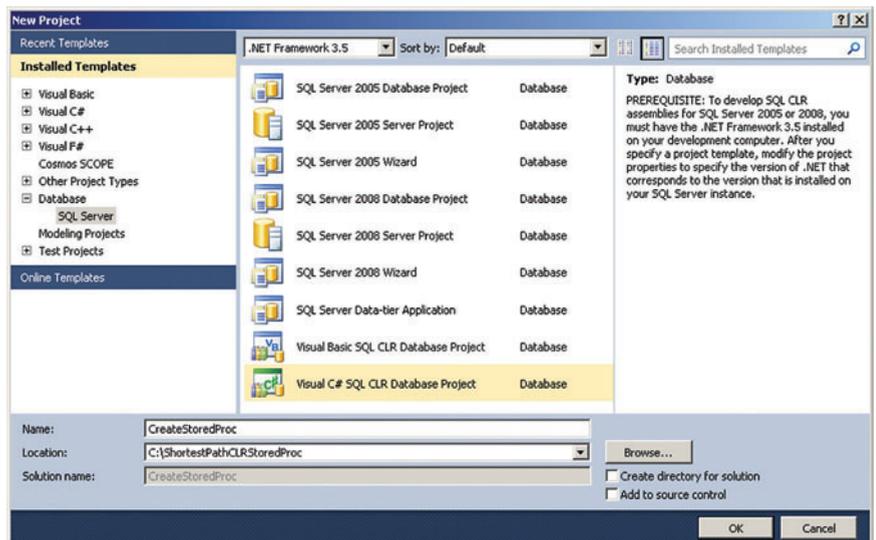


Figure 4 A New CLR Project in Visual Studio 2010

procedure, the procedure's parameters use a type model defined in namespace System.Data.SqlTypes. These types are usually pretty easy to map to SQL types and C# types. Here, type bigint maps to SqlInt64. Input parameter endNode is also declared as type SqlInt64.

Input parameter maxNodesToCheck is used to prevent the stored procedure from spinning out of control on extremely large graphs. Here, maxNodesToCheck is declared as type SqlInt32, which corresponds to C# type int.

If you're new to SQL stored procedures, you might be surprised to learn that they can have no explicit return value, or can return an int value, but they can't explicitly return any other data type. So in situations where a SQL stored procedure must return two or more values, or return a value that isn't an int type, the approach taken is to use out parameters. Here, the CLR stored procedure returns the shortest path as a string, such as "444;777;666;333;222," and also returns the total distance of the shortest path as a numeric value, such as 5.0. So out parameter pathResult is declared as type SqlString and out parameter distResult is declared as type SqlDouble, corresponding to C# types string and double, respectively.

The CLR stored procedure definition continues by setting up four data structures:

```
Dictionary<long, double> distance = new Dictionary<long, double>();
Dictionary<long, long> previous = new Dictionary<long, long>();
Dictionary<long, bool> beenAdded = new Dictionary<long, bool>();
MyPriorityQueue PQ = new MyPriorityQueue();
```

As the shortest-path algorithm executes, at any given point, the algorithm needs to access the current best (shortest) known total distance from the start node to all other nodes. The Dictionary collection named "distance" holds this information. The dictionary key is a node ID and the dictionary value is the shortest known total distance. The Dictionary collection named "previous" stores the immediate predecessor node ID to a key node ID in the shortest path. For example, in the example shown in Figure 2, the end node is 444. Its immediate predecessor in the shortest path is 777. So previous[444] = 777. In essence the previous collection holds the actual shortest path in an encoded way.

The Dictionary collection named "been-Added" holds information that indicates whether a graph node has been added to the algorithm's priority queue. The Boolean value is a dummy value because it isn't really needed to determine if a node is in the collection. You might want to use a Hashtable instead of a Dictionary collection. The custom priority queue named "PQ" was defined and explained in the previous section of this article.

The stored procedure definition continues:

```
SqlConnection conn = null;
long startNodeAsLong = (long)startNode;
long endNodeAsLong = (long)endNode;
int maxNodesToCheckAsInt = (int)
maxNodesToCheck;
```

The SqlConnection object is the single connection to the target graph database. I declare it here so that I can check its state

later if an Exception occurs. Although not strictly necessary, when writing CLR stored procedures I prefer to explicitly create local C# typed variables that correspond to the SQL typed parameter variables.

The definition continues:

```
distance[startNodeAsLong] = 0.0;
previous[startNodeAsLong] = -1;
PQ.Enqueue(new NodeDistance(startNodeAsLong, 0.0));
beenAdded[startNodeAsLong] = true;
```

These lines of code initialize the start node. The value in the distance Dictionary is set to 0.0 because the distance from the start node to itself is zero. The immediate predecessor to the start node doesn't exist so a value of -1 is used to indicate this. The priority queue is initialized with the start node, and the beenAdded dictionary collection is updated.

The definition continues:

```
try
{
    string connString = "Server=(local);Database=dbShortPathWithCLR;" +
        "Trusted_Connection=True;MultipleActiveResultSets=True";

    conn = new SqlConnection(connString);
    conn.Open();

    double alt = 0.0; // 'Test distance'
```

When writing CLR stored procedures I prefer using explicit try-catch blocks rather than the more elegant using statement. When setting up the connection string you have two options. In many cases, because the stored procedure is running on the same machine as the SQL database, you can simply set the connection string to "context connection=true." A context connection in theory will deliver better performance than a standard connection. However, a context connection has several limitations. One limitation is that it can support only a single SQL data reader. As you'll see shortly, shortest-path analysis often (but not always) requires two SQL data readers.

Because this CLR stored procedure requires two readers, a standard connection string that includes a clause that sets the MultipleActiveResultSets property to true is used. This clause isn't currently supported by SQL context connections. Because the stored procedure is using a standard connection rather than a context connection, the Database Permission Level for the Visual Studio stored procedure creation project must be set to External, as shown in Figure 7. However, in order to set this property the SQL database must have its trustworthy property set to "on," as shown back in Figure 2.

To summarize, when creating the graph database, the database trustworthy property is set to "on." This allows the Visual Studio project Database Permission Level property to be set to External. This allows the stored procedure definition to use a standard connection rather than a context connection. This allows the connection's MultipleActiveResultSets property to be set to true. And this allows two SQL data readers in the stored procedure.



Figure 5 New Database Reference for the Project

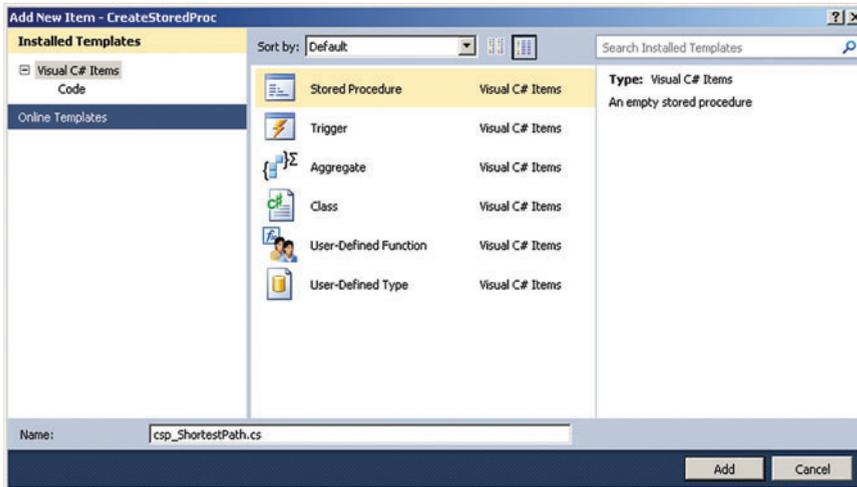


Figure 6 New Item Stored Procedure

The stored procedure definition continues:

```
while (PQ.Count() > 0 && beenAdded.Count < maxNodesToCheckAsInt)
{
    NodeDistance u = PQ.Dequeue();
    if (u.nodeID == endNode) break;
```

The algorithm used here is a variation of Dijkstra's shortest path, one of the most famous in computer science. Although short, the algorithm is very subtle and can be modified in many ways. The heart of the algorithm is a loop that terminates when the priority queue is empty. Here, an additional sanity check based on the total number of graph nodes processed is added. Inside the main loop, the priority queue Dequeue call returns the node in the queue that has the best (shortest) known total distance from the start node. If the node just removed from the priority queue is the end node, then the shortest path has been found and the loop terminates.

The definition continues:

```
SqlCommand cmd = new SqlCommand(
    "select toNode from tblGraph where fromNode=" + u.nodeID);
cmd.Connection = conn;
long v; // ID of a neighbor to u
SqlDataReader sdr = cmd.ExecuteReader();
while (sdr.Read() == true) {
    v = long.Parse(sdr[0].ToString());
    if (beenAdded.ContainsKey(v) == false) {
        distance[v] = double.MaxValue;
        previous[v] = -1;
        PQ.Enqueue(new NodeDistance(v, double.MaxValue));
        beenAdded[v] = true;
    }
}
```

These lines of code get all neighbors to the current node *u*. Notice that this requires a first SQL data reader. Each neighbor node *v* is checked to see if it's the first appearance of the node in the algorithm. If so, a *NodeDistance* item with *v* as its *nodeID* is instantiated and added to the priority queue. Instead of adding nodes to the priority queue as they're encountered, an alternative design is to initially add all nodes in the graph to the priority queue. However, for very large graphs this could require a very large amount of machine memory and take a very long time.

The inner read-all-neighbors loop continues:

```
SqlCommand distCmd =
    new SqlCommand("select edgeWeight from tblGraph where fromNode=" +
        u.nodeID + " and toNode=" + v);
distCmd.Connection = conn;
double d = Convert.ToDouble(distCmd.ExecuteScalar());
alt = distance[u.nodeID] + d;
```

This code queries the database to get the distance from the current node *u* to the current neighbor node *v*. Notice a second data reader is needed to do this. The existence of the second data reader necessitates the multiple changes to properties including the database trustworthy property and the Visual Studio project Permission property. If your shortest-path analysis uses an unweighted graph—that is, one where all edge weights are assumed to be 1—you can simplify by eliminating the second reader and substituting

```
alt = distance[u.nodeID] + 1;
for
double d = Convert.ToDouble(distCmd.ExecuteScalar());
alt = distance[u.nodeID] + d;
```

Variable *alt* is a test distance from the start node to the current neighbor node *v*. If you

examine the assignment statement carefully, you'll see that *alt* is the shortest known distance from the start node to node *u* plus the actual distance from node *u* to node *v*. This represents a potential new shorter distance from the start node to node *v*.

The inner read-all-neighbors loop and the main algorithm loop continue with:

```
if (alt < distance[v])
{
    distance[v] = alt;
    previous[v] = u.nodeID;
    PQ.ChangePriority(v, alt);
}
} // sdr Read loop
sdr.Close();
} // Main loop
conn.Close();
```

If the test distance from the start node to *v* is smaller than the shortest known distance from the start node to *v* (stored in the distance Dictionary), then a new shorter distance from start to *v* has been found and local data structures *distance*, *previous* and the priority queue are updated accordingly.

The stored-procedure logic now determines if the main algorithm loop terminated because a shortest path was in fact found, or terminated because no path between start node and end was found:

```
pathResult = "NOTREACHABLE";
distResult = -1.0;
if (distance.ContainsKey(endNodeAsLong) == true) {
    double sp = distance[endNodeAsLong];
    if (sp != double.MaxValue) {
        pathResult = "";
        long currNode = endNodeAsLong;
        while (currNode != startNodeAsLong) {
            pathResult += currNode.ToString() + ";";
            currNode = previous[currNode];
        }
        pathResult += startNode.ToString();
        distResult = sp;
    }
}
```

Recall that there are really two results to this shortest-path implementation: the shortest path expressed as a semicolon-delimited string in reverse order, and the shortest path measured by the sum of the edge weights between nodes in the shortest path. The stored procedure uses defaults of "NOTREACHABLE" and -1.0 for situations where the end node isn't reachable from the start node. The while loop extracts the shortest-path nodes from the

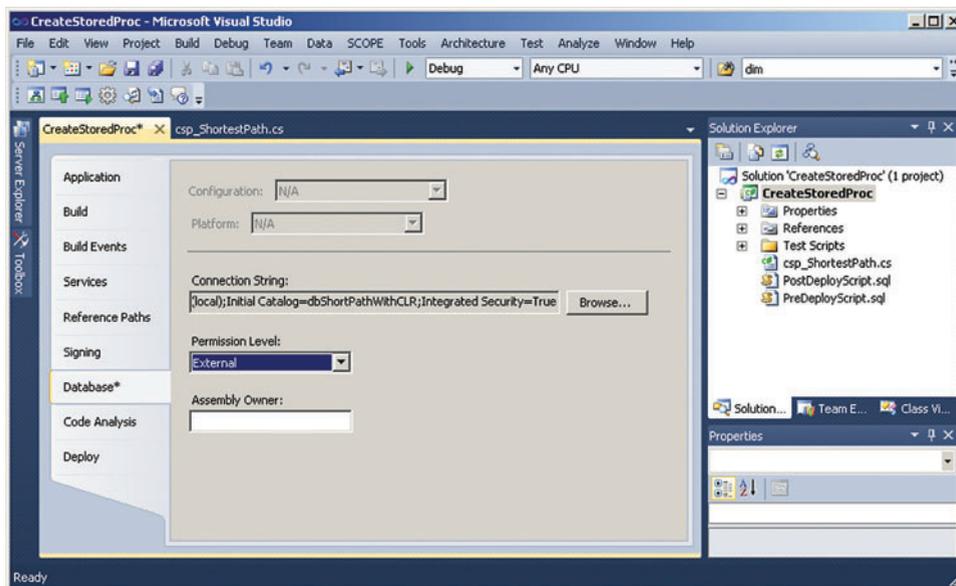


Figure 7 Setting Database Permission Level

previous Dictionary and stitches them together from end node to start node using string concatenation. If you're ambitious you can use a stack and construct the result string from start node to end node. Recall that the two results are returned via out parameters pathResult and distResult.

The stored procedure definition concludes by checking for errors:

```

} // Try

catch(Exception ex)
{
    pathResult = ex.Message;
    distResult = -1.0;
}

finally
{
    if (conn != null && conn.State == ConnectionState.Open)
        conn.Close();
}
} // csp_ShortestPath()

```

If an Exception has occurred—typically if the SQL Server runs out of memory due to an extremely large graph or due to a SQL connection time-out—the code attempts to clean the SQL connection up and return results that have some meaning.

Figure 8 Calling a Stored Procedure from Within C# Using ADO.NET

```

SqlConnection sc = null;
string connString = "Server=" + dbServer + ";Database=" +
    database + ";Trusted_Connection=True";
sc = new SqlConnection(connString);
SqlCommand cmd = new SqlCommand("csp_ShortestPath", sc);
cmd.CommandType = System.Data.CommandType.StoredProcedure; // sp
signature: (System.Data.SqlTypes.SqlInt64 startNode, SqlInt64 endNode,
    SqlInt32 maxNodesToCheck, out SqlString path)
cmd.CommandTimeout = commandTimeout; // Seconds
SqlParameter sqlStartNode = cmd.Parameters.Add("@startNode",
    System.Data.SqlDbType.BigInt);
sqlStartNode.Direction = ParameterDirection.Input;
sqlStartNode.Value = sn;
// ...
cmd.ExecuteNonQuery();
string result = (string)cmd.Parameters["@pathResult"].Value;
distResult = (double)cmd.Parameters["@distResult"].Value;

```

Building, Deploying and Calling the Stored Procedure

After making sure you've set the database trustworthy property to "on" and the Database Permission Level to External, select Build CreateStoredProc from the Visual Studio Build menu. If the build succeeds, select Deploy CreateStoredProc from the Build menu to copy the CLR stored procedure from your development machine to the SQL Server that hosts the SQL-based graph.

There are several ways to call the CLR stored procedure. The Visual Studio project contains a test template you can use. Or you can call the stored procedure directly from

SSMS as shown in Figure 2. For example:

```

declare @startNode bigint
declare @endNode bigint
declare @maxNodesToCheck int
declare @pathResult varchar(4000)
declare @distResult float

set @startNode = 222
set @endNode = 444
set @maxNodesToCheck = 100000

exec csp_ShortestPath @startNode, @endNode, @maxNodesToCheck,
    @pathResult output, @distResult output

select @pathResult
select @distResult

```

Another option is to call the CLR stored procedure from within a C# application using ADO.NET, along the lines of Figure 8.

Or, if you're a real glutton for punishment, you can call the CLR stored procedure using LINQ technology.

More Than Just Shortest Path

The code and explanation presented here should allow you to tackle shortest-path graph analysis using a CLR stored procedure. Additionally, you might find the design paradigm useful in general. Instead of fetching data from SQL to a client application and then doing filtering and complex processing on the client, use a CLR stored procedure to fetch, filter and process on the server—and then transfer results to the client. I've found this approach to be extremely useful in several situations with large databases and requirements for real-time performance. ■

DR. JAMES McCAFFREY works for Microsoft Research in Redmond, Wash. He has worked on several key Microsoft products including Internet Explorer and Bing. He holds degrees from the University of California at Irvine and the University of Southern California. His personal blog is at jamesmccaffrey.wordpress.com. He can be reached at 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...

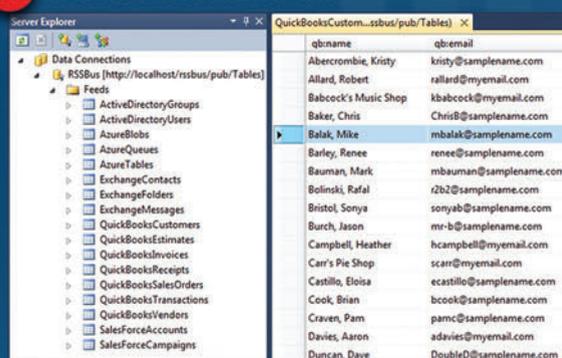
The RSSBus Data Providers make everything look like a SQL table, even local application data. Using the RSSBus Data Providers your .NET applications interact with local applications, databases, and services in the same way you work with SQL Tables and Stored Procedures. No code required. It simply doesn't get any easier!

*“Databind to anything...
...just like you do with SQL”*

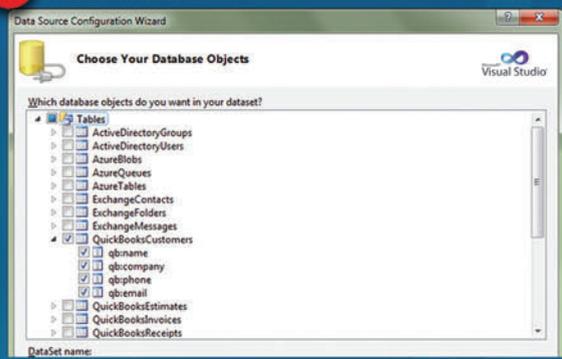
Also available for:

JDBC | ODBC | SQL SSIS | Excel | OData | SharePoint ...

1 SELECT CONNECTOR



2 DATABIND



3 GO!





Data Clustering Using Category Utility

Data clustering is the process of placing data items into different groups—clusters—in such a way that items in a particular group are similar to each other and different from those in other groups. Clustering is a machine learning technique that has many important practical uses. For example, cluster analysis can be used to determine what types of items are often purchased together so that targeted advertising can be aimed at consumers, or to determine which companies are similar so that stock market prices can be predicted.

The most basic form of clustering uses what's called the k-means algorithm. (See my article, "Detecting Abnormal Data Using k-Means Clustering," at msdn.microsoft.com/magazine/jj891054.) Unfortunately, k-means clustering can be used only in situations where the data items are completely numeric. Clustering categorical data (such as color, which can take on values such as "red" and "blue") is a challenging problem. In this article I present a previously unpublished (as far as I can determine) clustering algorithm that I designed that can handle either numeric or categorical data items, or a mixture of both. I call this algorithm Greedy Agglomerate Category Utility Clustering (GACUC) to distinguish it from the many other clustering algorithms. This article will give you all the information you need to experiment with data clustering, add clustering features to a .NET application or system, or create a powerful standalone data-clustering tool.

The best way to understand what clustering is and see where I'm headed in this article is to take a look at **Figure 1**. The demo program shown in the figure is clustering a small set of five dummy data items. Data items are sometimes called tuples in clustering terminology. Each tuple has three categorical attributes: color, length and rigidity. Color can take on one of four possible values: red, blue, green or yellow. Length can be short, medium or long. Rigidity can be false or true.

The demo program converts the raw string data into integer form for more-efficient processing. For color, red, blue, green and yellow are encoded as 0, 1, 2 and 3, respectively. For length, short, medium and long are encoded as 0, 1, 2, respectively. For rigidity, false is encoded as 0 and true is encoded as 1. So, the first raw data item, "Red Short True," is encoded as "0 0 1."

Many clustering algorithms, including GACUC, require the number of clusters to be specified. In this case, the number of clusters

is set to 2. The demo program then uses GACUC to find the best clustering of the data. Behind the scenes, the algorithm starts by placing seed tuples 0 and 4 into clusters 0 and 1, respectively. The clustering algorithm then iterates through the tuples, assigning each to the cluster that generates the best overall result. Because it doesn't use any kind of training data, clustering is called unsupervised learning. After a preliminary clustering pass, the GACUC algorithm performs a refining pass to try to improve the clustering. No improvement is found in this example.

Clustering is a machine learning technique that has many important practical uses.

Internally, the demo program defines a clustering as an array of int. The index of the array indicates a tuple index, and the cell value in the array indicates a cluster. In **Figure 1**, the best clustering obtained by the algorithm is [0,0,1,1,1], which means tuple 0 ("Red Short True") is in cluster 0; tuple 1 ("Red Long False") is in cluster 0; tuple 2 ("Blue Medium True") is in cluster 1; and so on. The demo program displays the final, best clustering in string format for readability, and also displays a key metric called the category utility (CU) of the best clustering, 0.3733 in this example. As you'll see, category utility is the key to the GACUC clustering algorithm.

This article assumes you have advanced programming skills with a C-family language but does not assume you know anything about data clustering. I coded the demo program using a non-OOP approach with the C# language, but you shouldn't have too much trouble refactoring the demo code to OOP or another language. I've removed all normal error checking for clarity. The demo program code is too long to present in its entirety in this article, so I focused on the GACUC algorithm so you'll be able to modify the demo code to meet your own needs. The complete source code for the demo program is available at archive.msdn.microsoft.com/mag201305TestRun.

Category Utility

Data clustering involves solving two main problems. The first problem is defining exactly what makes a good clustering of data. The second problem is determining an effective technique to search

Code download available at archive.msdn.microsoft.com/mag201305TestRun.

through all possible clustering combinations to find the best one. Category utility addresses the first problem. CU is a clever metric that defines a clustering's goodness. Small values of CU indicate poor clustering while larger values indicate better clustering. As far as I've been able to determine, CU was first defined by M. Gluck and J. Corter in a 1985 research paper titled "Information, Uncertainty, and the Utility of Categories."

The mathematical equation for CU is a bit intimidating at first glance:

$$CU(C) = \frac{1}{m} \sum_{k=0}^{m-1} P(C_k) \left[\sum_i \sum_j P(A_i = V_{ij} | C_k)^2 - \sum_i \sum_j P(A_i = V_{ij})^2 \right]$$

But the equation is actually simpler than it appears. In the equation, uppercase C is an overall clustering; lowercase m is the number of clusters; uppercase P means "probability of"; uppercase A means attribute (such as color); uppercase V means attribute value (such as red). Unless you're a mathematician, computing category utility is best understood by example. Suppose the data set to be clustered is the one shown in **Figure 1**, and you want to compute the CU of the clustering:

```
k = 0
Red   Short  True
Red   Long   False
```

```
k = 1
Blue  Medium True
Green Medium True
Green Medium False
```

The first step is to compute P(C_k), which are the probabilities of each cluster. For k = 0, because there are five tuples in the data set and two of them are in cluster 0, P(C₀) = 2/5 = 0.40. Similarly, P(C₁) = 3/5 = 0.60.

The second step is to compute the last double summation in the equation, called the unconditional probability term. The computation is the sum of N terms where N is the total number of different attribute values in the data set, and goes like this:

```
Red: (2/5)^2 = 0.1600
Blue: (1/5)^2 = 0.0400
Green: (2/5)^2 = 0.1600
Yellow: (0/5)^2 = 0.0000
Short: (1/5)^2 = 0.0400
Medium: (3/5)^2 = 0.3600
Long: (1/5)^2 = 0.0400
False: (2/5)^2 = 0.1600
True: (3/5)^2 = 0.3600
```

Unconditional sum = 1.3200

The third step is to compute the middle term double summation, called the conditional probability terms. There are m sums (where m is the number of clusters), each of which has N terms.

For k = 0 the computation goes:

```
Red: (2/2)^2 = 1.0000
Blue: (0/2)^2 = 0.0000
Green: (0/2)^2 = 0.0000
Yellow: (0/2)^2 = 0.0000
Short: (1/2)^2 = 0.2500
Medium: (0/2)^2 = 0.0000
Long: (1/2)^2 = 0.2500
False: (1/2)^2 = 0.2500
True: (1/2)^2 = 0.2500
```

Conditional k = 0 sum = 2.0000

For k = 1 the computation is:

```
Red: (0/3)^2 = 0.0000
Blue: (1/3)^2 = 0.1111
Green: (2/3)^2 = 0.4444
Yellow: (0/3)^2 = 0.0000
Short: (0/3)^2 = 0.0000
Medium: (3/3)^2 = 1.0000
Long: (0/3)^2 = 0.0000
False: (1/3)^2 = 0.1111
True: (2/3)^2 = 0.4444
```

Conditional k = 1 sum = 2.1111

The last step is to combine the computed sums according to the CU equation:

```
CU = 1/2 * [ 0.40 * (2.0000 - 1.3200) + 0.60 * (2.1111 - 1.3200) ]
    = 0.3733
```

A detailed explanation of exactly why CU measures the goodness of a clustering is fascinating, but unfortunately outside the scope of this article. The key point is that for any clustering of a data set containing categorical data, it's possible to compute a value that describes how good the clustering is.

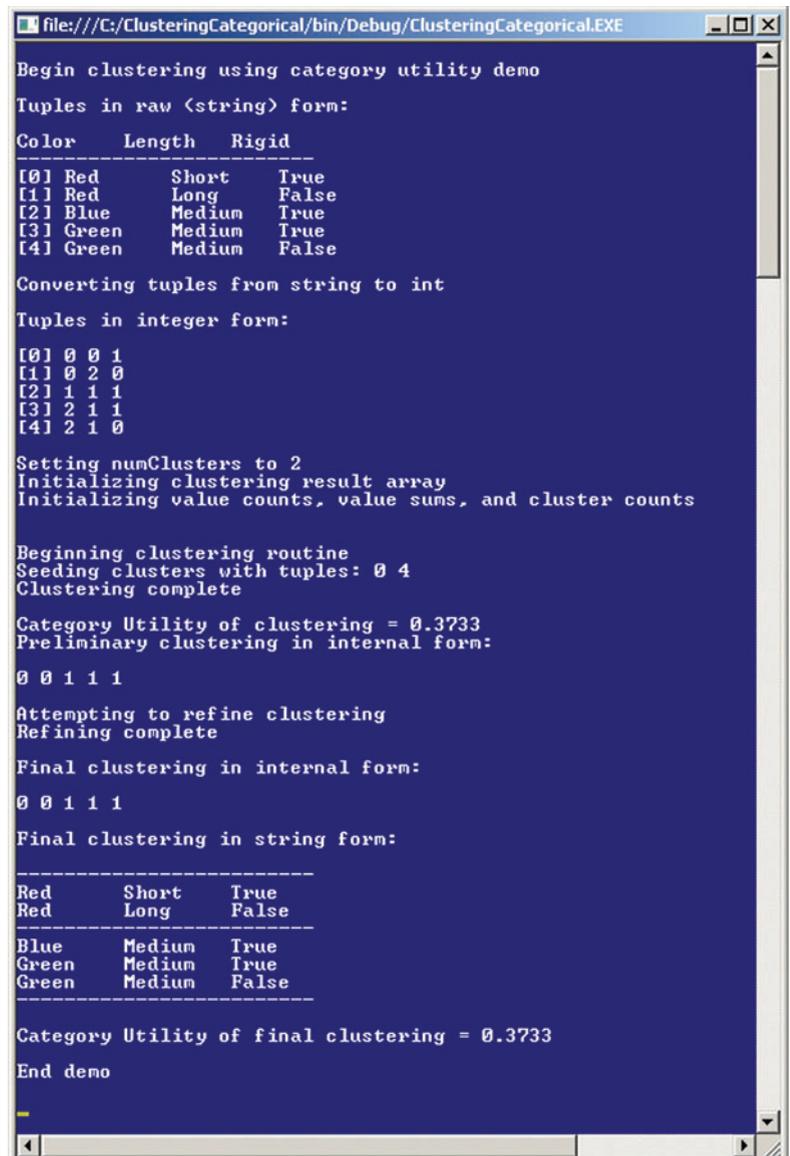


Figure 1 Clustering Categorical Data in Action

Figure 2 Overall Program Structure

```

using System;
using System.Collections.Generic;
namespace ClusteringCategorical
{
    class ClusteringCategoricalProgram
    {
        static Random random = null;
        static void Main(string[] args)
        {
            try
            {
                random = new Random(2);
                Console.WriteLine("\nBegin category utility demo\n");

                string[] attNames = new string[] { "Color", "Length", "Rigid" };

                string[][] attValues = new string[attNames.Length][];
                attValues[0] = new string[] { "Red", "Blue", "Green", "Yellow" };
                attValues[1] = new string[] { "Short", "Medium", "Long" };
                attValues[2] = new string[] { "False", "True" };

                string[][] tuples = new string[5][];
                tuples[0] = new string[] { "Red", "Short", "True" };
                tuples[1] = new string[] { "Red", "Long", "False" };
                tuples[2] = new string[] { "Blue", "Medium", "True" };
                tuples[3] = new string[] { "Green", "Medium", "True" };
                tuples[4] = new string[] { "Green", "Medium", "False" };

                Console.WriteLine("Tuples in raw (string) form:\n");
                DisplayMatrix(tuples);

                int[][] tuplesAsInt = TuplesToInts(tuples, attValues);

                Console.WriteLine("\nTuples in integer form:\n");
                DisplayMatrix(tuplesAsInt);

                int numClusters = 2;
                int numSeedTrials = 10;

                Console.WriteLine("Initializing clustering result array");
                int[] clustering = InitClustering(tuplesAsInt);

                int[][][] valueCounts = InitValueCounts(tuplesAsInt, attValues,
                    numClusters);
                int[][] valueSums = InitValueSums(tuplesAsInt, attValues);
                int[] clusterCounts = new int[numClusters];

                Console.WriteLine("\nBeginning clustering routine");
                Cluster(tuplesAsInt, attValues, clustering, valueCounts,
                    valueSums, clusterCounts, numSeedTrials);

                double cu = CategoryUtility(valueCounts, valueSums,
                    clusterCounts);
                Console.WriteLine("\nCategory Utility of clustering = " +
                    cu.ToString("F4"));
                DisplayVector(clustering);

                Refine(20, tuplesAsInt, clustering, valueCounts, valueSums,
                    clusterCounts);
                Console.WriteLine("Refining complete");
                DisplayVector(clustering);

                Console.WriteLine("\nFinal clustering in string form:\n");
                DisplayClustering(numClusters, clustering, tuples);

                cu = CategoryUtility(valueCounts, valueSums, clusterCounts);
                Console.WriteLine("\nFinal clustering CU = " + cu.ToString("F4"));

                Console.WriteLine("\nEnd demo\n");
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        } // Main

        // All other methods here

    } // class Program
} // ns

```

Searching for the Best Clustering

After defining a way to measure clustering goodness, the second problem to solve in any clustering algorithm is coming up with a technique to search through all possible clusterings for the best clustering. In general it's not feasible to examine every possible clustering of a data set. For example, for a data set with only 100 tuples, and $m = 2$ clusters, there are $2^{100} / 2! = 2^{99}$ possible clusterings. Even if you could somehow examine one trillion clusterings per second, it would take roughly 19 billion years to check every possible clustering. (By comparison, the estimated age of the universe is about 14 billion years.)

Different clustering algorithms use different techniques to search through all possible clusterings. The GACUC algorithm uses what is called a greedy agglomerative approach. The idea is to begin by seeding each cluster with a single tuple and then, for each remaining tuple, determine which cluster k , if the current tuple were added to it, would yield the best overall clustering. Then, the current tuple is actually assigned to cluster k . This technique doesn't guarantee that the optimal clustering will be found, but experiments have shown that the technique generally produces a reasonably good clustering. The final clustering produced by the GACUC algorithm depends on which m tuples are selected as seed tuples and the order in which the remaining tuples are assigned to clusters.

A better approach for selecting
seed tuples is to choose m tuples
that are as different as possible
from each other.

It turns out that selecting a seed tuple for each cluster is not trivial. One naive approach would be to simply select random tuples as the seeds. However, if the seed tuples are similar to each other, the resulting clustering will be poor. A better approach for selecting seed tuples is to choose m tuples that are as different as possible from each other. Here's where category utility comes in handy again—the CU of any potential set of candidate seed tuples can be computed, and the set of tuples with the best CU (largest value, meaning most dissimilar) can be used as the seed tuples. As before, it's generally not feasible to examine every possible set of seed tuples, so the approach is to repeatedly select m random tuples, compute the CU of those random tuples, and use the set that has the best CU as seeds.

Overall Program Structure

The Main method of the demo program shown running in **Figure 1**, with some WriteLine statements and comments removed, is listed in **Figure 2**. I used Visual Studio 2010 with the Microsoft .NET Framework 4, but the demo code has no significant dependencies and any version of Visual Studio that supports the .NET Framework 2.0 or greater should work fine. For simplicity, I

created a single C# console application named Clustering-Categorical; you might want to implement clustering as a class library.

In **Figure 2**, method `Cluster` performs one iteration of the basic GACUC clustering algorithm. Variable `numSeedTrials` is set to 10 and is passed to the routine that determines the initial seed tuples that are assigned to each cluster. Method `Refine` performs post-clustering passes through the data in an attempt to find a clustering that produces a better category utility.

The Key Data Structures

Although it's possible to compute the category utility of a data set on the fly by iterating through each tuple in the data set, because the clustering method needs to compute category utility many times, a much better approach is to store the counts of attribute values of tuples that are assigned to clusters at any given point in time. **Figure 3** shows most of the key data structures used by the GACUC algorithm.

Array `valueCounts` holds the number of attribute values by cluster. For example, `valueCounts[0][2][1]` holds the number of tuples with attribute 0 (Color) and value 2 (Green) that are currently assigned to cluster 1. Array `valueSums` holds the sum of the counts, across all clusters, for each attribute value. For example, `valueSums[0][2]` holds the total number of tuples with attribute 0 (Color) and value 2 (Green) that are assigned to all clusters. Array `clusterCounts` holds the number of tuples that are currently assigned to each cluster. For example, if `numClusters = 2` and `clusterCounts = [2,2]`, then there are two tuples assigned to cluster 0 and two tuples assigned to cluster 1. Array `clustering` encodes the assignment of tuples to clusters. The index of clustering represents a tuple and a cell value represents a cluster, and a value of -1 indicates the associated tuple has not yet been assigned to any cluster. For example, if `clustering[2] = 1`, then tuple 2 is assigned to cluster 1.

Coding the CategoryUtility Method

The code for the method that computes category utility is not conceptually difficult, but it is a bit tricky. The method's definition begins:

```
static double CategoryUtility(int[][][] valueCounts, int[][] valueSums,
    int[] clusterCounts)
{
    int numTuplesAssigned = 0;
    for (int k = 0; k < clusterCounts.Length; ++k)
        numTuplesAssigned += clusterCounts[k];

    int numClusters = clusterCounts.Length;
    double[] clusterProbs = new double[numClusters]; // P(Ck)
    for (int k = 0; k < numClusters; ++k)
        clusterProbs[k] = (clusterCounts[k] * 1.0) / numTuplesAssigned;
}
```

The three input arrays, `valueCounts`, `valueSums` and `clusterCounts` are assumed to hold valid values that reflect the current clustering, as described in the previous section and in **Figure 3**. The method begins by scanning through the `clusterCounts` array to compute the total number of tuples that are currently assigned to clusters. The number of clusters is inferred from the `Length` property of the `clusterCounts` array, and the probabilities for the clusters are then computed and stored into local array `clusterProbs`.

The next step is to compute the single unconditional probability for the current clustering:

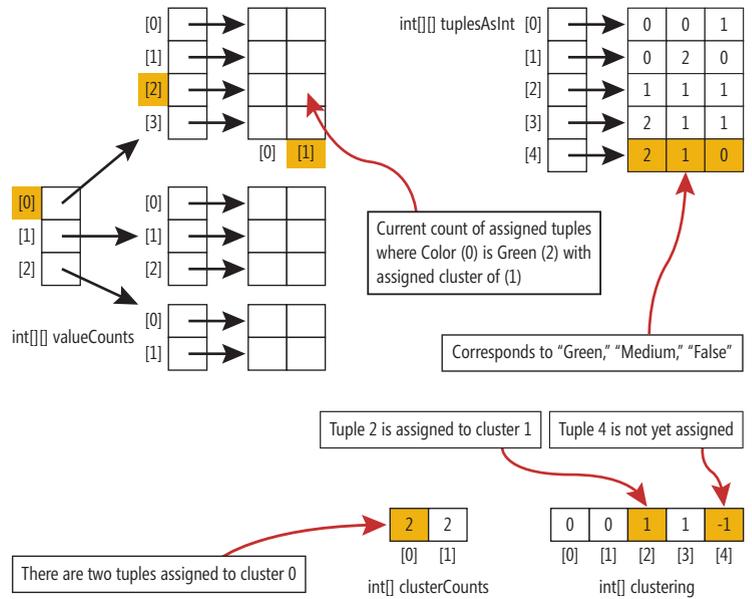


Figure 3 Key Data Structures

```
double unconditional = 0.0;
for (int i = 0; i < valueSums.Length; ++i)
{
    for (int j = 0; j < valueSums[i].Length; ++j)
    {
        double p = (valueSums[i][j] * 1.0) / numTuplesAssigned;
        unconditional += (p * p);
    }
}
```

After the unconditional probability has been computed, the next step is to compute a conditional probability for each cluster:

```
double[] conditionals = new double[numClusters];
for (int k = 0; k < numClusters; ++k)
{
    for (int i = 0; i < valueCounts.Length; ++i)
    {
        for (int j = 0; j < valueCounts[i].Length; ++j)
        {
            double p = (valueCounts[i][j][k] * 1.0) / clusterCounts[k];
            conditionals[k] += (p * p);
        }
    }
}
```

Now, with the probabilities of each cluster in array `clusterProbs`, the unconditional probability term in variable `unconditional` and the conditional probability terms in array `conditionals`, the category utility for the clustering can be determined:

```
double summation = 0.0;
for (int k = 0; k < numClusters; ++k)
    summation += clusterProbs[k] * (conditionals[k] - unconditional);

return summation / numClusters;
}
```

A good way to understand the behavior of the CU function is to experiment with the demo code by supplying hardcoded clusterings. For example, you can modify the code in **Figure 2** along the lines of:

```
string[] attNames = new string[] { "Color", "Length", "Rigid" };
// And so on, as in Figure 1
int[][] tuplesAsInt = TuplesToInts(tuples, attValues);
int[] clustering = new int[] { 0, 1, 0, 1, 0 };
// Hardcoded clustering
double cu = CategoryUtility(valueCounts, valueSums, clusterCounts);
Console.WriteLine("CU of clustering = " + cu.ToString("F4"));
```

Implementing the Cluster Method

With a category utility method in hand, implementing a method to cluster data is relatively simple. In high-level pseudo-code, method `Cluster` is:

```
define an empty clustering with all tuples unassigned
determine m good (dissimilar) seed tuples
assign one good seed tuple to each cluster
loop each unassigned tuple t
  compute the CU for each cluster if t were actually assigned
  determine the cluster that gives the best CU
  assign tuple t to best cluster
end loop (each tuple)
return clustering
```

For simplicity, method `Cluster` iterates through each unassigned tuple in order, starting at tuple [0]. This effectively gives more influence to tuples with lower-number indexes. An alternative approach I often use is to iterate through unassigned tuples in random order or in scrambled order by using a linear, congruential, full-cycle generator.

This approach generates random tuple indexes by using a brute-force generate-check-if-not-used approach, which has worked well for me in practice.

A surprisingly tricky part of the algorithm is finding m good seed tuples. In high-level pseudo-code, method `GetGoodIndexes` is:

```
loop numSeedTrials times
  create a mini-tuples array with length numClusters
  pick numClusters random tuple indexes
  populate the mini-tuples with values from data set
  compute CU of mini data set
  if CU > bestCU
    bestCU = CU
    indexes of best mini data set = indexes of data set
  end if
end loop
return indexes of best mini data set
```

This approach generates random tuple indexes by using a brute-force generate-check-if-not-used approach, which has worked well for me in practice.

Because method `Cluster` will, in general, return a good but not optimal clustering, the GACUC algorithm optionally calls a refine method that attempts to rearrange the tuple cluster assignments in an effort to find a clustering with a better category utility value. In pseudo-code, method `Refine` is:

```
loop numRefineTrials times
  select a random tuple that is in a cluster with at least two tuples
  determine the random tuple's cluster
  select a second cluster that is different from curr cluster
  unassign tuple from curr cluster
  assign tuple to different cluster
  compute new CU
  if new CU > old CU
    leave the cluster switch alone
  else
    unassign tuple from different cluster
    assign tuple back to original cluster
  end if
end loop
```

There are many other post-processing refinements with which you can experiment. The preceding refinement maintains a fixed number of clusters. One possibility is to define a refinement method that allows the number of clusters to increase or decrease.

A key helper method in the GACUC clustering algorithm is one that assigns a tuple to a cluster:

```
static void Assign(int tupleIndex, int[][] tuplesAsInt, int cluster,
  int[] clustering, int[][] valueCounts, int[][] valueSums,
  int[] clusterCounts)
{
  clustering[tupleIndex] = cluster; // Assign

  for (int i = 0; i < valueCounts.Length; ++i) // Update
  {
    int v = tuplesAsInt[tupleIndex][i];
    ++valueCounts[i][v][cluster];
    ++valueSums[i][v];
  }
  ++clusterCounts[cluster]; // Update
}
```

Notice that method `Assign` accepts a lot of parameters; this is a general weakness of using static methods rather than an OOP approach. Method `Assign` modifies the clustering array and then updates the arrays that hold the counts for each attribute value by cluster (`valueCounts`), the counts for each attribute value across all clusters (`valueSums`), and the number of tuples assigned to each cluster (`clusterCounts`). Method `Unassign` is essentially the opposite of `Assign`. The three key statements in method `Unassign` are:

```
clustering[tupleIndex] = -1; // Unassign
--valueCounts[i][v][cluster]; // Update
--valueSums[i][v]; // Update
--clusterCounts[cluster]; // Update
```

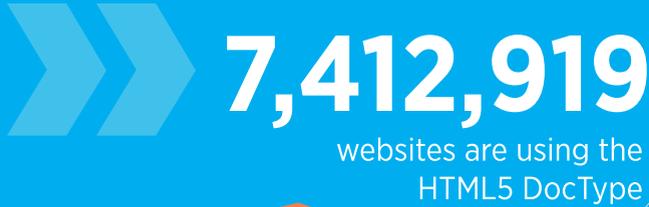
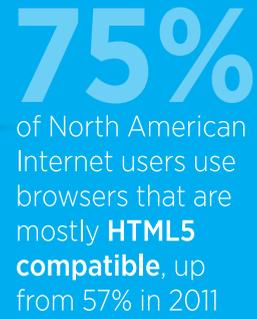
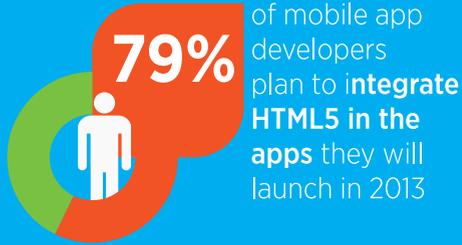
Wrapping Up

The code download that accompanies this article, along with the explanation of the GACUC clustering algorithm presented here, should get you up and running if you want to experiment with data clustering, add clustering features to an application, or create a clustering utility tool. Unlike many clustering algorithms (including k-means), the GACUC algorithm can be easily modified to deal with numeric data or mixed numeric and categorical data. The idea is to preprocess numeric data by binning it into categories. For example, if your raw data contained people's heights measured in inches, you could bin height so that values less than 60.0 are "short," values between 60.0 and 74.0 are "medium," and values greater than 74.0 are "tall."

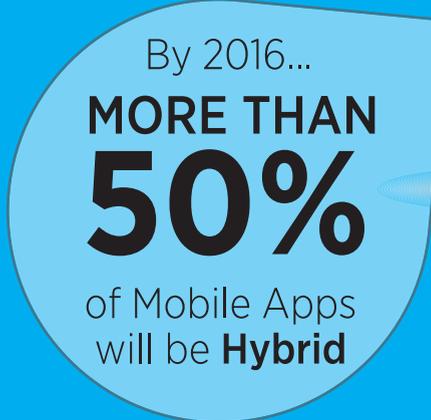
There are many algorithms available for clustering categorical data, including algorithms that rely on category utility to define clustering goodness. However, the algorithm presented here is relatively simple, has worked well in practice, can be applied to both numeric and categorical data, and scales well to huge data sets. ■

DR. JAMES McCaffrey works for Volt Information Sciences Inc., where he manages technical training for software engineers working at the Microsoft Redmond, Wash., campus. He has worked on several Microsoft products including Internet Explorer and MSN Search. He's the author of "NET Test Automation Recipes" (Apress, 2006), and can be reached at jammc@microsoft.com.

THANKS to the following technical expert for reviewing this article:
Dan Liebling (Microsoft Research)



HTML



Don't get left behind: infragistics.com/igniteui





Exploring Filters in XAudio2

In the pantheon of notable waveforms, the simple sine curve reigns supreme. Just by looking at it, you can see its quintessential smoothly undulating nature—slowing down as it reaches its peaks, almost stopping as it crests, and then progressively picking up speed, reaching maximum velocity as it crosses the horizontal axis to start another slowdown.

This visual impression is confirmed by a deeper mathematical analysis. The instantaneous velocity of the sine curve at any point is a tangent line to the curve. Graph those velocities, and you get another sine curve, offset from the original by a quarter cycle. Do it again using this second curve, and that's a sine curve showing acceleration, offset from the original by half a cycle, as shown in **Figure 1**.

In calculus terms, the sine curve is the negative of its own second derivative. From basic physics, we know that force is proportional to acceleration, which means that in any physical process where force is inversely proportional to displacement, motion is described by sine curves. Springs have this characteristic: The more you stretch them, the greater the force in the opposite direction. But many other substances found in nature have an intrinsic springiness as well, including the compression and rarefaction of air.

Consider the plucking of a taut string, the tapping of a stretched animal skin, the vibration of air within a pipe. These processes all involve springy objects that vibrate with the characteristic motion of a sine curve. More commonly, this sine curve is supplemented by additional sine curves whose frequencies are integral multiples of the fundamental frequency. The sine curves in this assemblage are known as *harmonics*.

By itself, a single sine curve is audibly quite boring. But put a few of them together in a harmonic relationship and the sound gets much more interesting. In real life, very often the frequencies of these harmonics are not exact integrals of a base frequency, and are more correctly referred to as *overtones*. It's this combination of overtones—including how they change over time—that defines a musical instrument's characteristic sound, or *timbre*.

A little fragment of a particular waveform can be graphed as a function of time, as shown on the top in **Figure 2**. If this waveform repeats every 4 ms, it has a frequency of 250 Hz, which is close to middle C on the piano.

With some Fourier analysis, we can separate this waveform into its constituent sine curves, and represent it in a somewhat different

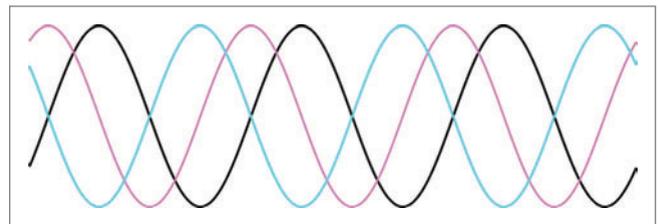


Figure 1 A Sine Curve, Its Velocity (in Violet) and Acceleration (in Aqua)

manner, as shown on the bottom in **Figure 2**. This graph shows the relative amplitudes of these constituent sine curves arranged by frequency. In signal-processing lingo, **Figure 2** shows the equivalence of the time-domain representation of a waveform on the top and the frequency-domain representation on the bottom.

In real life, a representation of a sound in its frequency domain would encompass the entire audio spectrum from 20 Hz to 20,000 Hz, and constantly change over the course of time.

Filter Basics

The frequency-domain representation allows us to think of sound as a collection of sine waves of various frequencies, and that's often useful in understanding audio processing.

A very common type of audio processing involves amplifying or attenuating certain ranges of frequencies in the audio spectrum, thus altering the harmonic composition of the sound. This is a tool known as a filter. In analog signal processing, filters are circuits; in digital signal processing, they're algorithms.

The most common filter types are called low-pass, high-pass and band-pass; the terms refer to the frequencies the filters let through. The low-pass filter emphasizes lower frequencies by attenuating higher frequencies. Similarly, the high-pass filter attenuates lower frequencies. Both the low-pass and high-pass filters are defined by a particular cutoff frequency that indicates where the attenuation begins. The band-pass filter doesn't have a cutoff frequency, but a center frequency serves a similar purpose. Frequencies outside of a range around that center frequency are attenuated.

Most filters can't simply block all sine waves above or below a particular frequency. Instead, a sine wave with a particular frequency is attenuated based on its distance from the cutoff or center frequency with a roll-off effect. The slope of this roll-off is governed by a property of the filter known as *Q*, which stands for *quality*. A filter with a higher *Q* has a steeper roll-off.

Code download available at archive.msdn.microsoft.com/mag201305DXF.

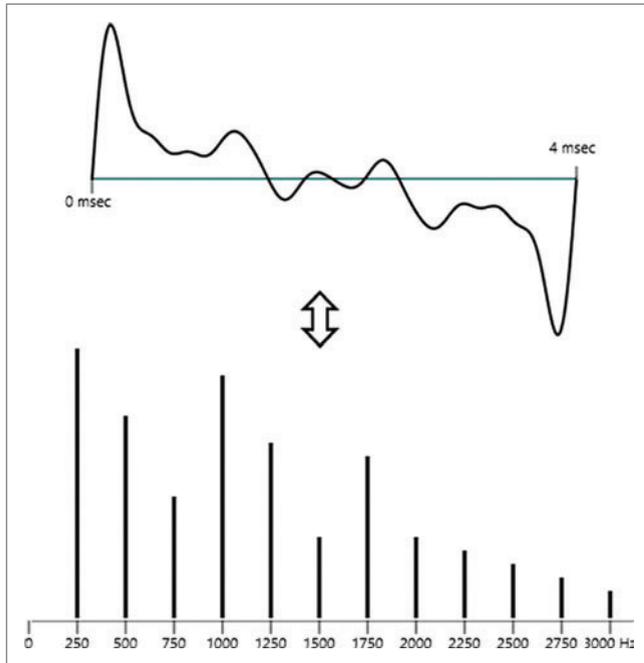


Figure 2 A Waveform in Time Domain (Top) and Frequency Domain (Bottom)

The Q factor is easiest to interpret with a band-pass filter. Figure 3 shows the effect of a band-pass filter applied to a range of frequencies. The center frequency is marked at f_0 , and two other frequencies are marked f_1 and f_2 where the band-pass filter attenuates the amplitude to 70.7 percent of the f_0 amplitude.

Why 70.7 percent? The power of a waveform is calculated as the square of the waveform's amplitude, and f_1 and f_2 indicate the frequencies where the waveform has been attenuated to half its original power. Because power is amplitude squared, the amplitude at those points is the square root of $1/2$, or 0.707.

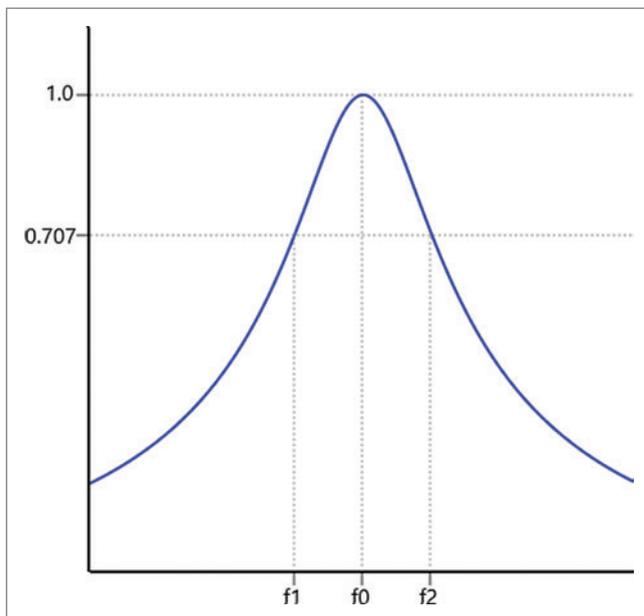


Figure 3 Bandwidth in a Band-Pass Filter

Q is calculated by dividing the center frequency by the difference between the two half-power frequencies:

$$Q = f_0 / (f_2 - f_1)$$

However, the difference between f_2 and f_0 is not the same as the difference between f_0 and f_1 . Instead, the ratios are the same: f_2 / f_0 equals f_0 / f_1 . If f_2 is double f_1 , that's an octave, and it's fairly easy to calculate that Q equals the square root of 2, or approximately 1.414.

The ratio between f_2 and f_1 is known as the filter's bandwidth, and is often specified in octaves. For a bandwidth B in octaves, you can calculate Q like so:

$$Q = \frac{\sqrt{2^B}}{2^B - 1}$$

As the bandwidth decreases, Q increases and the roll-off is steeper.

I said that f_2 and f_1 are the frequencies that the filter attenuates to half the power of f_0 . Half the power is also known as -3 decibels. The decibel is a logarithmic scale roughly approximating the human perception of loudness. The difference in decibels between two power levels P1 and P0 is:

$$db = 10 \cdot \log_{10}(P1/P0)$$

If P1 is half P0, the base-10 logarithm of 0.5 is -0.301, and 10 times that is approximately -3. When dealing with amplitudes, decibels are calculated as:

$$db = 20 \cdot \log_{10}(A1/A0)$$

The base-10 logarithm of 0.707 is -0.15, and 20 times that is also -3.

Every doubling of the amplitude corresponds to an increase of 6 decibels, which is why the 16-bit sampling rate of audio CDs is sometimes said to have a dynamic range of 96 decibels.

Applying a Filter

If you're using XAudio2 in a Windows 8 program to generate sounds or modify the sounds of existing music files, applying a filter to those sounds is as simple as initializing three fields of an XAUDIO2_FILTER_PARAMETERS structure and calling a method named SetFilterParameters.

As you've seen in recent installments of this column, a program creates one or more instances of IXAudio2SourceVoice to define the waveforms themselves, and a single instance of IXAudio2-MasteringVoice to effectively combine all the source voices into a single audio stream. Later in this article you'll also see how to create instances of IXAudio2SubmixVoice to control the processing and mixing of sounds on their way to the mastering voice. Both source voices and submix voices support the SetFilterParameters method, but only if the voices are created with the XAUDIO2_VOICE_USEFILTER flag.

A call to SetFilterParameters requires a pointer to an XAUDIO2_FILTER_PARAMETERS structure, which has three fields:

Type: set to one of the members of XAUDIO2_FILTER_TYPE enumeration, which includes members for low-pass, high-pass and band-pass filters, as well as a notch (or band-reject) filter, and single-pole low-pass and high-pass filters, which (as you'll see shortly) are simpler filters.

Frequency: set to $2 \cdot \sin(\pi \cdot f_0 / f_s)$ where f_0 is the cutoff frequency and f_s is the sampling frequency, where f_0 is not greater than $1/6 f_s$, which means that the value set to the field is not greater than 1.

OneOverQ: 1 divided by the desired Q factor, greater than zero and no greater than 1.5. Thus, Q cannot be less than 2/3, which corresponds to a bandwidth of 2 octaves.

I haven't yet shown you graphs, similar to **Figure 3**, that illustrate how the low-pass and high-pass filters attenuate frequencies. Sometimes such graphs simply show a roll-off effect and thus can be dangerously deceptive if the actual filter doesn't quite work like that. Such is the case with XAudio2 filters. For the low-pass, high-pass, band-pass, and notch filters, XAudio2 implements a type of digital filter known as a biquad, which involves a fairly simple algorithm but does not create a simple roll-off effect for low-pass and high-pass filters. (If you're interested in the algorithm, follow the links in the Wikipedia article on "Digital biquad filter" at bit.ly/Yoeeq1.)

Biquad filters tend to resonate at the center frequency of a band-pass filter, and near the cutoff frequencies of the low-pass and high-pass filters. This means that the filter not only attenuates some frequencies, but amplifies others. To use these filters intelligently, you must be aware of this effect. Fortunately, this amplification is fairly easy to predict. For the band-pass filter, the amplitude of a sine wave at the center frequency is increased by a factor equal to Q. For the low-pass and high-pass filters, the maximum amplification near the cutoff frequency is equal to Q for higher values of Q, but somewhat greater than Q for lower values.

Figure 4 shows the effects of all the XAudio2 filter types set for a frequency of 261.6 Hz (middle C) and a Q of 1.414. The horizontal axis is logarithmic with a range of 3 octaves above and below middle C. The vertical axis shows the resultant amplitude for sine curves at those frequencies. The horizontal black line at an amplitude of 1 is for no filter. All the other lines are identified with different colors.

For example, the low-pass filter not only lets through frequencies below the cutoff frequency, but amplifies them, and this amplification increases as you get closer to the cutoff frequency. The high-pass filter has the opposite effect.

Figure 5 is similar to **Figure 4** but for a Q of 4.318, which is associated with a bandwidth of 1/3 octave. Notice that the vertical axis is different to accommodate the increased amplification.

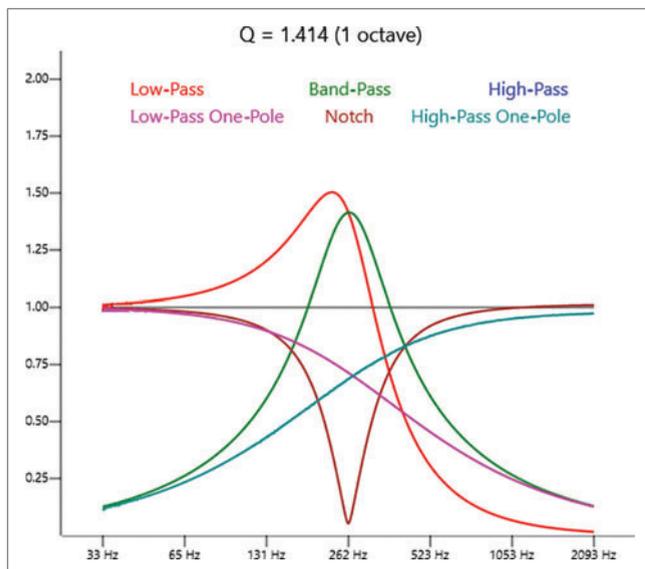


Figure 4 The Effect of Filters for a Q of 1.414

If you want to use a simple low-pass or high-pass filter that won't amplify at all, stick to the one-pole filters. These are very simple filters governed simply by a cutoff frequency and they don't use the Q setting. They function much like the simple bass and treble controls on a car stereo. But if you want to use the more sophisticated filters, your program must compensate for any amplification by the filter.

If you'd rather implement your own filters, you can do that as well by creating an XAudio2 Audio Processing Object (XAPO), which is a class that gets access to an audio stream and can implement effects.

Watching the Volume

To allow me (and you) to experiment with filters, I created a Windows 8 project named AudioFilterDemo that's included in the downloadable code for this article. **Figure 6** shows it running.

The three oscillators toward the top are all independently controllable, with a frequency range encompassing 3 octaves on either side of middle C. The slider is a logarithmic scale but adjustable to 10 divisions between notes, which is an increment known as 10 cents.

The filter has a frequency slider as well as a slider for Q. All the frequency sliders have tooltips that identify the note and its frequency. **Figure 7** shows the method that sets the filter on the three waveform source voices whenever there's a change in the controls.

The bottom panel is a volume meter scaled in decibels. This allows you to see the resultant volume for a particular waveform and filter settings. The program makes no adjustments to volume other than through user settings. If this meter goes into the red, it means that the program is generating a sound that's too loud, and that waveform is being clipped before going into the sound system on your computer.

The volume meter is based on a predefined effects class. **Figure 8** shows the code I used to create an instance of that effect. The program then sets a timer and calls `GetEffectParameters` to obtain the peak levels of the output sound since the last time `GetEffectParameters` was called.

One interesting exercise in this program is to play either a square wave or sawtooth wave through a band-pass filter with a Q of at

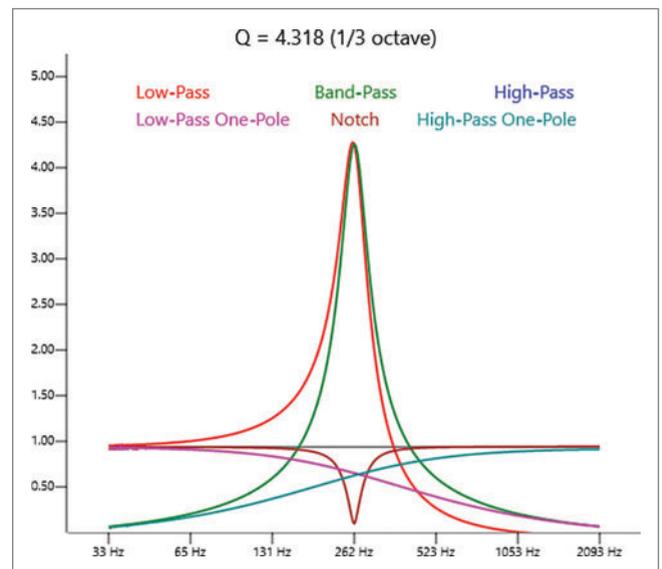


Figure 5 The Effect of Filters for a Q of 4.318

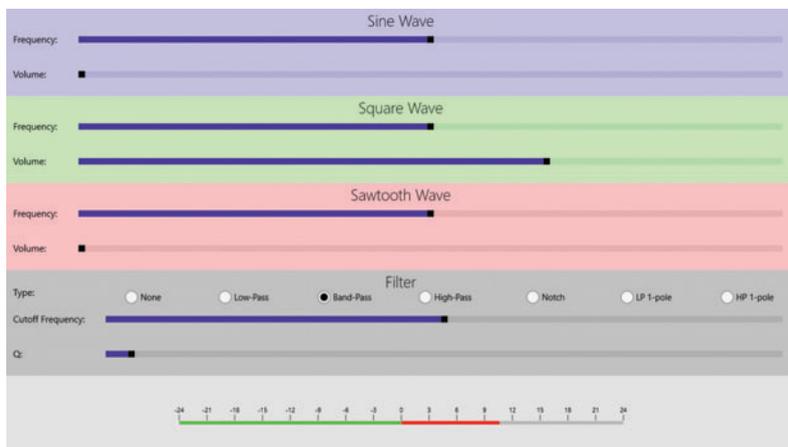


Figure 6 The AudioFilterDemo Program

least 4 or so. As you change the filter frequency, you can hear the individual overtones of the waveform. A square wave has only odd harmonics, but a sawtooth wave has both odd and even harmonics.

The Graphic Equalizer

Time was, every well-equipped home audio setup included a graphic equalizer containing a row of vertical slide potentiometers controlling a bank of band-pass filters. In a graphic equalizer, each band-pass filter covers two-thirds or one-third of an octave in the total audio spectrum. Among professional sound engineers, graphic equalizers are used to adjust for the acoustic response of a room by boosting or cutting various frequencies. Home users often arrange the sliders in a “smile” pattern as imitated in **Figure 9**, boosting both the low and high ends and leaving the middle range softer so as to not inordinately interfere with conversation.

Figure 7 AudioFilterDemo Setting XAudio2 Filter Parameters

```
void MainPage::SetFilterParameters()
{
    if (pSawtoothOscillator != nullptr)
    {
        XAUDIO2_FILTER_PARAMETERS filterParameters;

        if (currentFilterType != -1)
        {
            double cutoffFrequency =
                440 * pow(2, (filterFrequencySlider->Value - 69) / 12);
            filterParameters.Type = XAUDIO2_FILTER_TYPE(currentFilterType);
            filterParameters.Frequency =
                float(2 * sin(3.14 * cutoffFrequency / 44100));
            filterParameters.OneOverQ = float(1 / filterQSlider->Value);
        }
        else
        {
            // Documentation:
            // "acoustically equivalent to the filter being fully bypassed"
            filterParameters.Type = LowPassFilter;
            filterParameters.Frequency = 1.0f;
            filterParameters.OneOverQ = 1.0f;
        }
    }

    pSawtoothOscillator->GetVoice()->SetFilterParameters(
        &filterParameters, XAUDIO2_COMMIT_ALL);
    pSquareWaveOscillator->GetVoice()->SetFilterParameters(
        &filterParameters, XAUDIO2_COMMIT_ALL);
    pSineWaveOscillator->GetVoice()->SetFilterParameters(
        &filterParameters, XAUDIO2_COMMIT_ALL);
}
}
```

The GraphicEqualizer program allows you to load an MP3 or WMA file from your Windows 8 Music Library and play it through a 1/3-octave graphic equalizer. The program contains 26 vertical sliders, each of which is associated with a band-pass filter. As you can see, each slider is labeled with the center frequency for that band. In theory, each center frequency should be the cube root of 2 (or about 1.26) higher than the previous filter, but lots of rounding is employed to keep the numbers sensible. Based on a photograph of a 1/3 graphic equalizer I found on Wikipedia, I labeled the 26 sliders starting with 20 Hz, 25, 31.5, 40 and up through 6.3 KHz, stopping short of the 7,350 Hz limit for a 44,100 Hz sampling rate.

Most graphic equalizers have a separate band of potentiometers for left and right channels, but I decided to forgo that amenity.

You’ve seen how a single filter can be applied to a particular IXAudio2SourceVoice instance, but the GraphicEqualizer program needs to apply 26 filters to a source voice. This is accomplished by creating 26 instances of IXAudio2SubmixVoice corresponding to these filters (plus a couple more), and creating what’s called in XAudio2 an “audio processing graph,” as shown in **Figure 10**. Each box is an instance of one of the three interfaces that derive from IXAudio2Voice, and the box identifies the variable name used in the GraphicEqualizer program.

An IXAudio2SubmixVoice instance can’t generate sound on its own. That privilege is reserved for source voices. But it can get input from a source voice (or another submix voice); apply a volume, filter or effect; and pass the result on to one or more submix voices, or to the mastering voice.

At the top of **Figure 10** is the source voice that generates the music from a music file. At the bottom is the mastering voice that sends the result out to the computer’s sound hardware. In between are all the submix voices.

It’s a push model: Whenever you create a source voice or submix voice, you can indicate the destination voice (or voices) you want to receive the output of that voice. Later on, you can change the destination of that output with a call to SetOutputVoices. If you specify NULL in either case, the output goes to the mastering voice.

Figure 8 Creating a Volume Meter Effect

```
// Create volume meter effect
IUnknown * pVolumeMeterAPO;
XAudio2CreateVolumeMeter(&pVolumeMeterAPO);

// Reference the effect with two structures
XAUDIO2_EFFECT_DESCRIPTOR effectDescriptor;
effectDescriptor.pEffect = pVolumeMeterAPO;
effectDescriptor.InitialState = true;
effectDescriptor.OutputChannels = 2;

XAUDIO2_EFFECT_CHAIN effectChain;
effectChain.EffectCount = 1;
effectChain.pEffectDescriptors = &effectDescriptor;

// Set the effect on the mastering voice
pMasteringVoice->SetEffectChain(&effectChain);

// Release the local reference to the effect
pVolumeMeterAPO->Release();
```

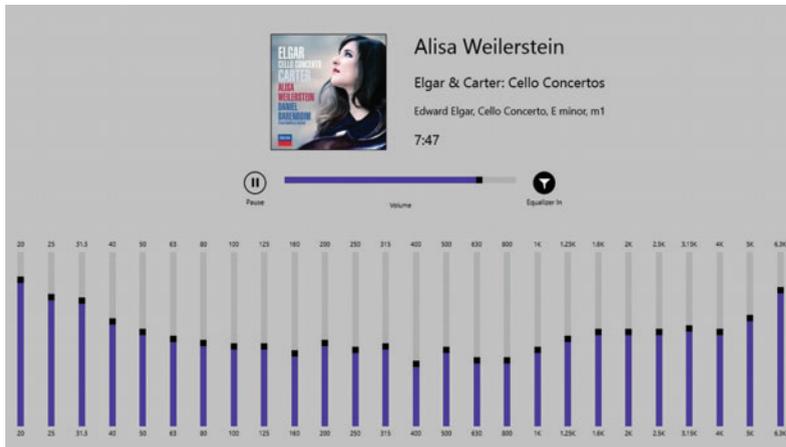


Figure 9 The Graphic Equalizer Program

You indicate where you want the output of a voice to go with a pointer to an `XAUDIO2_VOICE_SENDS` structure, which contains two fields:

- `SendCount` of type unsigned integer
- `pSends`, which is a pointer to zero or more `XAUDIO2_VOICE_DESCRIPTOR` structures

The `SendCount` indicates the number of `XAUDIO2_VOICE_DESCRIPTOR` structures pointed to by `pSends`. It can be zero to indicate that the voice doesn't go anywhere. The `XAUDIO2_VOICE_DESCRIPTOR` structure also has two fields:

- `Flags`, which can be 0 or `XAUDIO2_SEND_USEFILTER`
- `pOutputVoice`, of type `IXAudio2Voice`

The two `IXAudio2SubmixVoice` instances that feed into the bank of 26 submix voices, and the one that consolidates the output from those 26 voices, aren't strictly needed to build the graphic equalizer, but they simplify the structure of the program. Whenever the program creates a new source voice—which happens whenever the user loads in a new music file—it just needs to direct the source voice's output to the `pSourceVoiceOutput` instance.

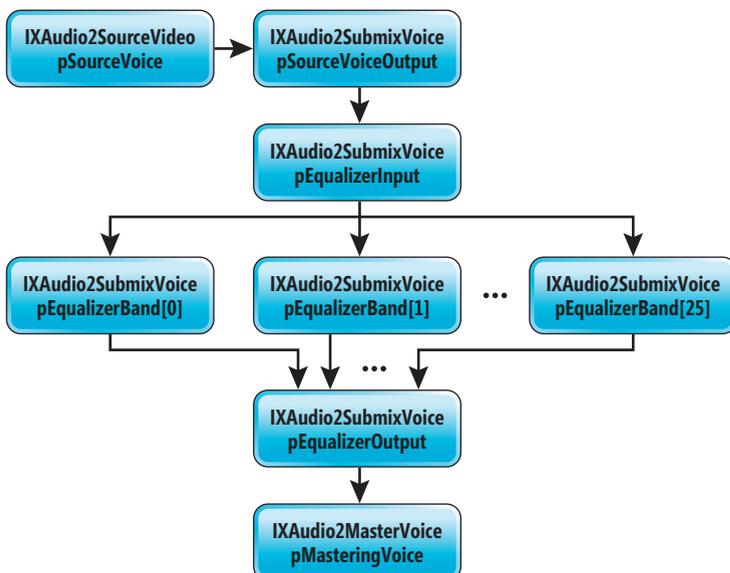


Figure 10 The Audio Processing Graph Used in the Graphic Equalizer Program

The program also has a `CheckBox` button to bypass the equalizer. To disconnect the equalizer from the audio processing graph, all that's necessary is to call `SetOutputVoices` on the `pSourceVoiceOutput` with a `NULL` pointer indicating it should go to the mastering voice. Restoring the equalizer involves a few lines of code to restore output from `pSourceVoiceOutput` to `pEqualizerInput`.

There are a couple of ways to define the filters that comprise a graphic equalizer. One approach is for each slider to change the `Q` of that filter—in effect making the filter more restrictive as you increase the slider. But I decided to keep the `Q` factor of each filter constant at a 1/3-octave bandwidth, or 4.318, and use the slider to vary the volume of that submix voice.

Graphic equalizers usually allow switching the bank of sliders between a ± 6 dB range and a ± 12 dB range, but I decided on a ± 24 dB range for more extreme effects.

When an equalizer slider is in its center position, the corresponding submix voice has a default volume of 1. Normally that would mean that a sound just passes through the submix voice unaltered. However, applying a filter with a `Q` of 4.318 in a submix voice causes the amplitude to increase by a factor of 4.318 at the center frequency. To compensate for that, the program sets the volume of the submix voice `pEqualizerOutput` to 1 divided by `Q`.

With all the sliders set in their center positions, clicking the `CheckBox` to switch the equalizer in and out of the audio graph causes no change in volume. The sound does change a little—undoubtedly resulting from the way the various band-pass filters overlap—but the overall volume does not.

The equalizer sliders have their `Minimum` properties set to -24 and `Maximum` set to 24, corresponding to the gain in decibels. When the slider value changes, the volume for the corresponding submix voice is set in the `ValueChanged` event handler, like so:

```
Slider^ slider = dynamic_cast<Slider^>(sender);
int bandIndex = (int)slider->Tag;
float amplitude = float(pow(10, (args->NewValue / 20)));
pEqualizerBands[bandIndex]->SetVolume(amplitude, 0);
```

That amplitude calculation is an inverse of the decibel calculation shown earlier. The resultant amplitude ranges from about 0.06 (at -24 dB) to about 16 (at 24 dB). If you keep in mind that each change of 6 dB is a halving or doubling of the center amplitude of 1, these ranges make sense. But if you crank up all the sliders to their maximum settings, the overall amplitude increases by a factor of 16, and the result is likely to be clipped and distorted.

In other words, the program implicitly assumes that the user maintains a balanced approach to life, and will reduce some sliders while increasing others. ■

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.

THANKS to the following technical expert for reviewing this article: Richard Fricks (Microsoft)



MAY 14-16, 2013
WASHINGTON, DC
WALTER E. WASHINGTON CONVENTION CENTER

FREE EXPO

Hands-on demos, free education, networking, and more!



KEYNOTES



Government Tech Talks

Valuable technology implementation —3 strategies in 15 minutes!



FIRST @ FOSE

See industry giants' latest products and services.

Solutions Sessions

Discover the latest innovations from government technology visionaries.



THE FREE FOSE EXPERIENCE



SEE IT. HEAR IT. TRY IT. KNOW IT.
Experience NEW Solutions at FOSE.



FOSETV

The latest news right in the exhibit hall.

RECORDS & INFORMATION MANAGEMENT SESSIONS

Comply & Implement Take advantage of others' successful programs.



App Arcade

Experience the latest apps for government.



FREE TRAINING & EDUCATION FOR GOVERNMENT IN YOUR BACKYARD!

Your FREE Expo pass includes:

- Solutions Sessions
- Government Tech Talks
- App Arcade—NEW
- FIRST @ FOSE—NEW
- FOSETV—NEW
- Access to 100+ exhibitors
- 4+ Hours of Daily Networking
- ... and much more!

REDEEM YOUR FREE EXPO PASS TODAY

USE PRIORITY CODE: FOSEAD3

FOSE.com

Expo is free for government; \$50 for industry suppliers.

HEAR DIRECTLY FROM THESE LEADERS



General Stan McChrystal
Former Commander of U.S. & International Forces in Afghanistan



Joe Theismann
Former Washington Redskins Quarterback



Jan R. Frye
Dept. Asst. Secretary, Office of Acquisition & Logistics Dept. of Veterans Affairs



Steven VanRoekel
U.S. Chief Information Officer (Invited)



Senator Tom Carper
Homeland Security & Governmental Affairs Committee Chairman



1105 GOVERNMENT EVENTS

PRODUCED BY

PLATINUM PARTNER



GOLD SPONSORS



SILVER SPONSORS



TECHNOLOGY SPONSORS





Do As I Say, Not As I Do

I'd really hoped that I wouldn't have to write this particular column. Last November ("Here We Go Again," msdn.microsoft.com/magazine/jj721604) I expressed hope that the Microsoft Windows 8 UI guidelines would prevent geeks from throwing distracting random animations at users simply because they think it's cool. Unfortunately, I was wrong, and the biggest offender is another part of Microsoft.

One of the biggest changes to the Windows 8 UI is the live tile. This allows a program to show its current state in its home screen tile even if the program isn't currently running. When used properly, this can be a great idea. For example, a messenger program could show the number of messages awaiting the user's attention, or an app for rabid boaters could show the current state of the ocean tide.

Benjamin Franklin wrote:
"Actions speak louder
than words." And Mark Twain
added, "But not nearly as often."
In the case of live tile updates,
Twain was wrong.

Unfortunately, when I installed Windows 8 on my ThinkPad, I saw no fewer than six live tiles clamoring for my attention: News, Finance, Sports, People, Bing and Travel. Between them they change about once per second, and each change features an animation specifically directed to grab my eye. The content itself doesn't change that often—for example, the News tile rotates the same three headlines for an hour or so. It's like a hyperactive golden retriever slobbering in my face: "You didn't want me 15 seconds ago? How about now? Pick me, pretty, pretty please!" Microsoft's own design guidelines for live tile updates (bit.ly/10QJNvQ) say not to do this. Here's the relevant portion, my emphasis added:

- For personalized content, such as message counts or whose turn it is in a game, we recommend that you update the

tile as the information becomes available, particularly if the user would notice that the tile content was lagging, incorrect, or missing.

- For nonpersonalized content, such as weather updates, we recommend that the tile be updated no more than once every 30 minutes. This allows your tile to feel up-to-date without overwhelming your user.

So, what do you think app developers are going to do—dig down deep into the specs and do what Microsoft's words say to do, or imitate what they see Microsoft's programs are doing in front of their noses all day, every day? Rudolf Steiner, founder of the Waldorf School movement, says, "There are only three ways to educate a child: The first is by example. The second is by example. And the third is by example." Benjamin Franklin wrote: "Actions speak louder than words." And Mark Twain added, "But not nearly as often." In the case of live tile updates, Twain was wrong.

I can turn off updates for a live tile, but then I get no content at all. There's no setting for a polite, respectful update once in a while; it's the fire hose or nothing. A better design would be to show the latest content when I return to the homepage, as a Web browser does. Better still would be for the news program to automatically track my preferences and select new articles for me based on the ones I click. That would've been fabulous. But, no—we've got animations in the toolkit, we have to employ them, whether they help our user or not. "Look and feel," the designers say. "Isn't it cool?" No, it isn't. It's distracting. It's counterproductive. It's juvenile. Follow your own guidelines, the ones written by grownups. The world will be a better place.

Will Microsoft now modify its Windows 8 apps to conform to its own rules? Or will Microsoft fall back on the argument-ending pronouncement used by every parent: "Because I'm the daddy, that's why"? ■

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.



100s of UI controls for all .NET platforms including grids, charts, reports & schedulers
 New Tile controls for WinForms, WPF, Silverlight, WinRT & Windows Phone
 Now includes MVC 4 Scaffolding & new MVC 4 project templates (C# & VB)
 Data Visualization controls for Windows Store apps

ComponentOne® Studio® Enterprise

ComponentOne®
 a division of GrapeCity®

Download your free trial @
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.

Introducing the latest e-book in the

Syncfusion Succinctly Series



14 titles and growing | Ad-free | 100 pages | Kindle and PDF formats

Download your free copy today to build a foundation
for your future iOS development with *iOS Succinctly*.



syncfusion.com/ios

Technology Resource Portal

