DevExpress

When the web means business

Performance
Elegance
Productivity

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

DevExpress

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

u dxp dxt | asp win wpf sl | cr vcl

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

THE MICROSOFT JOURNAL FOR DEVELOPERS

JUNE 2013 VOL 28 NO 6

msdn magazine

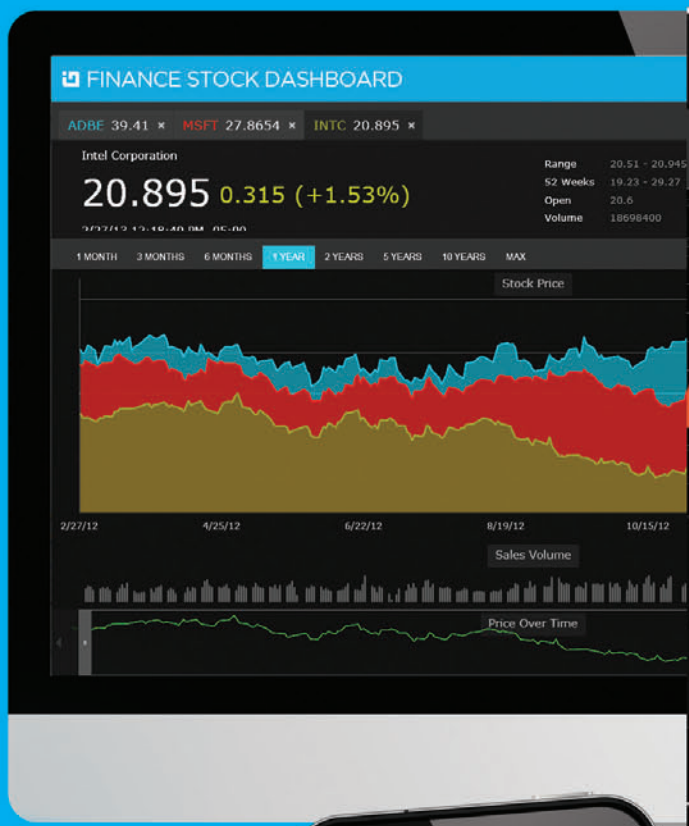
Share Windows Store & Windows Phone Code....28

COLUMNS

Microsoft

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

Dr.Dobb's JOLT 2013 BEST PROGRAMMER LIBRARIES

Cross-Device

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
DESIGN / DEVELOP / EXPERIENCE

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

Copyright 1996-2013 Infragistics, Inc. All rights reserved. Infragistics and NetAdvantage are registered trademarks of Infragistics, Inc. The Infragistics logo is a trademark of Infragistics, Inc.

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

JUNE 2013 VOLUME 28 NUMBER 6

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/dlabianca@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 INC

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

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

Jeffrey S. Klein Chairman of the Board

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

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

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

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

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

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

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

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

Microsoft

BPA WORLDWIDE

Printed in the USA

LEADTOOLS V18
THE WORLD LEADER IN IMAGING SDKs ANYWHERE

C++ C# JavaScript
VB, Objective-C, Java

Stop By TechEd Booth #328 To Learn More

The world's leading Imaging SDK
NOW RUNS ANYWHERE

OCR
BARCODE
COGNITION &
DOCUMENT PREPROCESSING
ANNOTATIONS
DICOM & PACS
MEDICAL WORKSTATI
MPEG-2 TRANSPORT STREAM
MULTIMED
COMPREHENSIVE IMAGIN

& FOR A FREE COUPON TO Café Du Monde®

MICHAEL DESMOND

Unbridled Expectations

Comedian Louis C.K., well known for his deeply cynical and at times wildly off-color rants, has an extended riff on how crazy it is for people to be disappointed with wonderful things like air travel and cell phones. I ran across an interview Louis C.K. did on the Conan O'Brien show describing his observations, and realized that what he's talking about poses a real challenge for software developers.

> Today, a smartphone UI will pause for a moment, and the immediate reaction is to get angry at a thing that, just a decade or so ago, would've seemed miraculous.

"Everything is amazing and *nobody* is happy," he told O'Brien, describing how he grew up with rotary phones that made you dread calling people with zeroes in their number, because it took so long to dial. Today, a smartphone UI will pause for a moment, and the immediate reaction is to get angry at a thing that, just a decade or so ago, would've seemed miraculous.

"Flying is the worst one," Louis C.K. continued, scoffing at people who describe airline flights as horror stories, as if sitting on the tarmac for 40 minutes before takeoff is a calamity. "Oh really? What happened next? Did you fly through the air incredibly like a bird? Did you partake in the miracle of human flight, you non-contributing zero? You're flying! It's amazing."

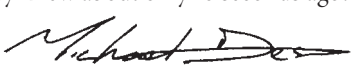
Louis C.K. then described a flight he was on with in-flight Wi-Fi service and Internet access. When the Wi-Fi suddenly went out, a man behind him started swearing in frustration. "Like, how quickly does the world owe him something that he knew about only 10 seconds ago?" he asked.

I was thinking about this because I've been frustrated with the Netflix app on my old Samsung Galaxy Tab 10.1 Android tablet. The app had been rock-solid for months, particularly after I learned to take the time to use the Back button to invoke a dialog box that completely shuts down the app. About a week ago the behavior changed, and the Back button now just ushers the app off to a suspended state sans dialog box. But launching Netflix from a suspended state causes the app to hang on my tablet. Every. Single. Time.

This makes me angry. And given Louis C.K.'s observations, perhaps unreasonably so. I mean, work an extra 20 seconds to kill the Netflix app in the background and then relaunch it, so I can get at the next episode of "Arrested Development"? Bah!

But this reflects a real challenge for modern application development. When code on the Web or in the cloud or in a Windows Store app changes, it often changes the UX—and in ways your users might not be expecting. It used to be that upgrading an application or OS was a big deal that implied the somber shuffling of floppy discs or CDs, or at least a lengthy file download. There was a ceremony to the process, and it helped ensure that end users were fully vested and informed.

Today, not so much. Windows Store app updates are a quick click away, and Web apps are prone to sudden change or even retirement (I'm looking at you, Google Reader). But when a putative upgrade removes a beloved feature or familiar control, or creates a break in functionality, suddenly you've got Louis C.K.'s irate consumer problem, in all its self-centered glory. So, how quickly do you owe your users something that they knew about only 10 seconds ago? It's a worthwhile question.



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.

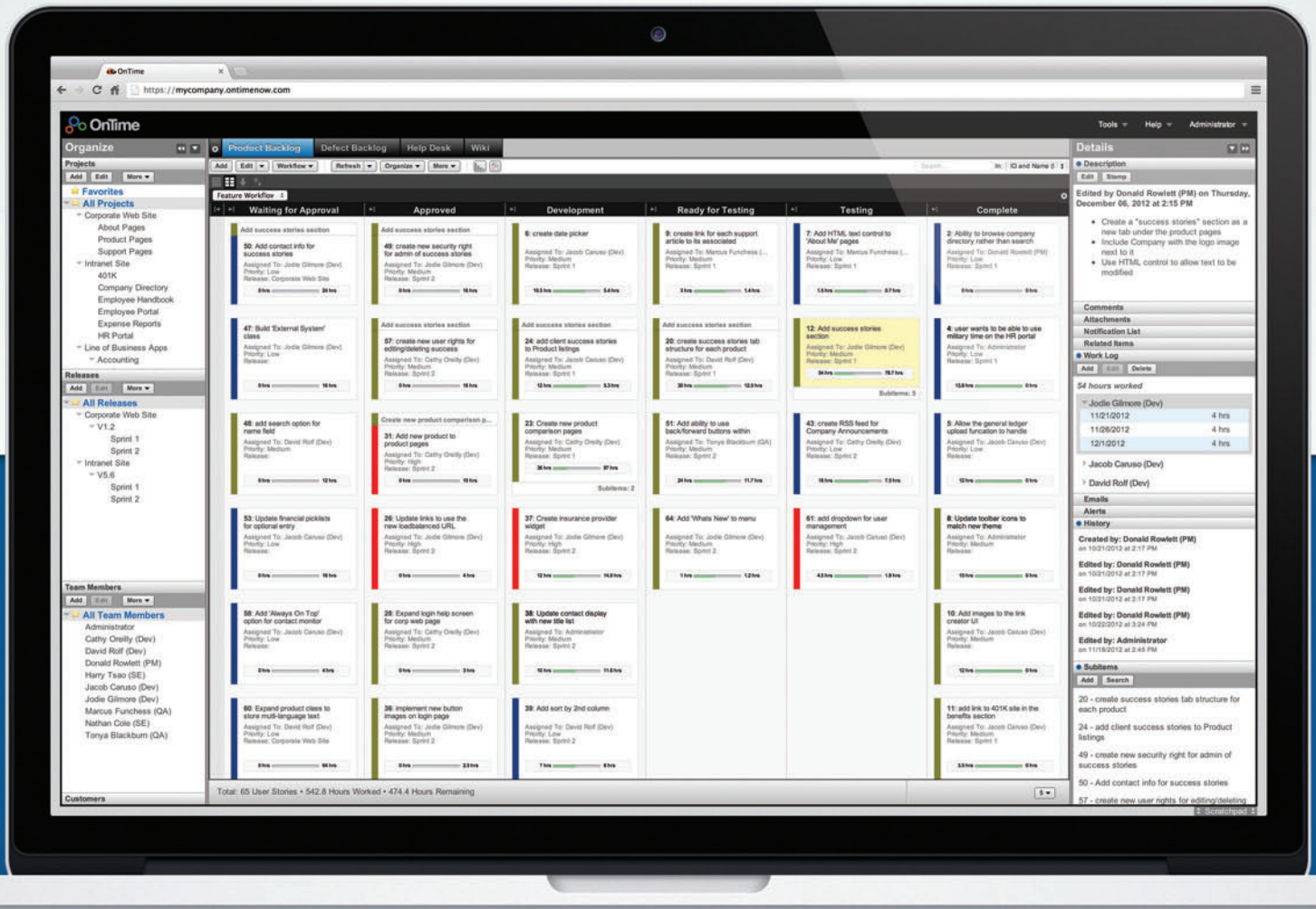
Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, you are not permitted to reproduce, store, or introduce into a retrieval system *MSDN Magazine* or any part of *MSDN Magazine*. If you have purchased or have otherwise properly 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

axosoft

800.653.0024 • www.ontimenow.com • www.axosoft.com • @axosoft

KENNY KERR

A Modern C++ Library for DirectX Programming

I've written a lot of DirectX code and I've written about DirectX extensively. I've even produced online training courses on DirectX. It really isn't as hard as some developers make it out to be. There's definitely a learning curve, but once you get over that, it's not hard to understand how and why DirectX works the way it does. Still, I'll admit that the DirectX family of APIs could be easier to use.

A few nights ago, I decided to remedy this. I stayed up all night and wrote a little header file. A few nights later, it approached 5,000 lines of code. My goal was to provide something that you can use to build apps more easily with Direct2D and to challenge all of the "C++ is hard" or "DirectX is hard" arguments that are so prevalent today. I didn't want to produce yet another heavy wrapper around DirectX. Instead, I decided to leverage C++11 to produce a simpler API for DirectX without imposing any space and time overheard to the core DirectX API. You can find this library I developed at dx.codeplex.com.

> I stayed up all night and wrote a little header file. A few nights later, it approached 5,000 lines of code.

The library itself consists entirely of a single header file called dx.h—the rest of the source code on CodePlex provides examples of its use.

In this column, I'll show you how you can use the library to perform various common DirectX-related activities more easily. I'll also describe the design of this library to give you an idea of how C++11 can help to make classic COM APIs more palatable without resorting to high-impact wrappers such as the Microsoft .NET Framework.

Obviously, the focus is on Direct2D. It remains the simplest and most effective way of leveraging DirectX for the broadest class of applications and games. Many developers seem to fall into two opposing camps. There are the hardcore DirectX developers who have cut their teeth on various versions of the DirectX API. They're hardened by years of DirectX evolution and are happy to be in an exclusive club with a high bar of entry, a club that very few developers may join. In the other camp are those developers who heard the message that DirectX is hard and don't want anything to do with it. They also tend to reject C++ as a matter of course.

I don't belong to either of these camps. I believe C++ and DirectX don't have to be hard. In last month's column (msdn.microsoft.com/magazine/dn198239), I introduced Direct2D 1.1 and the prerequisite Direct3D and DirectX Graphics Infrastructure (DXGI) code to create a device and manage a swap chain. The code for creating a Direct3D device with the D3D11CreateDevice function suitable for GPU or CPU rendering comes to roughly 35 lines of code. However, with the help of my little header file, it amounts to this:

```
auto device = CreateDevice();
```

The CreateDevice function returns a Device1 object. All of the Direct3D definitions are in the Direct3D namespace, so I could be more explicit and write it as follows:

```
Direct3D::Device1 device = Direct3D::CreateDevice();
```

The Device1 object is simply a wrapper around an ID3D11Device1 COM interface pointer, the Direct3D device interface introduced with the DirectX 11.1 release. The Device1 class derives from the Device class, which in turn is a wrapper around the original ID3D11Device interface. It represents one reference and adds no additional overhead compared to just holding on to the interface pointer itself. Note that Device1 and its parent class, Device, are regular C++ classes rather than interfaces. You could think of them as smart pointers, but that would be overly simplistic. Sure, they handle reference counting and provide the "->" operator to directly call the method of your choosing, but they really begin to shine when you start using the many nonvirtual methods provided by the dx.h library.

Here's an example. Often you might need the Direct3D device's DXGI interface to pass to some other method or function. You could do it the hard way:

```
auto device = Direct3D::CreateDevice();
wrl::ComPtr<IDXGIDevice2> dxdevice;
HR(device->QueryInterface(dxdevice.GetAddressOf()));
```

That certainly works, but now you also have to deal with the DXGI device interface directly. You also need to remember that the IDXGIDevice2 interface is the DirectX 11.1 version of the DXGI device interface. Instead, you can simply call the AsDxgi method:

```
auto device = Direct3D::CreateDevice();
auto dxdevice = device.AsDxgi();
```

The resulting Device2 object, defined in the Dxgi namespace this time, wraps the IDXGIDevice2 COM interface pointer, providing

Code download available at dx.codeplex.com.

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


ASPOSE
Your File Format Experts

Follow us on Facebook & Twitter

Get your FREE evaluation copy at www.aspose.com

its own set of nonvirtual methods. As a further example, you might want to use the DirectX "object model" to make your way to the DXGI factory:

```
auto device   = Direct3D::CreateDevice();
auto dxdevice = device.AsDxgi();
auto adapter  = dxdevice.GetAdapter();
auto factory  = adapter.GetParent();
```

This is, of course, such a common pattern that the Direct3D Device class provides the GetDxgiFactory method as a convenient shortcut:

```
auto d3device = Direct3D::CreateDevice();
auto dxfactory = d3device.GetDxgiFactory();
```

Many of the best libraries lead you to a working solution quickly and easily.

So, apart from a few convenience methods and functions such as GetDxgiFactory, the nonvirtual methods map one-to-one to the underlying DirectX interface methods and functions. That might not seem like much, but a number of techniques are combined to produce a far more convenient and productive programming model for DirectX. The first is the use of scoped enumerations. The DirectX family of APIs defines a dizzying array of constants, many of which are traditional enums, flags or constants. They aren't strongly typed, they're hard to find, and they don't play well with Visual Studio IntelliSense. Ignoring the factory options for a minute, here's what you need to create a Direct2D factory:

```
wrl::ComPtr<ID2D1Factory1> factory;

HR(D2D1CreateFactory(D2D1_FACTORY_TYPE_SINGLE_THREADED,
                     factory.GetAddressOf()));
```

The D2D1CreateFactory function's first parameter is an enum, but because it's not a scoped enum, it's hard to discover with Visual Studio IntelliSense. These old-school enums provide a bit of type safety, but not a whole lot. At best, you'll be treated to an E_INVALIDARG result code at run time. I don't know about you, but I'd rather catch such errors at compile time, or better yet, avoid them altogether:

```
auto factory = CreateFactory(FactoryType::MultiThreaded);
```

Again, I don't need to spend time digging around to find out what the latest version of the Direct2D factory interface is called.

Figure 1 **Creating a Linear Gradient Brush the Hard Way**

```
D2D1_GRADIENT_STOP stops[] =
{
  { 0.0f, COLOR_WHITE },
  { 1.0f, COLOR_BLUE },
};

wrl::ComPtr<ID2D1GradientStopCollection> collection;

HR(target->CreateGradientStopCollection(stops,
                                        _countof(stops),
                                        collection.GetAddressOf()));

wrl::ComPtr<ID2D1LinearGradientBrush> brush;

HR(target->CreateLinearGradientBrush(D2D1_LINEAR_GRADIENT_BRUSH_PROPERTIES(),
                                     collection.Get(),
                                     brush.GetAddressOf()));
```

Certainly, the biggest benefit here is productivity. Of course, the DirectX API is about more than creating and calling COM interface methods. Many plain old data structures are used to bundle up various properties and parameters. The description of a swap chain is a good example. With all of its confusing members, I can never quite remember how this structure should be prepared, not to mention the platform specifics. Here, again, the library comes in handy by providing a replacement for the intimidating DXGI_SWAP_CHAIN_DESC1 structure:

```
SwapChainDescription1 description;
```

In this case, binary-compatible replacements are provided to ensure that DirectX sees the same type, but you get to use something a little more practical. This isn't unlike what the Microsoft .NET Framework provides with its P/Invoke wrappers. The default constructor provides suitable defaults for most desktop and Windows Store apps. You might, for example, want to override this for desktop apps to produce smoother rendering while resizing:

```
SwapChainDescription1 description;
description.SwapEffect = SwapEffect::Discard;
```

This swap effect, by the way, is also required when targeting Windows Phone 8, but it's not permitted in Windows Store apps. Go figure.

Many of the best libraries lead you to a working solution quickly and easily. Let's consider a concrete example. Direct2D provides a linear gradient brush. Creating such a brush involves three logical steps: defining the gradient stops, creating the gradient stop collection and then creating the linear gradient brush given that collection. **Figure 1** shows what this might look like using the Direct2D API directly.

With the help of dx.h it becomes a whole lot more intuitive:

```
GradientStop stops[] =
{
  GradientStop(0.0f, COLOR_WHITE),
  GradientStop(1.0f, COLOR_BLUE),
};

auto collection = target.CreateGradientStopCollection(stops);
auto brush = target.CreateLinearGradientBrush(collection);
```

Although this isn't dramatically shorter than the code in **Figure 1**, it's certainly easier to write, and you're far less likely to get it wrong the first time, especially with the help of IntelliSense. The library uses various techniques to produce a more enjoyable programming model. In this case, the CreateGradientStopCollection method is overloaded with a function template to deduce the size of the GradientStop array at compile time so there's no need to use the _countof macro.

What about error handling? Well, one of the prerequisites of producing such a concise programming model is the adoption of exceptions for error propagation. Consider the definition of the Direct3D Device AsDxgi method I mentioned before:

```
inline auto Device::AsDxgi() const -> Dxgi::Device2
{
  Dxgi::Device2 result;
  HR(m_ptr.CopyTo(result.GetAddressOf()));
  return result;
}
```

This is a very common pattern in the library. First, notice that the method is const. Virtually all of the methods in the library are const, as the only data member is the underlying ComPtr and there's no need to modify it. In the method body, you can see how the

resulting Device object comes to life. All the library classes provide move semantics, so even though this might appear to perform a number of copies—and, by inference, a number of AddRef/Release pairs—there is in fact none of that happening at run time. The HR wrapping the expression in the middle is an inline function that throws an exception if the result is not S_OK. Finally, the library will always try to return the most specific class to avoid the caller having to perform additional calls to QueryInterface to expose more functionality.

The previous example used the ComPtr CopyTo method, which effectively just calls QueryInterface. Here's another example from Direct2D:

```
inline auto BitmapBrush::GetBitmap() const -> Bitmap
{
  Bitmap result;
  (*this)->GetBitmap(result.GetAddressOf());
  return result;
}
```

This one is a little different in that it directly calls a method on the underlying COM interface. This pattern is in fact what constitutes the bulk of the library's code. Here I'm returning the bitmap with which the brush paints. Many Direct2D methods return void, as is the case here, so there's no need for the HR function to check the result. The indirection leading to the GetBitmap method might not be so obvious, however.

As I was prototyping early versions of the library, I had to make a choice between playing tricks with the compiler and playing tricks with COM. My early attempts involved playing tricks with C++ using templates, specifically type traits, but also compiler type traits (also known as intrinsic type traits). This was fun at first, but it quickly became apparent that I was making more work for myself.

You see, the library models the "is-a" relationship between COM interfaces as concrete classes. COM interfaces may only inherit directly from one other interface. With the exception of IUnknown itself, each COM interface must directly inherit from one other interface. Ultimately, this leads all the way back up the type hierarchy to IUnknown. I started by defining a class for each COM interface. The RenderTarget class contained an ID2D1RenderTarget interface pointer. The DeviceContext class contained an ID2D1DeviceContext interface pointer. This seems reasonable enough until you want to treat a DeviceContext as a RenderTarget. After all, the ID2D1DeviceContext interface is derived from the ID2D1RenderTarget interface. It would be quite reasonable to have a DeviceContext and want to pass it to a method that expects a RenderTarget as a reference parameter.

Unfortunately, the C++ type system doesn't see it that way. Using this approach, DeviceContext can't actually derive from RenderTarget or else it would hold two references. I tried a combination of move semantics and intrinsic type traits to correctly move the references around as needed. It almost worked, but there were cases where an extra AddRef/Release pair was being introduced. Ultimately, it proved too complex, and a simpler solution was needed.

Unlike C++, COM has a very well-defined binary contract. That is, after all, what COM is all about. As long as you stick to the agreed-upon rules, COM won't let you down. You can play tricks with COM, so to speak, and use C++ to your advantage rather than fighting against it. This means that each C++ class isn't holding a strongly typed COM interface pointer, but simply a generic reference to IUnknown. C++ adds the type safety back, and its rules for class inheritance—and, more recently, move semantics—let me once again treat these COM "objects" as C++ classes. Conceptually, I started with this:

```
class RenderTarget { ComPtr<ID2D1RenderTarget> ptr; };
class DeviceContext { ComPtr<ID2D1DeviceContext> ptr; };
```

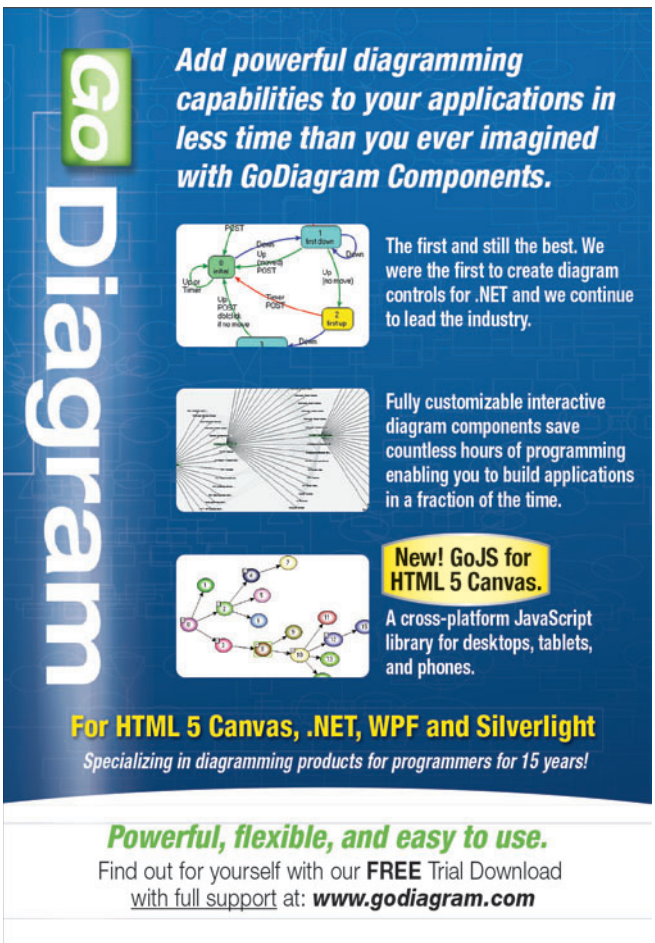
And I ended up with this:

```
class Object { ComPtr<IUnknown> ptr; };
class RenderTarget : public Object {};
class DeviceContext : public RenderTarget {};
```

As the logical hierarchy implied by the COM interfaces and their relationships is now embodied by a C++ object model, the programming model as a whole is far more natural and usable. There's a lot more to it, and I encourage you to take a closer look at the source code. As of this writing, it covers virtually all of Direct2D and the Windows Animation Manager, as well as useful chunks of DirectWrite, the Windows Imaging Component (WIC), Direct3D and DXGI. In addition, I'm regularly adding more functionality, so check back regularly. Enjoy! ■

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)*



JULIE LERMAN

A New Option for Creating OData: Web API

Microsoft .NET developers have been able to create OData feeds since before there was even an OData spec. By using WCF Data Services, you could expose an Entity Data Model (EDM) over the Web using Representational State Transfer (REST). In other words, you could consume these services through HTTP calls: GET, PUT, DELETE and so on. As the framework for creating these services evolved (and was renamed a few times along the way), the output evolved as well and eventually became encapsulated in the OData specification (odata.org). Now there's a great variety of client APIs for consuming OData—from .NET, from PHP, from JavaScript and from many other clients as well. But until recently, the only easy way to create a service was with WCF Data Services.

WCF Data Services is a .NET technology that simply wraps your EDM (.edmx, or a model defined via Code First) and then exposes that model for querying and updating through HTTP. Because the calls are URIs (such as http://mysite.com/mydataservice/Clients(34)), you can even query from a Web browser or a tool such as Fiddler. To create a WCF Data Service, Visual Studio provides an item template for building a data service using a set of APIs.

Now there's another way to create OData feeds—using an ASP.NET Web API. In this article I want to provide a high-level look at some of the differences between these two approaches and why you'd choose one over the other. I'll also look at some of the ways creating an OData API differs from creating a Web API.

API vs. Data Service at a High Level

A WCF Data Service is a System.Data.Services.DataService that wraps around an ObjectContext or DbContext you've already defined. When you declare the service class, it's a generic DataService of your context (that is, DataService<MyDbContext>). Because it starts out completely locked down, you set access permissions in the constructor to the DbSets from your context that you want the service to expose. That's all you need to do. The underlying DataService API takes care of the rest: interacting directly with your context, and querying and updating the database in response to your client application's HTTP requests to the service. It's also possible to add some customizations to the service, overriding some of its query or update logic. But for the most part, the point is to let the DataService take care of most of the interaction with the context.

Code download available at archive.msdn.microsoft.com/mag201306DataPoints.

With a Web API, on the other hand, you define the context interaction in response to the HTTP requests (PUT, GET and the like). The API exposes methods and you define the logic of the methods. You don't necessarily have to be interacting with Entity Framework or even a database. You could have in-memory objects that the client is requesting or sending. The access points won't just be magically created like they are with the WCF Data Service; instead, you control what's happening in response to those calls. This is the defining factor for choosing a service over an API to expose your OData. If most of what you want to expose is simple Create, Read, Update, Delete (CRUD) without the need for a lot of customization, then a data service will be your best choice. If you'll need to customize a good deal of the behavior, a Web API makes more sense.

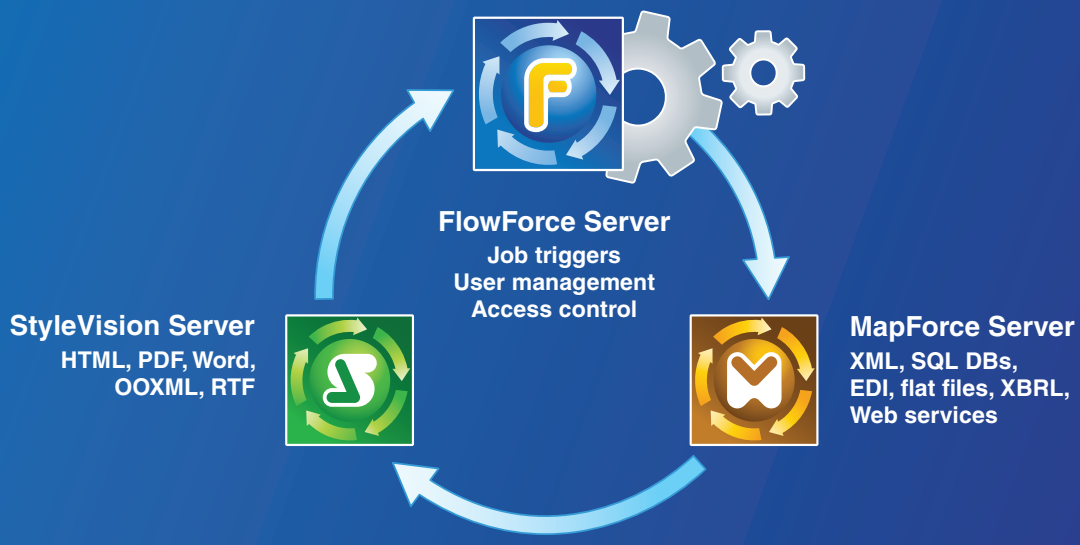
The access points won't just be magically created like they are with the data service; instead, you control what's happening in response to those calls.

I like the way Microsoft Integration MVP Matt Milner put it at a recent gathering: "WCF Data Services is for when you're starting with the data and model and just want to expose it. Web API is better when you're starting with the API and want to define what it should expose."

Setting the Stage with a Standard Web API

For those with limited experience with Web API, prior to looking at the new OData support I find it helpful to get a feel for the Web API basics and then see how they relate to creating a Web API that exposes OData. I'll do that here—first creating a simple Web API that uses Entity Framework as its data layer, and then converting it to provide its results as OData.

One use for a Web API is as an alternative to a standard controller in a Model-View-Controller (MVC) application, and you can create it as part of an ASP.NET MVC 4 project. If you don't want the front end, you can start with an empty ASP.NET Web



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.

ALTOVA®
flowforce®
SERVER 2013

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

application and add Web API controllers. However, for the sake of newbies, I'll start with an ASP.NET MVC 4 template because that provides scaffolding that will spit out some starter code. Once you understand how all the pieces go together, starting with an empty project is the right way to go.

So I'll create a new ASP.NET MVC 4 application and then, when prompted, choose the Empty template (not the Web API template, which is designed for a more robust app that uses views and is overkill for my purposes). This results in a project structured for an MVC app with empty folders for Models, Views and Controllers. **Figure 1** compares the results of the Empty template to the Web API template. You can see that an Empty template results in a much simpler structure and all I need to do is delete a few folders.

I also won't need the Models folder because I'm using an existing set of domain classes and a DbContext in separate projects to provide the model. I'll then use the Visual Studio tooling to create my first controller, which will be a Web API controller to interact with my DbContext and domain classes that I've referenced from my MVC project. My model contains classes for Airline, Passengers, Flights and some additional types for airline-related data.

Because I used the Empty template, I'll need to add some references in order to call into the DbContext—one to System.Data.Entity.dll and one to EntityFramework.dll. You can add both of these references by installing the EntityFramework NuGet package.

You can create a new Web API controller in the same manner as creating a standard MVC controller: right-click the Controllers folder in the solution and choose Add, then Controller. As you can see in **Figure 2**, you now have a template for creating an API controller with EF read and write actions. There's also an Empty API controller. Let's start with the EF read/write actions for a point of comparison to the controller we want for OData that will also use Entity Framework.

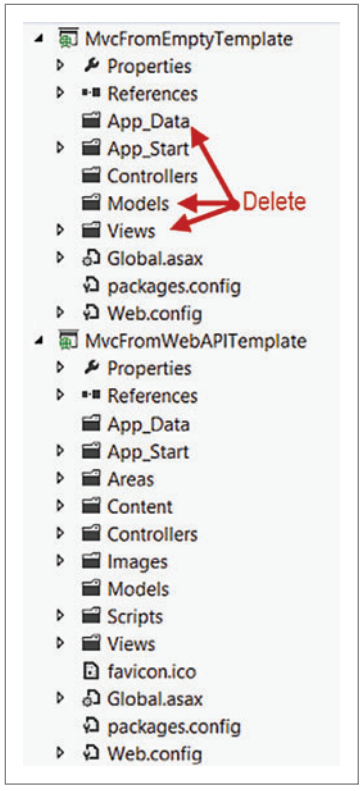


Figure 1 **ASP.NET MVC 4 Projects from Empty Template and Web API Template**

If you've created MVC controllers before, you'll see that the resulting class is similar, but instead of a set of view-related action methods, such as Index, Add and Edit, this controller has a set of HTTP actions.

For example, there are two Get methods, as shown in **Figure 3**. The first, GetAirlines, has a signature that takes no parameters and uses an instance of the AirlineContext (which the template scaffolding has named db) to return a set of Airline instances in an Enumerable. The other, GetAirline, takes an integer and uses that to find and return a particular airline.

The template adds comments to demonstrate how you'd use these methods.

After providing some configurations to my Web API, I can check it out directly in a browser using the example syntax on the port my app has assigned: http://localhost:1702/api/Airline. This is the default HTTP GET call and is therefore routed by the application to execute the GetAirlines method. Web API uses content negotiation to determine how the result set should be formatted. I'm using Google Chrome as my default browser, which triggered the results to be formatted as XML. The request from the client controls the format of the results. Internet Explorer, for example, sends no specific header information with respect to what format it accepts, so Web API will default to returning JSON. My XML results are shown in **Figure 4**.

If, following the guidance of the GetAirline method, I were to add an integer parameter to the request—http://localhost:1702/api/Airline/3—then only the single airline whose key (Id) is equal to 3 would be returned:

```
<Airline xmlns:i=http://www.w3.org/2001/XMLSchema-instance
  xmlns="http://schemas.datacontract.org/2004/07/DomainClasses">
    <Id>3</Id>
    <Legs/>
    <ModifiedDate>2013-02-22T00:00:00</ModifiedDate>
    <Name>Salt Lake Flyer</Name>
</Airline>
```

If I were to use Internet Explorer, or a tool such as Fiddler where I could explicitly control the request to the API to ensure I get JSON, the result of the request for Airline with the Id 3 would be returned as JSON:

```
{"Id":3,
  "Name":"Salt Lake Flyer",
  "Legs":[],
  "ModifiedDate":"2013-03-17T00:00:00"
}
```

These responses contain simple representations of the airline type with elements for each property: Id, Legs, ModifiedDate and Name.

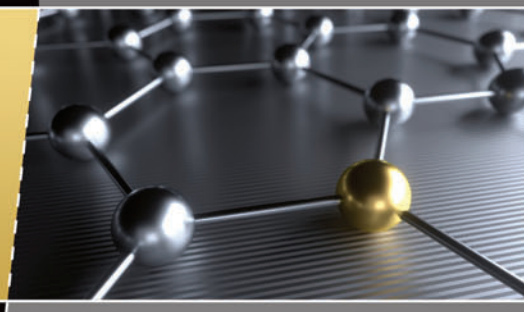
The controller also contains a PutAirline method that Web API will call in response to a PUT HTTP request. PutAirline contains code for using the AirlineContext to update an airline. There's also a PostAirline method for inserts and a DeleteAirline method for deleting. These can't be demonstrated in a browser URL but you can find plenty of getting-started content for Web API on MSDN, Pluralsight and elsewhere, so I'll move on to converting this to output its result according to the OData spec.

Turning Your Web API into an OData Provider

Now that you have a basic understanding of how Web API can be used to expose data using the Entity Framework, let's look at the special use of Web API to create an OData provider from your data model. You can force your Web API to return data formatted as OData by turning your controller into an OData controller—using a class that's available in the ASP.NET and Web Tools 2012.2 package—and then overriding its OData-specific methods. With this new type of controller, you won't even need the methods that were created by the template. In fact, a more efficient path for creating an OData controller is to choose the Empty Web API scaffolding template rather than the one that created the CRUD operations.

Empower Your Customers

New! v5.0



Create & Edit PDFs in .Net - ActiveX - WinRT

- Edit, process and print PDF 1.7 documents
- Create, fill-out and annotate PDF forms
- Fast and lightweight 32- and 64-bit components for .Net and ActiveX/COM
- New WinRT Component enables publishing C#, C++CX or Javascript apps to Windows Store
- New Postscript/EPS to PDF conversion module

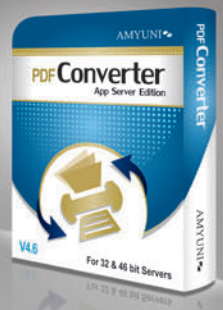
Complete Suite of Accurate PDF Components

- All your PDF processing, conversion and editing in a single package
- Combines Amyuni PDF Converter and PDF Creator for easy licensing, integration and deployment
- Includes our Microsoft certified PDF Converter printer driver
- Export PDF documents into other formats such as JPeg, PNG, XAML or HTML5

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 For Desktops and Servers



- 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 2012 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

CERTIFIED FOR Windows Server 2008 · Windows 7 · Windows 8 · Windows Server 2012 Certified

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

All trademarks are property of their respective owners. © Amyuni Technologies Inc. All rights reserved.

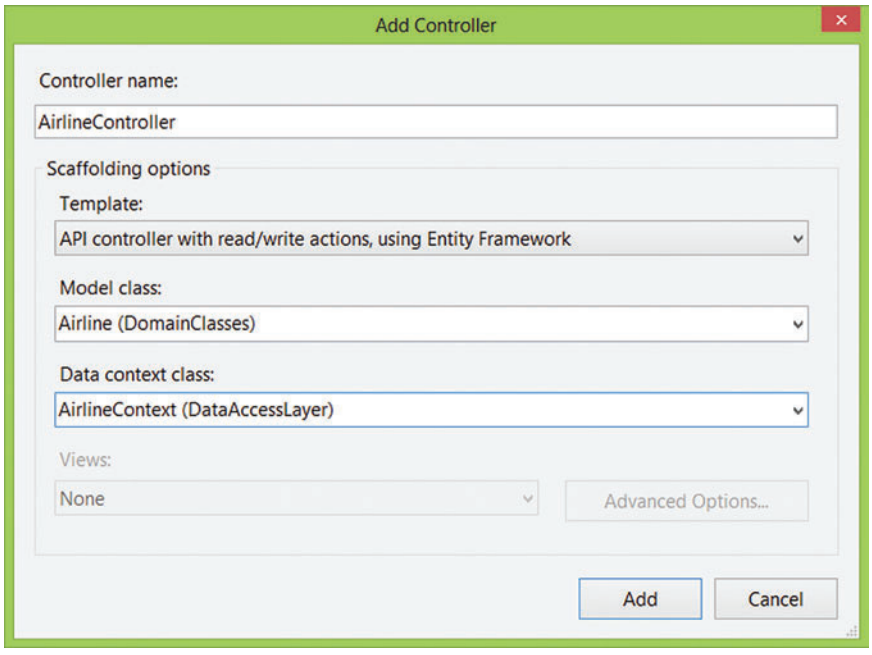


Figure 2 **A Template for Creating an API Controller with Pre-Populated Actions**

There are four steps I'll need to perform for this transition:

1. Make the controller a type of ODataController and implement its HTTP methods. I'll use a shortcut for this.
2. Define the available EntitySets in the project's WebAPIConfig file.
3. Configure the routing in WebAPIConfig.
4. Pluralize the name of the controller class to align with OData conventions.

Creating an ODataController Rather than inherit from ODataController directly, I'll use EntitySetController, which derives from ODataController and provides higher-level support by way of a number of virtual CRUD methods. I used NuGet to install the Microsoft ASP.NET Web API OData package for the proper assemblies that contain both of these controller classes.

Here's the beginning of my class, now inheriting from EntitySetController and specifying that the controller is for the Airline type:

```
public class AirlinesController : EntitySetController<Airline,int>
{
  private AirlineContext db = new AirlineContext();

  public override IQueryable<Airline> Get()
  {
    return db.Airlines;
  }
```

I've fleshed out the override for the Get method, which will return db.Airlines. Notice that I'm not calling ToList or AsEnumerable on the Airlines DbSet. The Get method needs to return an IQueryable of Airline and that's what db.Airlines does. This way, the consumer of the OData can define queries over this set, which will then get executed on the database, rather than pulling all of the Airlines into memory and then querying over them.

The HTTP methods you can override and add logic to are GET, POST (for inserts), PUT (for updates), PATCH (for merging updates) and DELETE. But for updates you'll actually use the virtual method CreateEntity to override the logic called for a POST, the UpdateEntity for logic invoked with PUT and PatchEntity for logic needed for the PATCH HTTP call. Additional virtual methods that can be part of this OData provider are: CreateLink, DeleteLink and GetEntityByKey.

In WCF Data Services, you control which CRUD actions are allowed per EntitySet by configuring the SetEntitySetAccessRule. But with Web API, you simply add the methods you want to support and leave out the methods you don't want consumers to access.

Specifying EntitySets for the API The Web API needs to know which EntitySets should be available to consumers. This confused me at first. I expected it to discover this by reading the AirlineContext. But as I thought about it more, I realized it's similar to using the SetEntitySetAccessRule in WCF Data Services. In WCF Data Services, you define which CRUD operations are allowed at the same time you expose a particular set. But with the Web API, you start by modifying the WebApiConfig.Register method to specify which sets will be part of the API and then use the methods in the controller to expose the particular CRUD operations. You specify the sets using the ODataModelBuilder—similar to the DbContext.ModelBuilder you may have used with Code First. Here's the code in the Register method of the WebApiConfig file to let my OData feed expose Airlines and Legs:

```
ODataModelBuilder modelBuilder = new ODataConventionModelBuilder();
                  modelBuilder.EntitySet<Airline>("Airlines");
                  modelBuilder.EntitySet<FlightLeg>("Legs");
```

Defining a Route to Find the OData Next, the Register method needs a route that points to this model so that when you call into the Web API, it will provide access to the EntitySets you defined:

```
Microsoft.Data.Edm.IEdmModel model = modelBuilder.GetEdmModel();
config.Routes.MapODataRoute("ODataRoute", "odata", model);
```

You'll see that many demos use "odata" for the RoutePrefix parameter, which defines the URL prefix for your API methods. While this is a good standard, you can name it whatever you like.

Figure 3 **Some of the Web API Controller Methods Created by the MVC Scaffolding**

```
public class AirlineController : ApiController
  {
    private AirlineContext db = new AirlineContext2();

    // GET api/Airline
    public IEnumerable<Airline> GetAirlines()
    {
      return db.Airlines.AsEnumerable();
    }

    // GET api/Airline/5
    public Airline GetAirline(int id)
    {
      Airline airline = db.Airlines.Find(id);
      if (airline == null)
      {
        throw new HttpResponseException
          (Request.CreateResponse(HttpStatusCode.NotFound));
      }
      return airline;
    }
```

Windows Azure

You've got it, now use it.
Accelerate development & test.
You could win* an Aston Martin.

Eligible MSDN subscribers now receive up to $150** in Windows Azure credits every month for no additional fee. Use virtual machines to accelerate development and test in the cloud. You can activate your benefits in less than five minutes.

Activate your Windows Azure MSDN Benefits
http://aka.ms/AzureContest

*No purchase necessary. Open to eligible Visual Studio Professional, Premium or Ultimate with MSDN subscribers as of June 1, 2013. Ends 11:59 p.m. PT on September 30, 2013. For full official rules including odds, eligibility and prize restrictions see website. Sponsor: Microsoft Corporation. Aston Martin is a trademark owned and licensed by Aston Martin Lagonda Limited. Image copyright Evox Images. All rights reserved.
**Actual credits based on subscription level.

So I'll change it just to prove my point:

```
config.Routes.MapODataRoute("ODataRoute", "oairlinedata", model);
```

Renaming the Controller Class The application template generates code that uses a singular naming convention for controllers, such as AirlineController and LegController. However, the focus of OData is on the EntitySets, which are typically named using the plural form of the entity name. And because my EntitySets are indeed plural, I need to change the name of my controller class to AirlinesController to align with the Airlines EntitySet.

Consuming the OData

Now I can work with the API using familiar OData query syntax. I'll start by requesting a listing of what's available using the request: http://localhost:1702/oairlinedata/. The results are shown in **Figure 5**.

The results show me that the service exposes Airlines and Legs. Next, I'll ask for a list of the Airlines as OData with http://localhost:1702/oairlinedata/Airlines. OData can be returned as XML or JSON. The default for Web API results is the JSON format:

```
{
  "odata.metadata":
    "http://localhost:1702/oairlinedata/$metadata#Airlines","value":[
    {
      "Id":1,"Name":"Vermont Balloons","ModifiedDate":"2013-02-26T00:00:00"
    },{
      "Id":2,"Name":"Olympic Airways","ModifiedDate":"2013-02-26T00:00:00"
    },{
      "Id":3,"Name":"Salt Lake Flyer","ModifiedDate":"2013-02-26T00:00:00"
    }
  ]
}
```

Figure 4 **Airline WebAPI's Response to GET, Displayed as XML in My Browser**

```
<ArrayOfAirline xmlns:i=http://www.w3.org/2001/XMLSchema-instance
  xmlns="http://schemas.datacontract.org/2004/07/DomainClasses">
    <Airline>
      <Id>1</Id>
      <Legs/>
      <ModifiedDate>2013-02-22T00:00:00</ModifiedDate>
      <Name>Vermont Balloon Transporters</Name>
    </Airline>
    <Airline>
      <Id>2</Id>
      <Legs/>
      <ModifiedDate>2013-02-22T00:00:00</ModifiedDate>
      <Name>Olympic Airways</Name>
    </Airline>
    <Airline>
      <Id>3</Id>
      <Legs/>
      <ModifiedDate>2013-02-22T00:00:00</ModifiedDate>
      <Name>Salt Lake Flyer</Name>
    </Airline>
</ArrayOfAirline>
```

Figure 5. **Requesting a List of Available Data**

```
http://localhost:1702/oairlinedata/
<service xmlns="http://www.w3.org/2007/app" xmlns:atom=
  "http://www.w3.org/2005/Atom" xml:base="http://localhost:1702/oairlinedata /">
    <workspace>
      <atom:title type="text">Default</atom:title>
      <collection href="Airlines">
        <atom:title type="text">Airlines</atom:title>
      </collection>
      <collection href="Legs">
        <atom:title type="text">Legs</atom:title>
      </collection>
    </workspace>
</service>
```

One of the many OData URI features is querying. By default, the Web API doesn't enable querying, as that imposes an extra load on the server. So you won't be able to use these querying features with your Web API until you add the Queryable annotation to the appropriate methods. For example, here I've added Queryable to the Get method:

```
[Queryable]
public override IQueryable<Airline> Get()
{
  return db.Airlines;
}
```

Now you can use the $filter, $inlinecount, $orderby, $sort and $top methods. Here's a query using the OData filter method:

```
http://localhost:1702/oairlinedata/Airlines?$filter=startswith(Name,'Vermont')
```

The ODataController allows you to constrain the queries so that consumers don't cause performance problems on your server. For example, you can limit the number of records that are returned in a single response. See the Web API-specific "OData Security Guidance" article at bit.ly/X0hyv3 for more details.

Just Scratching the Surface

I've looked at only a part of the querying capabilities you can provide with the Web API OData support. You can also use the virtual methods of the EntitySetController to allow updating to the database. An interesting addition to PUT, POST, and DELETE is PATCH, which lets you send an explicit and efficient request for an update when only a small number of fields have been changed, rather than sending the full entity for a POST. But the logic within your PATCH method needs to handle a proper update, which, if you're using Entity Framework, most likely means retrieving the current object from the database and updating it with the new values. How you implement that logic depends on knowing at what point in the workflow you want to pay the price of pushing data over the wire. It's also important to be aware that this release (with the ASP.NET and Web Tools 2012.2 package) supports only a subset of OData features. That means not all of the API calls you can make into an OData feed will work with an OData provider created with the Web API. The release notes for the ASP.NET and Web Tools 2012.2 package list which features are supported.

There's a lot more to learn than I can share in the limited space of this column. I recommend Mike Wasson's excellent series on OData in the official Web API documentation at bit.ly/14cfHIm. You'll learn about building all of the CRUD methods, using PATCH, and even using annotations to limit what types of filtering are allowed in your OData APIs and working with relationships. Keep in mind that many of the other Web API features apply to the OData API, such as how to use authorization to limit who can access which operations. Also, the .NET Web Development and Tools Blog (blogs.msdn.com/webdev) has a number of detailed blog posts about OData support in the Web API. ■

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

Thanks to the following technical experts for reviewing this article: *Jon Galloway (Microsoft) and Mike Wasson (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

BRUNO TERKALY AND
RICARDO VILLALOBOS

Architecting Multi-Tenant Applications in Windows Azure

There's a large and vibrant ecosystem of services providers that package up solutions and host them on cloud platforms. Generally speaking, these Software as a Service (SaaS) companies use multi-tenant architectures. You can find many real-world examples at bit.ly/vlPcyc. If you think about it, most of the popular applications on the Web, such as Facebook or Hotmail, are multi-tenant applications. This approach makes sense from a business perspective because compute and storage resources can be maximized by sharing them among multiple subscribers. After all, one of the main points of cloud computing is to maximize efficiencies by sharing a large pool of compute and storage resources.

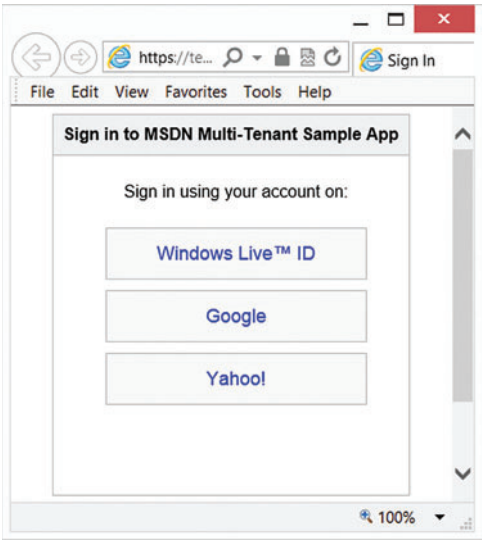


Figure 1 **Supporting Multiple Identity Providers**

We get frequent questions relating to best practices and patterns for implementing multi-tenant architectures using Windows Azure. This is the first of two columns that will introduce you to some key concepts and principles, as well as provide some guidance on where you can get the hands-on skills to start getting up to speed on some core building blocks.

The concept of multi-tenant software is simple. It's a single, comprehensive system that supports multiple client organizations (tenants), where each tenant runs securely in parallel on the same hardware and software, completely unaware that there are others. Done properly, it allows services providers to offer a lot of functionality to tenants while realizing extremely efficient utilization of Windows Azure. Tenants are focused on security, low cost and good performance.

Cloud architects must carefully consider some key pillars when building multi-tenant systems:

DEVELOP AND TEST IN THE CLOUD WITH WINDOWS AZURE

Deliver better-tested apps faster, on-premises or in the cloud. Activate your MSDN Windows Azure benefit now. You could win an Aston Martin!

aka.ms/AzureContest

Code download available at archive.msdn.microsoft.com/mag201306AzureInsider.

- Identity and security
- Data isolation and segregation
- Metering and performance analytics
- Scaling up and down while maintaining SLAs

We'll discuss the first two here and the latter two in our next column.

Identity

Identity plays a central role in multi-tenant architectures. For starters, tenants must be guaranteed that their private data isn't accessible by other users, except in cases where the goal is to share among subscribers. Microsoft SharePoint provides an example. You might have documents that are visible only to a specific tenant, but you might also have some that are shared by a workgroup (two or more tenants that may span organizations). The bottom line—identity plays a central role with respect to the visibility of data.

In today's world, identity management systems must be interoperable, adhering to open standards (as maintained by the Organization for the Advancement of Structured Information Standards, or OASIS), such as WS-Trust and WS-Security. Together, these standards allow multi-tenant systems to assess the presence of, and broker trust relationships between, tenant and services provider in a secure message exchange. Moreover, multi-tenant architectures typically support multiple identity providers, such as Google, Facebook, Yahoo! and Microsoft Live. In more sophisticated scenarios, services providers will support corporate identities, such as Active Directory. Support for Web single sign-on (SSO) is also important.

Claims-Based Identity Management

It's generally accepted that claims-based approaches to identity management provide the most value. For starters, claims-based approaches centralize the disparate parts of identity into a single abstract token composed of claims and an issuer or authority. The value of such approaches is that they're based on open standards and are highly interoperable. Claims-based identity management allows services providers to decouple applications from the vast, low-level plumbing code of identity management. Supporting Web services standards (WS-Trust, WS-Federation, WS-Security), token

1&1 WEB HOSTING

YOUR HOSTING PARTNER

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

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

- **Dual Hosting for Maximum Reliability**
  Your website hosted across multiple servers in two different data centers, and in two geographic locations.
- **Unlimited Bandwidth** (Traffic)
- **IPv6 Ready**
- **NEW: Choose your prepaid billing plan**

1&1 Starter Windows	1&1 Starter Linux
50 GB Web Space	
Unlimited Bandwidth (Traffic)	
250 E-Mail Accounts (2 GB each)	
24/7 Phone and E-mail Support	
NEW! ASP.NET/ .NET Framework **4, 4.5**	**NEW! PHP 5.4** and host multiple websites
NEW! 1 MSSQL **2012** Database (**1 GB**)	**10 MySQL** Databases (**1 GB** each)
NEW! ASP.NET MVC	**NEW!** Webspace Recovery and daily server backups
NEW! Dedicated App Pools	**2 Click & Build** Applications, like WordPress, Joomla!, TYPO3

As low as $0.99 per month*
Save up to $180

.com
domain
first year $0.99

1&1®

MEMBER OF united internet

DOMAINS | E-MAIL | WEB HOSTING | eCOMMERCE | SERVERS

Call **1 (877) 461-2631** or buy online

1and1.com

* Offers valid for a limited time only. Starter hosting package price of $0.99 per month is valid for a 36 month prepaid package; purchase amount of $35.64. Visit www.1and1.com for billing information and full promotional offer details. Program and pricing specifications and availability subject to change without notice. 1&1 and the 1&1 logo are trademarks of 1&1 Internet, all other trademarks are the property of their respective owners. © 2013 1&1 Internet. All rights reserved.

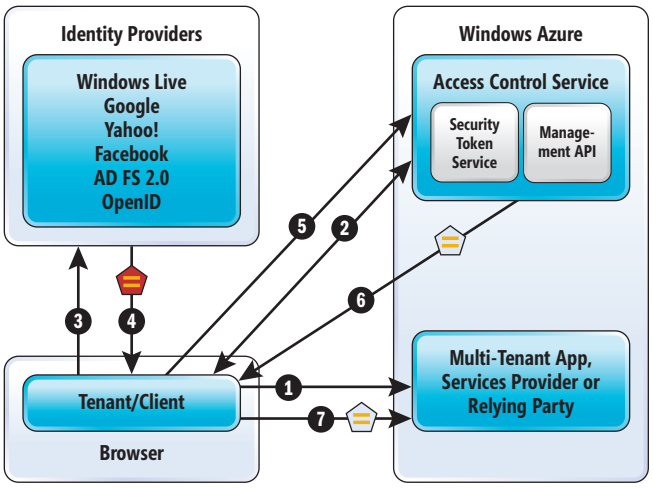


Figure 2 **High-Level View of the Access Control Service Portal**

formats (Security Assertion Markup Language [SAML], JSON Web Key [JWK], Simple Web Token [SWT]) and cryptography (X.509) isn't trivial. After all, these standards are constantly evolving, so encapsulating and abstracting identity management is crucial.

The answer to all these challenges is the Windows Azure Active Directory Access Control Service (ACS), which radically simplifies implementing a claims-based identity management system. The ACS removes this low-level plumbing and factors it out of a services provider's code, making it relatively easy to implement a claims-based identity management system. What makes the ACS so powerful is that much of the work can be done through the Web-based portal. The ACS greatly streamlines the provisioning of new tenants.

This means you, the developer, do not need to deal with the many complexities of identity management inside your application. The ACS solves difficult, time-intensive problems, such as:

- How to redirect unauthenticated requests to the chosen identity provider
- How to validate the incoming token issued by the identity provider
- How to parse the incoming token (read the claims)
- How to implement authorization checks
- How to transform tokens by adding, removing or changing the claims types and values

We recommend you run through a couple of Web-based tutorials to get started. You'll find a good one on authenticating users at bit.ly/uyDLkt. If you're running Visual Studio 2012, Vittorio Bertocci's four-part post at bit.ly/12ZOwN9 will provide the needed guidance. The tooling within Visual Studio continues to improve, freeing developers from hand-editing configuration files. You can download the tooling and SDKs from bit.ly/NN9NVE.

Figure 1 depicts the experience of a tenant with support for multiple identity providers.

Figure 2 represents what really takes place when you make use of the ACS. Tenants are completely unaware that the ACS is doing all the work behind the scenes. The workflow you see in **Figure 2** ends with your multi-tenant application getting a security token, typically in SAML 2.0 format. The real beauty is that the token your multi-tenant application receives has been standardized, regardless of the identity provider chosen by the tenant. The ACS transparently brokers the transformation from an identity provider's specific token format to a token format you specify at the ACS portal.

There is JavaScript that queries the ACS for the identity provider list you want your application to support, and then generates the login links for each Identity Provider. All the necessary redirections to get a standard security token to your application are handled by the system. The "relying party" is simply your multi-tenant application, because the multi-tenant application relies on an issuer to provide information about identity.

If you go through the tutorials mentioned previously, you'll see the screen shown in **Figure 3**. The Getting Started part of the ACS portal is divided into four sections, simplifying the process of integrating identity management into a cloud-based, multi-tenant application. Section 1 allows you to select from a list the identity providers you wish to support. For this walk-through, we'll choose Google, Microsoft Live ID and Yahoo! as the identity providers. With a little extra work, you could also include Facebook and Active Directory identities. Section 2 is where the binding takes place. It's where you'll specify the URL to which ACS returns the security token, typically the starting page of the application. After all, the ACS needs an endpoint to pass a token to in the format that's needed. In section 2 you also specify the desired token format.

Section 3 is where you select the claims you want the identity provider to include in the security token. The relying party (the services provider's multi-tenant application) will use these claims to both uniquely identify a tenant and to make authorization decisions, defining a tenant's privileges within the service. The available claims vary by identity provider; Google differs from Windows Live, for example. With Google and Yahoo!, the claims in the token can include the e-mail address, user name, nameidentifier and provider name. For Microsoft Live ID, however, you get only

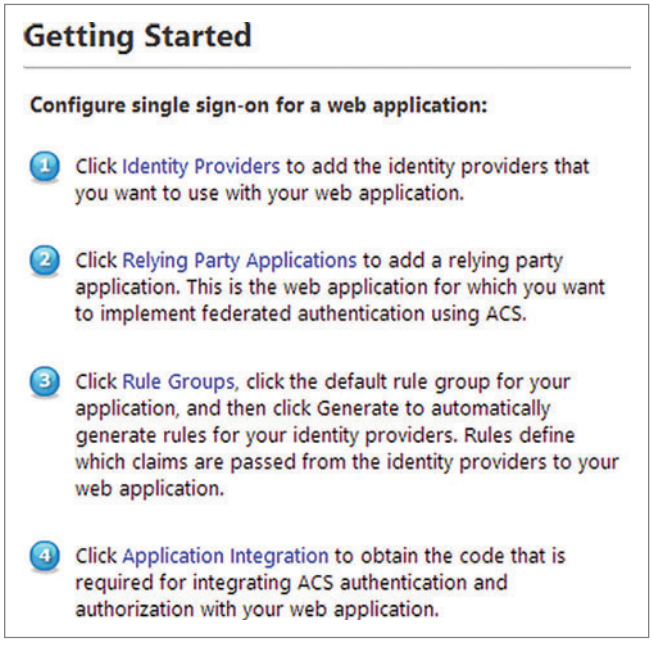


Figure 3 **The Windows Azure ACS Portal**

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

the nameidentifier and the provider name, not the e-mail or user name. You'll need to handle those subtleties within the application.

Section 4 lets you integrate some metadata and code into the service. This is done within the Visual Studio tooling. You need metadata inside the application configuration files to bind the application to the ACS. The tooling in Visual Studio 2012 makes this step a point-and-click experience, whereas with Visual Studio 2010 you'll need to manually edit the sytem.web section of web.config.

> Multi-tenant applications need to persist the logged-in user to permanent storage.

Microsoft recently released Windows Azure Active Directory, which allows you to take advantage of Web SSO with your line-of-business (LOB) applications, on-premises and in the cloud. If your goal is to implement identity management in an LOB application, where the application runs both on-premises and in the cloud, you'll want read the information at bit.ly/157yVPR. The documentation illustrates how Windows Azure Active Directory can take an existing LOB application and make it available for other Windows Azure Active Directory tenant administrators to use in their organizations.

The Bertocci blog post (previously mentioned) takes you to the screen in **Figure 4**, which shows that Identity and Access can be used to add configuration code in the multi-tenant application. Visual Studio 2012 completely eliminates the need to manually edit any configuration files. As you can see, we selected three identity providers and defined the realm and return URL for the multi-tenant application. The realm is simply a URI that identifies the application the user is logging in to. This URI also allows you to associate the claims for the application and the reply-to addresses. You'll change the realm and return URL once you deploy your application to a Microsoft datacenter. Finally, you'll add a management key that you get from the ACS portal to digitally sign tokens for security purposes.

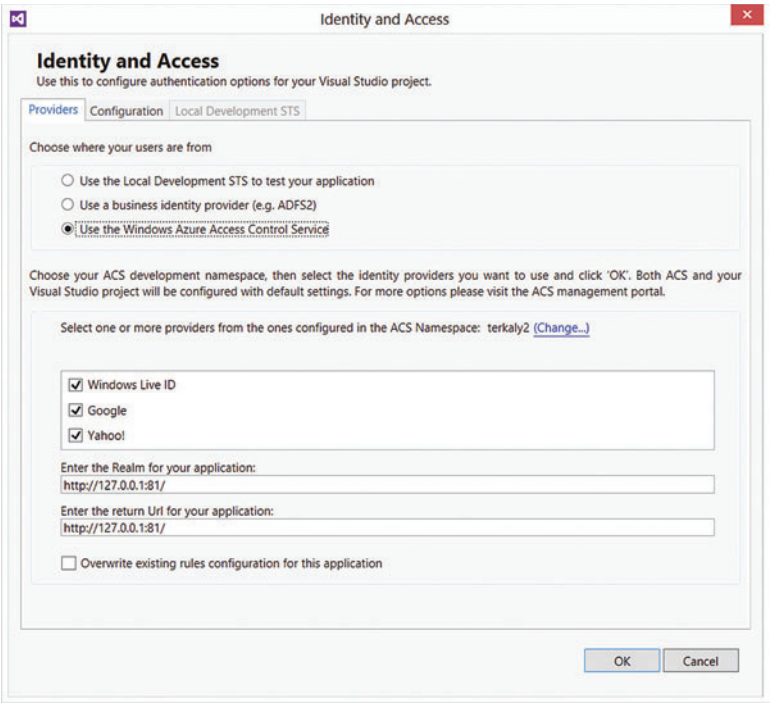


Figure 4 **The Visual Studio 2012 Identity and Access Dialog Box**

Adding Code to the Application

Multi-tenant applications need to persist the logged-in user to permanent storage. This is initially necessary for the provisioning process but may also be necessary if tenant activity is being tracked. The data store used in this article is Windows Azure SQL Database, but you could easily change this to Windows Azure Tables and could also include on-premises data stores. In the Page_Load method of the Default.aspx.cs file, we can easily read the security token and parse the claims, as seen here:

```
protected void Page_Load(object sender, EventArgs e)
{
  // Grab the ClaimsIdentity object. Think of it as a token
  // and set of claims for the currently logged in user.
  ClaimsIdentity ci =
    Thread.CurrentPrincipal.Identity as ClaimsIdentity;

  // Create a TenantTokenManager object that parses the claims.
  TenantTokenManager tenantTokenManager = new TenantTokenManager(ci);

  // Save logged in user to persistent storage.
  tenantTokenManager.SaveTenantToDatabase();
}
```

To encapsulate this logic, we'll add a TenantTokenManager class to our cloud project, shown in **Figure 5**.

The TenantTokenManager class encapsulates two important functions. First, it parses the claims from the security token. In our case, the token format is SAML, but it could just as easily be JWT or SWT. Note that simple LINQ queries can be used to extract individual claims from the token. And notice in the code that for Microsoft Live ID, you can't get any name or e-mail data. In this case, null values will return from the LINQ query. The comments in the code explain what the claims mean. We also added support for saving to the TenantTokenManager using the SaveToDatabase method, so that tenants can be provisioned and tracked.

The next logical step in the application is to make authorization decisions based on identity or the role membership of identities. For example, as shown in **Figure 6**, the stored procedure only returns records based on TenantId. This illustrates how you might start thinking about data isolation and segregation.

Data Isolation and Segregation

When it comes to combining identity with data in a multi-tenant system, there are a number of significant problems to solve. First, the architect must figure out how to keep sensitive data completely isolated from other tenants. Obviously, you can't allow things like credit-card numbers, Social Security numbers, e-mail and so forth to be compromised in any way. Each tenant must absolutely be guaranteed that its data is kept away from prying eyes. At the same time,

Aspose.Total

just got

BIGGER

Aspose.Diagram

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

Supported Files

VSD VTX
VSS VDW
VST VDX
VSX

NEW

Aspose.OCR

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

Supported Files

BMP
TIFF

NEW

Aspose.Imaging

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

Supported Files

PSD BMP
TIFF PNG
JPEG
GIF

NEW

Already own Aspose.Total for .NET?

These are yours for FREE!

Free Evaluations at www.aspose.com

ASPOSE
Your File Format Experts

Figure 5 **The TenantTokenManager Class**

```
public class TenantTokenManager
{
  // Claims that uniquely identify tenants.
  public System.Security.Claims.ClaimsIdentity SecurityToken { get; set; }
  public string IdentityProviderName                         { get; set; }
  public string IdentityProviderNameIdentifier               { get; set; }
  public string IdentityProviderEmail                        { get; set; }
  public string TenantId                                     { get; set; }
  public string TenantName                                   { get; set; }

  public TenantTokenManager(System.Security.Claims.ClaimsIdentity ci)
  {
    try
    {
      // Save a copy of the ClaimsIdentity security token.
      this.SecurityToken = ci;
      // Extract the provider name (Yahoo, Google, Microsoft Live).
      IdentityProviderName = (from c in ci.Claims
        where c.Type ==
        "http://schemas.microsoft.com/accesscontrolservice/
        2010/07/claims/identityprovider"
        select c.Value).SingleOrDefault();
      // Extract the nameidentifier (a unique identifier of
      // a tenant from the identity provider).
      IdentityProviderNameIdentifier = (from c in ci.Claims
        where c.Type ==
          "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier"
        select c.Value).SingleOrDefault();
      // Extract the emailaddress (not available in Microsoft Live ID).
      IdentityProviderEmail = (from c in ci.Claims
        where c.Type ==
          "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress"
        select c.Value).SingleOrDefault();
      // Create a unique TenantId based on nameidentifier
      // and provider name.
      TenantId =
        IdentityProviderName + "-" + IdentityProviderNameIdentifier;
      // Extract name from security token (not available
      // from Microsoft Live ID).
      TenantName = SecurityToken.Name == null ? "Not Avail." : SecurityToken.Name;

    } catch (Exception ex) { throw ex; }
  }
  public void SaveTenantToDatabase() { IdentityDataStore.
SaveTenantToDatabase(this); }
}
```

data may also need to be shared among tenants, such as the sharing of calendars, photos or simple text messages. Another problem to solve is the provisioning of audit trails, whereby services providers can track the usage and login patterns of tenants. This lets organizations track changes made by administrator-level users to help resolve unexpected application behavior.

In any multi-tenant system, there's the additional technical challenge to enable the services provider to maximize cost efficiencies while providing tenants with well-performing and flexible data services and data models. This challenge is especially difficult due to the sheer variety of data types. Consider that Windows Azure alone supports a multitude of storage options, such as tables, blobs, queues, SQL Database, Hadoop and more. The technical challenge for the services provider is significant, because each storage type is very different. Architects need to balance cost, performance, scalability and, ultimately, the data needs of tenants.

Let's take a quick look at the four common Windows Azure storage types. Blobs are used to store unstructured binary and text data. Typically, blobs are images, audio or other multimedia objects, though sometimes binary executable code is stored as a blob. Queues are used for storing messages that may be accessed by a tenant. They provide reliable messaging with the scaled instances of the multi-tenant application. Tables offer NoSQL capabilities for applications that require storage of large amounts of unstructured data. Finally, SQL Database provides a full-featured, relational Database as a Service (DBaaS) for applications that need this.

Blobs These are comparatively easy to understand relative to Windows Azure Tables or SQL Database. You can read more about blobs at bit.ly/VGHszH. A Windows Azure account can have many blob containers. A blob container can have many blobs. Most services providers will have more than one account. The typical approach taken is to designate a container name for each tenant. This gives you the ability to define access rights, measure performance and quantify consumption of the blob service. Recently, Microsoft has revised its network topology to dramatically improve the bandwidth between compute and storage resources to support storage node network speeds up to 10Gbps. Read more about this at bit.ly/PrxhAW.

Naming conventions for blobs may force you to maintain a map of container names to TenantIds. You can use e-mail addresses because they're unique, but Windows Live ID doesn't provide that claim within the security token. For Google and Yahoo!, you need to remove the "@" sign and "." because blob containers can only contain letters, numbers and dashes.

The code in **Figure 7** demonstrates an approach to provisioning blob containers. This code might be part of the sign-up and provisioning process for a tenant. Note that the TenantId is used as the container name. In a real-world scenario, of course, there would probably be some type of lookup table to provide container names based on TenantIds. In the code in **Figure 7**, the services provider has chosen to have a separate Windows Azure account (AccountTenantImages) to store the images of tenants. Note the code "blobClient.GetContainerReference(tm.TenantId)," which is where a blob container is provisioned for a new tenant.

Another important point is that blobs support user-defined metadata, which means that each blob can support one or more name-value pairs. However, blob storage doesn't allow you to query blobs on its metadata globally. To find information in metadata, you have two options. The first is to fetch all blobs from a blob container and then enumerate over them, searching the name-value pairs. The more scalable approach is to store metadata in table storage, along

Figure 6 **A Stored Procedure That Returns Records Based on TenantId**

```
CREATE PROCEDURE [dbo].[GetTenantData]
  @TenantId nvarchar(max)
AS
BEGIN
  SELECT
    [id],
    [TenantId],
    [TenantName],
    [ProviderName],
    [NameIdentifier],
    [Email],
    [SomeTenantData1],
    [SomeTenantData2]
  FROM TenantRecords
  WHERE TenantId = @TenantId

  return;
END
```

With ScaleOut GeoServer, the sky's no limit.

Global Data Integration

ScaleOut StateServer's industry-leading in-memory data grid technology is the key to fast data access and scalable performance. With ScaleOut GeoServer, now your applications can seamlessly access memory-based data worldwide.

Give Your Apps...

- Blazing Speed
- Global Data Access
- Transparent Data Migration

Take your application's scalability to new heights. Download your FREE trial copy of ScaleOut StateServer® and ScaleOut GeoServer® today!

SCALEOUT SOFTWARE

In-Memory Data Grids for the Enterprise

www.scaleoutsoftware.com | 503.643.3422

with the blob URL. This approach lets you query table storage to find the desired blob and then retrieve the blob from storage and query the metadata directly from a single blob.

Queues Because queues are typically used as mechanisms to provide reliable, asynchronous messaging within multi-tenant applications, tenants are generally insulated from them. With that said, an account has many queues and queues have many messages. Conceivably, the cloud architect could provision a separate queue for each tenant. But this decision would be based on the specific needs of the multi-tenant app. And keep in mind that this approach may not scale well and could become difficult to manage as the number of tenants rises beyond 100.

Tables There are many data isolation and segregation options available within Windows Azure Tables. For example, the services provider could provision one storage account per tenant. Because each storage account appears as a line item on your Windows Azure bill, this approach can be useful if you want to identify the precise costs per tenant. Another option is to combine multiple tenants in a single storage account. This approach enables you to group tenants by geographic region, by regulatory requirements and potentially by replication requirements. However, you still need to layer in a partitioning scheme so the tenants within one account don't get access to other tenants' data—unless, of course, data-sharing strategies are desired. One approach to hosting multiple tenants in a single account is to include the TenantId in the table name and give each tenant its own copy of the tables.

Yet another approach is to put multiple tenants in a single table, in which case you'd probably use a table's built-in partition keys and row keys to keep tenants' data separate from one another. It's common to use the TenantId as the partition key because it scales well while providing excellent query performance.

The code in **Figure 8** demonstrates how multiple tenants could share a single table. The code illustrates how a tenant might retrieve an e-mail address and phone number using Windows Azure Tables. Notice that the PartionKey is essentially the TenantId, which can be derived from the login credentials, either directly or through another lookup table.

Figure 7 **Provisioning a Blob Container for a Tenant**

```
// Grab the ClaimsIdentity object. Think of it as a token
// and set of claims for the currently logged in user.
ClaimsIdentity ci =
  Thread.CurrentPrincipal.Identity as ClaimsIdentity;
// Parse the claim.
TokenManager tm = new TokenManager(ci);
// Retrieve storage account from connection string.
CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
  Microsoft.WindowsAzure.CloudConfigurationManager.GetSetting(
  "AccountTenantImages"));
// Create the blob client.
CloudBlobClient blobClient = storageAccount.CreateCloudBlobClient();
// Retrieve a reference to a container.
CloudBlobContainer container = blobClient.GetContainerReference(tm.TenantId);
// Create the container if it doesn't already exist.
container.CreateIfNotExists();
```

Figure 8 **Segregating Data in Windows Azure Tables Using a PartionKey and a RowKey**

```
// Grab the ClaimsIdentity object. Think of it as a token
// and set of claims for the currently logged in user.
ClaimsIdentity ci =
  Thread.CurrentPrincipal.Identity as ClaimsIdentity;
// Parse the claim.
TokenManager tm = new TokenManager(ci);
// Retrieve storage account from connection string.
CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
  Microsoft.WindowsAzure.CloudConfigurationManager.GetSetting(
  "AccountTenantTables"));
// Create a cloud table client object.
CloudTableClient tableClient = storageAccount.CreateCloudTableClient();
// Create an email address table object.
CloudTable emailAddressTable =
  tableClient.GetTableReference("EmailAddressTable");
// Set up the filter for the partition key.
string pkFilter = TableQuery.GenerateFilterCondition("PartitionKey",
  QueryComparisons.Equal, tm.TenantId);
// Set up the filter for the row key.
string rkFilter = TableQuery.GenerateFilterCondition("RowKey",
  QueryComparisons.Equal, tm.IdentityProviderName);
// Combine the partition key filter and the Row Key filter.
string combinedFilter = TableQuery.CombineFilters(rkFilter,
  TableOperators.And, pkFilter);
// Construct a query.
TableQuery<EmailAddressEntity> query =
  new TableQuery<EmailAddressEntity>().Where(combinedFilter);
// Execute the query.
EmailAddressEntity emailAddressEntity =
  emailAddressTable.ExecuteQuery(query).First();
// Pull the data out that you searched for;
// do something with emailAddress and phoneNumber.
string emailAddress = emailAddressEntity.EMailAddress;
string phoneNumber = emailAddressEntity.PhoneNumber;
```

Wrapping Up

Architecting multi-tenant applications requires a good understanding of identity and data isolation and segregation issues. Services providers can insulate themselves from having to write a lot of the low-level plumbing code relating to identity management by taking advantage of the ACS. A multi-tenant application can support a variety of identity providers and avoid the obfuscation of its code base with some of the techniques provided here. Moreover, robust identity management simplifies the work it takes to either isolate or share tenant data.

In next month's column, we'll address the other key pillars of multi-tenant architectures—metering and performance analytics and scaling. In the meantime, we recommend that readers interested in more information read "Developing Multi-Tenant Applications for the Cloud, 3rd Edition," available at bit.ly/asHI9I. ■

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. He has published two applications to the Windows Store: Teach Kids Music and Kids Car Colors. 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.*

Terkaly and Villalobos jointly present at large industry conferences; for availability, e-mail them at bterkaly@microsoft.com or Ricardo.Villalobos@microsoft.com.

Thanks to the following technical experts for reviewing this article: *Patrick Butler Monterde (Microsoft) and Bhushan Nene (Microsoft)*

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!

XCEED
MULTI-TALENTED COMPONENTS
xceed.com

Code-Sharing Strategies for Windows Store and Windows Phone Apps

Doug Holland

With Visual Studio 2012, you have an excellent set of tools to build apps for Windows 8 and Windows Phone 8. Therefore, it's appropriate to explore how much code you're able to share between the Windows Store and Windows Phone versions of your apps.

You can write Windows Store apps using several different programming languages—XAML with C#, Visual Basic, C++ and even HTML5 with JavaScript.

Windows Phone 8 apps are usually written using XAML with either C# or Visual Basic, though the Windows Phone 8 SDK now lets you write Direct3D apps using XAML and C++. While the Windows Phone 8 SDK also provides a template for HTML5-based apps, these are merely XAML-based, with the WebBrowser control hosting HTML5-based Web pages.

In this article I'll explore three strategies for sharing code between Windows Store and Windows Phone apps: Portable Class Libraries (PCLs), Windows Runtime (WinRT) components (and Windows Phone Runtime components) and the Visual Studio Add as Link option. You can find further guidance on sharing code between Windows Store and Windows Phone apps at the Dev Center (aka.ms/sharecode).

It's worth noting that while there are many similarities between Windows Store and Windows Phone apps—live tiles, for example—these are distinct platforms for which you should specifically design the UX.

DOWNLOAD THE WINDOWS PHONE SDK TODAY

The Windows Phone SDK includes all of the tools that you need to develop apps and games for Windows Phone.

bit.ly/UbFlDG

This article discusses:

- Using the Model-View-ViewModel and dependency injection patterns to increase the level of code sharing
- Three strategies for sharing code between Windows Store and Windows Phone apps

Technologies discussed:

Windows 8, Windows Phone 8, Visual Studio 2012 Professional, C++

Code download available at:

archive.msdn.microsoft.com/mag201306CodeSharing

Architecture

The architectural principles that enable you to increase the percentage of code that can be shared are, in general, those that promote a separation of concerns. If you're already using patterns that promote a separation of concerns, such as the Model-View-ViewModel (MVVM) or Model-View-Controller (MVC), you'll find it easier to enable code sharing—and that's also the case if you use dependency-injection patterns in your architecture. You should definitely consider using these patterns when architecting new apps to increase the level of code sharing that can be achieved.

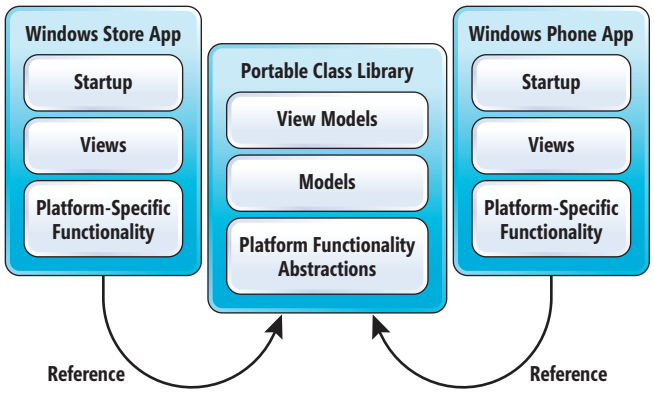


Figure 1 **Sharing Code Using the MVVM Design Pattern**

With existing apps, you may want to consider refactoring the architecture to promote a separation of concerns and therefore sharing of code. With the separation of concerns provided by MVVM or MVC there are additional benefits, such as the ability to have designers and developers work simultaneously. Designers design the UX with tools such as Expression Blend, while developers write code to bring the UX to life in Visual Studio.

Portable Class Libraries

The PCL project in Visual Studio 2012 enables cross-platform development, which lets you choose the target frameworks the resulting assembly will support. Introduced in Visual Studio 2010 as an optional add-in, the PCL project template is now included within Visual Studio Professional 2012 and above.

So what code can you share within a PCL?

PCLs are so named because they enable the sharing of portable code, and for code to be portable it must be managed code written in either C# or Visual Basic. Because a PCL produces a single binary, portable code does not employ conditional compilation directives; instead, platform-specific capabilities are abstracted using either interfaces or abstract base classes. When portable code needs to interact with platform-specific code, dependency-injection patterns are used to provide platform-specific implementations of the abstractions. When compiled, the PCL results in a single assembly that can be referenced from any project based on the target frameworks.

Figure 1 shows one recommended architectural approach to enable shared code using PCLs. With the MVVM pattern, the view models and models are contained within the PCL, along with abstractions of any platform-specific capabilities. The Windows Store and Windows Phone apps provide startup logic, views, and implementations of any abstractions of platform-specific capabilities. While the MVVM design pattern is not specifically required to enable portable code, the separation of concerns the pattern promotes results in a clean and extensible architecture.

The Add Portable Class Library dialog in Visual Studio 2012 is where you can select the target frameworks the resulting assembly will support.

> For code to be portable it must be managed code written in either C# or Visual Basic.

You might initially think you should check the box for Silverlight 5, but it isn't necessary for sharing code between Windows Store and Windows Phone apps. In fact, selecting Silverlight 5 means your portable code won't be able to take advantage of some very useful new types, such as the CallerMemberNameAttribute class introduced in the Microsoft .NET Framework 4.5.

If you've been developing for Windows Phone, you're most likely familiar with the MessageBox class that enables messages to be presented to the user. Windows Store apps use the Windows Runtime MessageDialog class for this purpose. Let's take a look at how to abstract this platform-specific functionality in a PCL.

The IMessagingManager interface in **Figure 2** abstracts the platform-specific functionality to display messages to the user. The

Figure 2 **IMessagingManager Interface**

```
/// <summary>
/// Provides an abstraction for platform-specific user messaging capabilities.
/// </summary>
public interface IMessagingManager
{
  /// <summary>
  /// Displays the specified message using platform-specific
  /// user-messaging capabilities.
  /// </summary>
  /// <param name="message">The message to be displayed to the user.</param>
  /// <param name="title">The title of the message.</param>
  /// <returns>A <see cref="T:MessagingResult"/>
  value representing the user's response.</returns>
  Task<MessagingResult> ShowAsync(string message, string title);

  /// <summary>
  /// Displays the specified message using platform-specific
  /// user-messaging capabilities.
  /// </summary>
  /// <param name="message">The message to be displayed to the user.</param>
  /// <param name="title">The title of the message.</param>
  /// <param name="buttons">The buttons to be displayed.</param>
  /// <returns>A <see cref="T:MessagingResult"/>
  value representing the user's response.</returns>
  Task<MessagingResult> ShowAsync(string message, string title,
    MessagingButtons buttons);
}
```

Figure 3 **MessagingButtons Enumeration**

```
/// <summary>
/// Specifies the buttons to include when a message is displayed.
/// </summary>
public enum MessagingButtons
{
  /// <summary>
  /// Displays only the OK button.
  /// </summary>
  OK = 0,

  /// <summary>
  /// Displays both the OK and Cancel buttons.
  /// </summary>
  OKCancel = 1,

  /// <summary>
  /// Displays both the Yes and No buttons.
  /// </summary>
  YesNo = 2
}
```

Figure 4 **MessagingResult Enumeration**

```
/// <summary>
/// Represents the result of a message being displayed to the user.
/// </summary>
public enum MessagingResult
{
  /// <summary>
  /// This value is not currently used.
  /// </summary>
  None = 0,

  /// <summary>
  /// The user clicked the OK button.
    /// </summary>
    OK = 1,

    /// <summary>
    /// The user clicked the Cancel button.
    /// </summary>
    Cancel = 2,

    /// <summary>
    /// The user clicked the Yes button.
    /// </summary>
    Yes = 6,
   /// <summary>
  /// The user clicked the No button.
  /// </summary>
  No = 7
}
```

IMessagingManager interface provides an overloaded ShowAsync method that takes the message and title of the message to be displayed to the user.

The ShowAsync method is overloaded to let you optionally specify buttons to display along with the message. The MessagingButtons enumeration provides a platform-independent abstraction for displaying an OK button, OK and Cancel buttons, or Yes and No buttons (see **Figure 3**).

The underlying integer values of the MessagingButtons enumeration were intentionally mapped to the Windows Phone MessageBoxButton enumeration in order to safely cast the MessagingButtons enumeration to the MessageBoxButton enumeration.

ShowAsync is an asynchronous method that returns a Task<MessagingResult> indicating the button the user clicked when dismissing the message. The MessagingResult enumeration (see **Figure 4**) is also a platform-independent abstraction.

In this example, the IMessagingManager interface and MessagingButtons and MessagingResult enumerations are portable, and therefore sharable within a PCL.

With the platform-specific functionality abstracted in the PCL, you need to provide platform-specific implementations of the IMessagingManager interface for both Windows Store and Windows Phone apps. **Figure 5** shows the implementation for Windows Phone apps, and **Figure 6** shows the implementation for Windows Store apps.

The Windows Phone version of the MessagingManager class uses the platform-specific MessageBox class to display the message. The underlying integer values of the MessagingButtons enumeration were intentionally mapped to the Windows Phone MessageBoxButton enumeration, which allows you to safely cast from the MessagingButtons enumeration to the MessageBoxButton enumeration. In the same way, the underlying integer values of the MessagingResult enumeration allow you to safely cast to the MessageBoxResult enumeration.

Figure 5 **MessagingManager—Windows Phone Implementation**

```
/// <summary>
/// Windows Phone implementation of the <see cref="T:IMessagingManager"/> interface.
/// </summary>
internal class MessagingManager : IMessagingManager
{
  /// <summary>
  /// Initializes a new instance of the <see cref="T:MessagingManager"/> class.
  /// </summary>
  public MessagingManager()
  {
  }

  /// <summary>
  /// Displays the specified message using platform-specific
  /// user-messaging capabilities.
  /// </summary>
  /// <param name="message">The message to be displayed to the user.</param>
  /// <param name="title">The title of the message.</param>
  /// <returns>A <see cref="T:MessagingResult"/>
      value representing the users response.</returns>
  public async Task<MessagingResult> ShowAsync(string message, string title)
  {
    MessagingResult result = await this.ShowAsync(message, title,
      MessagingButtons.OK);

    return result;
  }

  /// <summary>
  /// Displays the specified message using platform-specific
  /// user-messaging capabilities.
  /// </summary>
  /// <param name="message">The message to be displayed to the user.</param>
  /// <param name="title">The title of the message.</param>
  /// <param name="buttons">The buttons to be displayed.</param>
  /// <exception cref="T:ArgumentException"/>
  /// The specified value for message or title is <c>null</c> or empty.
  /// </exception>
  /// <returns>A <see cref="T:MessagingResult"/>
  /// value representing the users response.</returns>
  public async Task<MessagingResult> ShowAsync(
    string message, string title, MessagingButtons buttons)
  {
    if (string.IsNullOrEmpty(message))
    {
      throw new ArgumentException(
        "The specified message cannot be null or empty.", "message");
    }

    if (string.IsNullOrEmpty(title))
    {
      throw new ArgumentException(
        "The specified title cannot be null or empty.", "title");
    }

    MessageBoxResult result = MessageBoxResult.None;

    // Determine whether the calling thread is the thread
    // associated with the Dispatcher.
    if (App.RootFrame.Dispatcher.CheckAccess())
    {
      result = MessageBox.Show(message, title, (MessageBoxButton)buttons);
    }
    else
    {
      // Execute asynchronously on the thread the Dispatcher is associated with.
      App.RootFrame.Dispatcher.BeginInvoke(() =>
      {
        result = MessageBox.Show(message, title, (MessageBoxButton)buttons);
      });
    }

    return (MessagingResult) result;
  }
}
```

When the web means business

Performance
Elegance
Productivity

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

DevExpress

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

u dxp dxt | asp win wpf sl | cr vcl

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

The Windows Store version of the MessagingManager class in **Figure 6** uses the Windows Runtime MessageDialog class to display the message. The underlying integer values of the MessagingButtons enumeration were intentionally mapped to the Windows Phone MessageBoxButton enumeration, which allows you to safely cast from the MessagingButtons enumeration to the MessageBoxButton enumeration.

Dependency Injection

With the application architecture defined as illustrated in **Figure 1**, IMessagingManager provides the platform-specific abstraction for messaging users. I'll now use dependency-injection patterns to inject platform-specific implementations of this abstraction into the portable code. In the example in **Figure 7**, the HelloWorldViewModel uses constructor injection to inject a platform-specific implementation of the IMessagingManager interface. The HelloWorldViewModel.DisplayMessage method then uses the injected implementation to message the user. To learn more about dependency injection, I recommend reading "Dependency Injection in .NET" by Mark Seemann (Manning Publications, 2011, bit.ly/dotnetdi).

Windows Runtime Components

Windows Runtime components let you share non-portable code between Windows Store and Windows Phone apps. The components are not binary-compatible, however, so you'll also need to create comparable Windows Runtime and Windows Phone Runtime component projects to leverage code across both platforms. While you'll have to include projects within your solution for both Windows Runtime components and Windows Phone Runtime components, these projects are built using the same C++ source files.

Figure 6 **MessagingManager—Windows Store Implementation**

```
/// <summary>
/// Windows Store implementation of the <see cref="T:IMessagingManager"/> interface.
/// </summary>
internal class MessagingManager : IMessagingManager
{
  /// <summary>
  /// Initializes a new instance of the <see cref="T:MessagingManager"/> class.
  /// </summary>
  public MessagingManager()
  {
  }

  /// <summary>
  /// Displays the specified message using platform-specific
  /// user-messaging capabilities.
  /// </summary>
  /// <param name="message">The message to be displayed to the user.</param>
  /// <param name="title">The title of the message.</param>
  /// <returns>A <see cref="T:MessagingResult"/>
      value representing the users response.</returns>
  public async Task<MessagingResult> ShowAsync(string message, string title)
  {
    MessagingResult result = await this.ShowAsync(message, title,
      MessagingButtons.OK);

    return result;
  }

  /// <summary>
  /// Displays the specified message using platform-specific
  /// user-messaging capabilities.
  /// </summary>
  /// <param name="message">The message to be displayed to the user.</param>
  /// <param name="title">The title of the message.</param>
  /// <param name="buttons">The buttons to be displayed.</param>
  /// <exception cref="T:ArgumentException"/>
  /// The specified value for message or title is <c>null</c> or empty.
  /// </exception>
  /// <exception cref="T:NotSupportedException"/>
  /// The specified <see cref="T:MessagingButtons"/> value is not supported.
  /// </exception>
  /// <returns>A <see cref="T:MessagingResult"/>
  /// value representing the users response.</returns>
  public async Task<MessagingResult> ShowAsync(
    string message, string title, MessagingButtons buttons)
  {
    if (string.IsNullOrEmpty(message))
    {
      throw new ArgumentException(
        "The specified message cannot be null or empty.", "message");
    }

    if (string.IsNullOrEmpty(title))
    {
      throw new ArgumentException(
        "The specified title cannot be null or empty.", "title");
    }

    MessageDialog dialog = new MessageDialog(message, title);

    MessagingResult result = MessagingResult.None;

    switch (buttons)
    {
      case MessagingButtons.OK:
        dialog.Commands.Add(new UICommand("OK",
          new UICommandInvokedHandler((o) => result = MessagingResult.OK)));
        break;

      case MessagingButtons.OKCancel:
        dialog.Commands.Add(new UICommand("OK",
          new UICommandInvokedHandler((o) => result = MessagingResult.OK)));
        dialog.Commands.Add(new UICommand("Cancel",
          new UICommandInvokedHandler((o) => result = MessagingResult.Cancel)));
        break;

      case MessagingButtons.YesNo:
        dialog.Commands.Add(new UICommand("Yes",
          new UICommandInvokedHandler((o) => result = MessagingResult.Yes)));
        dialog.Commands.Add(new UICommand("No",
          new UICommandInvokedHandler((o) => result = MessagingResult.No)));
        break;

      default:
        throw new NotSupportedException(
          string.Format("MessagingButtons.{0} is not supported.",
          buttons.ToString()));
            }

    dialog.DefaultCommandIndex = 1;

    // Determine whether the calling thread is the
    // thread associated with the Dispatcher.
    if (Window.Current.Dispatcher.HasThreadAccess)
    {
      await dialog.ShowAsync();
    }
    else
    {
      // Execute asynchronously on the thread the Dispatcher is associated with.
      await Window.Current.Dispatcher.RunAsync(
        CoreDispatcherPriority.Normal, async () =>
      {
        await dialog.ShowAsync();
      });
    }

    return result;
  }
}
```

VS Anywhere

Something is happening in Visual Studio

https://vsanywhere.com/vsm/

(Discount coupon available at this link)

- Team cooperation on steroids
- Writing code simultaneously

Visual Studio®

VS Anywhere

Figure 7 **Portable HelloWorldViewModel Class**

```
/// <summary>
/// Provides a portable view model for the Hello World app.
/// </summary>
public class HelloWorldViewModel : BindableBase
{
  /// <summary>
  /// The message to be displayed by the messaging manager.
  /// </summary>
  private string message;

  /// <summary>
  /// The title of the message to be displayed by the messaging manager.
  /// </summary>
  private string title;

  /// <summary>
  /// Platform specific instance of the <see cref="T:IMessagingManager"/> interface.
  /// </summary>
  private IMessagingManager MessagingManager;

  /// <summary>
  /// Initializes a new instance of the <see cref="T:HelloWorldViewModel"/> class.
  /// </summary>
  public HelloWorldViewModel(IMessagingManager messagingManager,
    string message, string title)
  {
    this.messagingManager = MessagingManager;

    this.message = message;

    this.title = title;

    this.DisplayMessageCommand = new Command(this.DisplayMessage);
  }

  /// <summary>
  /// Gets the display message command.
  /// </summary>
  /// <value>The display message command.</value>
  public ICommand DisplayMessageCommand
  {
    get;
    private set;
  }

  /// <summary>
  /// Displays the message using the platform-specific messaging manager.
  /// </summary>
  private async void DisplayMessage()
  {
    await this.messagingManager.ShowAsync(
      this.message, this.title, MessagingButtons.OK);
  }
}
```

With the ability to share native C++ code between Windows Store and Windows Phone apps, Windows Runtime components are an excellent choice for writing computationally intensive operations using C++ to attain optimal performance.

API definitions within Windows Runtime components are exposed in the metadata contained in .winmd files. Using this metadata, language projections let the consuming language determine how the API is consumed within that language. **Figure 8** shows the supported languages for creating and consuming

Figure 8 **Creating and Consuming Windows Runtime Components**

Platform	Create	Consume
Windows Runtime Components	C++, C#, Visual Basic	C++, C#, Visual Basic, JavaScript
Windows Phone Runtime Components	C++	C++, C#, Visual Basic

Figure 9 **Fibonacci.h**

```
#pragma once

namespace MsdnMagazine_Fibonacci
{
  public ref class FibonacciCalculator sealed
  {
  public:
    FibonacciCalculator();

    uint64 GetFibonacci(uint32 number);

  private:
    uint64 GetFibonacci(uint32 number, uint64 p0, uint64 p1);
  };
}
```

Windows Runtime components. At the time of this writing, only C++ is supported for creating both types of components.

In the following example, I'll demonstrate how a C++ class designed to calculate Fibonacci numbers can be shared across Windows Store and Windows Phone apps. **Figure 9** and **Figure 10** show the implementation of the FibonacciCalculator class in C++/Component Extensions (CX).

In **Figure 11** you can see the solution structure in Visual Studio Solution Explorer for the samples that accompany this article, showing the same C++ source files contained in both components.

Visual Studio Add as Link Capability

When adding an existing item to a project in Visual Studio, you might have noticed the little arrow to the right of the Add button. If you click that arrow, you'll be presented with the option to either Add or Add as Link. If you choose the default option to Add a file, the file will be copied to the project and the two copies of the file will exist separately on disk and within source control (if used). If you choose to Add as Link there will be only a single instance of the file on disk and within source control, which can be extremely useful for versioning. When adding existing files within Visual C++ projects this is the default behavior, and therefore the Add Existing Item dialog doesn't provide an option on the Add button. The Dev Center provides further guidance on sharing code with Add as Link at bit.ly/addaslink.

Windows Runtime APIs are not portable, and therefore can't be shared within a PCL. Windows 8 and Windows Phone 8 expose a subset of the Windows Runtime API, so code can be written against

Figure 10 **Fibonacci.cpp**

```
#include "pch.h"
#include "Fibonacci.h"

using namespace Platform;

using namespace MsdnMagazine_Fibonacci;

FibonacciCalculator::FibonacciCalculator()
{
}

uint64 FibonacciCalculator::GetFibonacci(uint32 number)
{
  return number == 0 ? 0L : GetFibonacci(number, 0, 1);
}

uint64 FibonacciCalculator::GetFibonacci(uint32 number, uint64 p0, uint64 p1)
{
  return number == 1 ? p1 : GetFibonacci(number - 1, p1, p0 + p1);
}
```

this subset and subsequently shared between both apps using Add as Link. The Dev Center provides details on the shared subset of the Windows Runtime API at bit.ly/wpruntime.

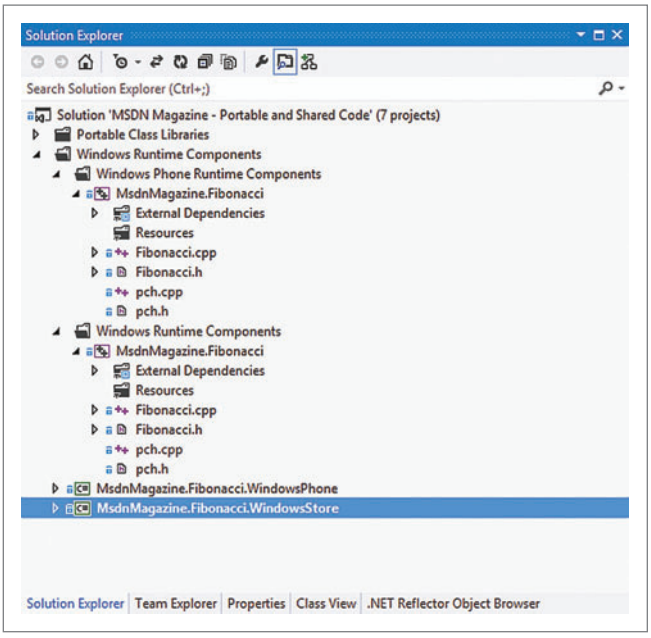


Figure 11 **Visual Studio Solution Explorer**

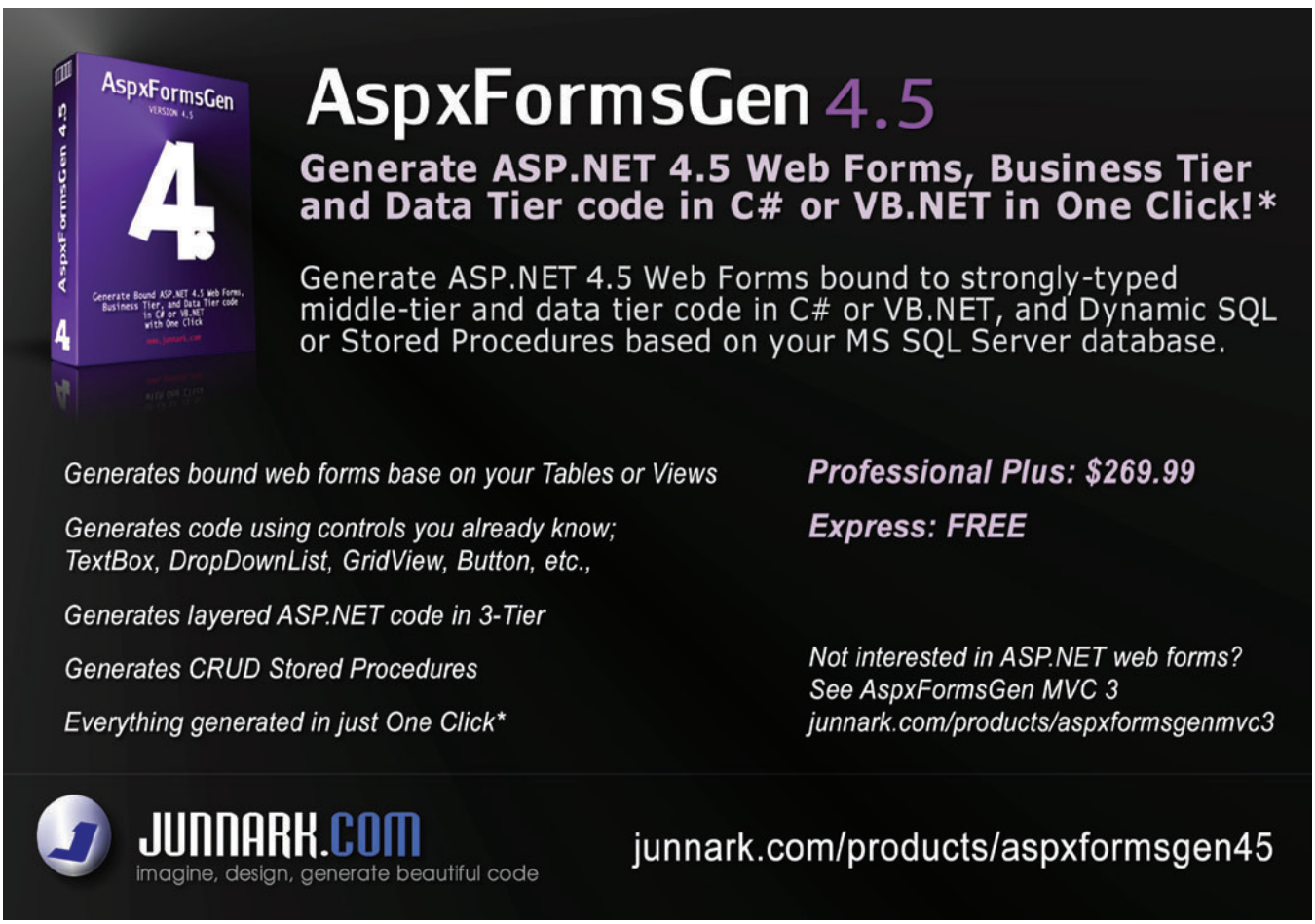
Wrapping Up

With the releases of Windows 8 and Windows Phone 8, you could begin to explore ways in which you could share code between the two platforms. In this article, I explored how portable code can be shared using binary-compatible PCLs and how platform-specific capabilities can be abstracted. I then demonstrated how non-portable native code can be shared using Windows Runtime components. Finally, I discussed the Visual Studio Add as Link option.

In terms of architecture, I noted that patterns that promote a separation of concerns, such as MVVM, can be useful for enabling code sharing, and that dependency-injection patterns enable shared code to leverage platform-specific capabilities. The Windows Phone Dev Center provides further guidance on sharing code between Windows Store and Windows Phone apps at aka.ms/sharecode, and also provides the PixPresenter sample app at bit.ly/pixpresenter. ■

Doug Holland *works as a senior architect evangelist in the Microsoft Developer and Platform Evangelism team. He has spent the last few years working with strategic partners to bring consumer-focused apps to Windows and Windows Phone. He's a former Visual C# MVP and Intel Black Belt Developer and an author of "Professional Windows 8 Programming: Application Development with C# and XAML" (Wrox, 2012), available at bit.ly/prowin8book.*

Thanks to the following technical experts for reviewing this article: *Andrew Byrne (Microsoft), Doug Rothaus (Microsoft) and Marian Laparu (Microsoft)*



Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

20 LIVE! CELEBRATING TWENTY YEARS

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 REDJUN4

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

TURN THE PAGE FOR MORE EVENT DETAILS

vslive.com/redmond

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

YOUR BACKSTAGE PASS TO THE MICROSOFT PLATFORM

Visual Studio Live! has partnered with the Hyatt Regency Bellevue for conference attendees at a special reduced rate.

CONNECT WITH VISUAL STUDIO LIVE!

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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

Register at vslive.com/redmond

Use Promo Code REDJUN4

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

AGENDA AT-A-GLANCE

Windows 8 / WinRT | **Web and JavaScript Development**

START TIME	END TIME	
7:00 AM	8:00 AM	
8:00 AM	12:00 PM	**MW01** - Workshop: Building Windows 8 Applications - *Rockford Lhotka*
12:00 PM	2:30 PM	
2:30 PM	6:00 PM	**MW01** - Workshop: Building Windows 8 Applications - *Rockford Lhotka*

START TIME	END TIME	
7:30 AM	8:30 AM	
8:30 AM	9:30 AM	
9:45 AM	11:00 AM	**T01** - Interaction and Navigation Patterns in Windows 8 Applications - *Billy Hollis*
11:15 AM	12:30 PM	**T06** - Introduction to the WinRT API - *Jason Bock*
12:30 PM	2:30 PM	
2:30 PM	3:45 PM	**T11** - A Primer in Windows 8 Development with WinJS - *Philip Japikse*
3:45 PM	4:15 PM	
4:15 PM	5:30 PM	**T16** - Getting Beyond the Datagrid in Windows 8 - *Billy Hollis*
5:30 PM	7:30 PM	

START TIME	END TIME	
7:30 AM	8:00 AM	
8:00 AM	9:00 AM	
9:15 AM	10:30 AM	**W01** - Expression Blend 5 for Developers: Design Your XAML or HTML5\CSS3 UI Faster - *Ben Hoelting*
10:45 AM	12:00 PM	**W06** - Using Your Favorite JavaScript Frameworks When Developing Windows Store Apps - *Keith Burnell*
12:00 PM	1:30 PM	
1:30 PM	2:45 PM	**W11** - Migrating from WPF or Silverlight to WinRT - *Rockford Lhotka*
2:45 PM	3:15 PM	
3:15 PM	4:30 PM	**W16** - Sharing Code Between Windows 8 and Windows Phone 8 apps - *Ben Dewey*
8:00 PM	10:00 PM	

START TIME	END TIME	
7:30 AM	8:00 AM	
8:00 AM	9:15 AM	**TH01** - Windows Azure Mobile Services: Backend for Your Windows 8, iOS, and Android Apps - *Sasha Goldshtein*
9:30 AM	10:45 AM	**TH06** - Win8 + Cloud Services - *Rockford Lhotka*
11:00 AM	12:15 PM	**TH11** - Demystifying LOB Deployments in Windows 8 and Windows Phone 8 - *Tony Champion*
12:15 PM	2:45 PM	
2:45 PM	4:00 PM	**TH16** - Deep Dive into the Windows 8 Background APIs - *Tony Champion*
4:15 PM	5:30 PM	**TH21** - Windows 8 Apps with MVVM, HTML/JS, and Web API (An eCommerce Story) - *Ben Dewey*

START TIME	END TIME	
7:30 AM	8:00 AM	
8:00 AM	12:00 PM	**FW01** - Workshop: Rich Data HTML Mobile and
12:00 PM	1:00 PM	
1:00 PM	5:00 PM	**FW01** - Workshop: Rich Data HTML Mobile and

**Speakers and sessions subject to change*

REDMOND, WA | AUGUST 19-23, 2013
MICROSOFT CAMPUS

Visual Studio 2012 / .NET 4.5	SharePoint / Office	Azure / Cloud Computing	Data Management	Mobile Development	SQL Server

Visual Studio Live! Pre-Conference Workshops: Monday, August 19, 2013 (Separate entry fee required)

Pre-Conference Workshop Registration - Coffee and Morning Pastries			
MW02 - Workshop: End-to-End Service Orientation - Designing, Developing, & Implementing Using WCF and the Web API - *Miguel Castro*		**MW03** - Workshop: UX Design Bootcamp for Developers and Analysts - *Billy Hollis*	
Lunch @ The Mixer - Visit the Microsoft Company Store & Visitor Center			
MW02 - Workshop: End-to-End Service Orientation - Designing, Developing, & Implementing Using WCF and the Web API - *Miguel Castro*		**MW03** - Workshop: UX Design Bootcamp for Developers and Analysts - *Billy Hollis*	

Visual Studio Live! Day 1: Tuesday, August 20, 2013

Registration - Coffee and Morning Pastries			
Keynote: To Be Announced			
T02 - Controlling ASP.NET MVC4 - *Philip Japikse*	**T03** - Intro to Azure - *Vishwas Lele*	**T04** - Data Visualization with SharePoint and SQL Server - *Paul Swider*	**T05** - Building Windows 8 Line of Business Apps - *Robert Green*
T07 - I'm Not Dead Yet! AKA The Resurgence of Web Forms - *Philip Japikse*	**T08** - Building Apps for Azure - *Vishwas Lele*	**T09** - A Developers Perspective on the Social Architecture of SharePoint 2013 - *Paul Swider*	**T10** - Working With Data in Windows Store Apps - *Robert Green*
Lunch - Visit Exhibitors			
T12 - jQuery Fundamentals - *Robert Boedigheimer*	**T13** - Moving Web Apps to the Cloud - *Eric D. Boyd*	**T14** - Developing Using the SharePoint 2013 REST Services - *Matthew DiFranco*	**T15** - Session To Be Announced
Sponsored Break - Visit Exhibitors			
T17 - Fiddler and Your Website - *Robert Boedigheimer*	**T18** - JavaScript, Meet Cloud: Node.js on Windows Azure - *Sasha Goldshtein*	**T19** - What's New in SharePoint 2013 Workflow Development? - *Matthew DiFranco*	**T20** - Windows Store Application Contracts and Extensibility - *Brian Peek*
Microsoft Ask the Experts & Exhibitor Reception – Attend Exhibitor Demos			

Visual Studio Live! Day 2: Wednesday, August 21, 2013

Registration - Coffee and Morning Pastries			
Keynote: To Be Announced			
W02 - Web API 101 - *Deborah Kurata*	**W03** - EF Code First: Magic: Unicorns and Beyond - *Keith Burnell*	**W04** - Building Apps for SharePoint 2013 - *Andrew Connell*	**W05** - What's New in the .NET 4.5 BCL - *Jason Bock*
W07 - Exposing Data Services with ASP.NET Web API - *Brian Noyes*	**W08** - Display Maps in Windows Phone 8 - *Al Pascual*	**W09** - Deep Dive into the Cloud App Model for SharePoint - *Keenan Newton*	**W10** - Understanding Dependency Injection and Those Pesky Containers - *Miguel Castro*
Luncheon Round Table - Visit Exhibitors			
W12 - Knocking It Out of the Park, with Knockout.js - *Miguel Castro*	**W13** - Busy Developer's Guide to Cassandra - *Ted Neward*	**W14** - A Deep Dive into Creating Apps for Office - *Keenan Newton*	**W15** - Mastering Visual Studio 2012 - *Deborah Kurata*
Sponsored Break - Exhibitor Raffle @ 3:00 pm (Must be present to win)			
W17 - Tips for Building Multi-Touch Enabled Web Sites - *Ben Hoelting*	**W18** - Busy Developer's Guide to MongoDB - *Ted Neward*	**W19** - Real World Workflows with Visual Studio 2012, Workflow Manager and Web Services - *Andrew Connell*	**W20** - TFS vs Team Foundation Service - *Brian Randell*
Lucky Strike Evening Out Party			

Visual Studio Live! Day 3: Thursday, August 22, 2013

Registration - Coffee and Morning Pastries			
TH02 - Building Rich Data HTML Client Apps with Breeze.js - *Brian Noyes*	**TH03** - Building Your First Windows Phone 8 Application - *Brian Peek*	**TH04** - Introducing SQL Server Data Tools - *Leonard Lobel*	**TH05** - Build It So You Can Ship It! - *Brian Randell*
TH07 - Busy Developer's Guide to AngularJS - *Ted Neward*	**TH08** - From Prototype to the Store: How to Truly Finish a Windows Phone App - *Nick Landry*	**TH09** - Big Data-BI Fusion: Microsoft HDInsight & MS BI - *Andrew Brust*	**TH10** - Better Process and Tools with Microsoft ALM - *Brian Randell*
TH12 - Session To Be Announced	**TH13** - Connecting to Data from Windows Phone 8 - *Christopher Woodruff*	**TH14** - Programming the T-SQL Enhancements in SQL Server 2012 - *Leonard Lobel*	**TH15** - Building Great Windows Store Apps with XAML and C# - *Pete Brown*
Lunch @ The Mixer - Visit the Microsoft Company Store & Visitor Center			
TH17 - Get your Node.js Under Control with TypeScript - *Yaniv Rodenski*	**TH18** - Developing Mobile Solutions with Azure and Windows Phone - *Christopher Woodruff*	**TH19** - Getting to Know the BI Semantic Model - *Andrew Brust*	**TH20** - Modern Single Page Applications with HTML 5, CSS3, JS, ASP.NET and Visual Studio
TH22 - SignalRity - *Yaniv Rodenski*	**TH23** - Designing Your Windows Phone Apps for Multitasking and Background Processing - *Nick Landry*	**TH24** - Intro to Windows Azure SQL Database and What's New - *Eric D. Boyd*	**TH25** - Web API 2 – Web Services for Websites, Modern Apps, and Mobile Apps - *Daniel Roth*

Visual Studio Live! Post Conference Workshops: Friday, August 23, 2013 (Separate entry fee required)

Post Conference Workshop Registration - Coffee and Morning Pastries	
Browser Clients with Knockout, JQuery, Breeze, and Web API - *Brian Noyes*	**FW02** - Workshop: SQL Server for Developers - *Andrew Brust & Leonard Lobel*
Lunch	
Browser Clients with Knockout, JQuery, Breeze, and Web API - *Brian Noyes*	**FW02** - Workshop: SQL Server for Developers - *Andrew Brust & Leonard Lobel*

Getting Your App into the Windows Store

Bruno Terkaly

With the right guidance, getting an application into the Windows Store is surprisingly easy. I should know—I've done it myself for two applications (Kids Car Colors and Teach Kids Music) and I've helped others submit hundreds of apps. I believe you can get a quality, useful app into the store in a weekend or two and in this article I'll show you how to do just that in 10 simple steps.

There are many reasons you should consider writing a Windows Store application for Windows 8. To start, with more than 100 million app downloads, the Windows Store is proving to be a viable ecosystem for entrepreneurs and developers. The opportunity to monetize your Windows developer skills has never been greater. One of my colleagues is already making more than $30,000 per month with a card game!

Another good reason to think seriously about writing apps for software marketplaces is that the future of software development is clearly headed in this direction. Not too long ago, companies or individual developers identified or branded themselves by creating a Web site presence. Although that's still the case, the emerging trend is to create and distribute software through Web-based marketplaces, such as the Windows Store.

This software model solves a lot of headaches for budding entrepreneurs, because it minimizes the distracting challenges of customer acquisition, billing and collection, deployment, and installation—to name just a few. Windows 8 and the Windows Store make it simple for millions of customers to find, try and buy high-quality, certified apps from practically anywhere in the world; for the developer, it's easy to distribute, update and get paid for the apps you develop.

Before getting started, I want to mention that this article is focused on Windows 8, not on Windows Phone. Windows Phone has a separate store (Marketplace) and its own separate SDK.

GET HELP BUILDING YOUR WINDOWS STORE APP!

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

bit.ly/XLjOrx

This article discusses:

- The Modern UI design paradigm
- Setting up your development environment
- Monetizing your application
- Creating an app via a template
- Modifying the template-generated code
- Requirements for certification

Technologies discussed:

Windows 8, Visual Studio 2012

10 Simple Steps

My goal in this article is to explain the steps I took to get my two apps into the Windows Store. As you can imagine, I had to learn some new skills, and I want to pass these along to you. Let's take a look at those steps.

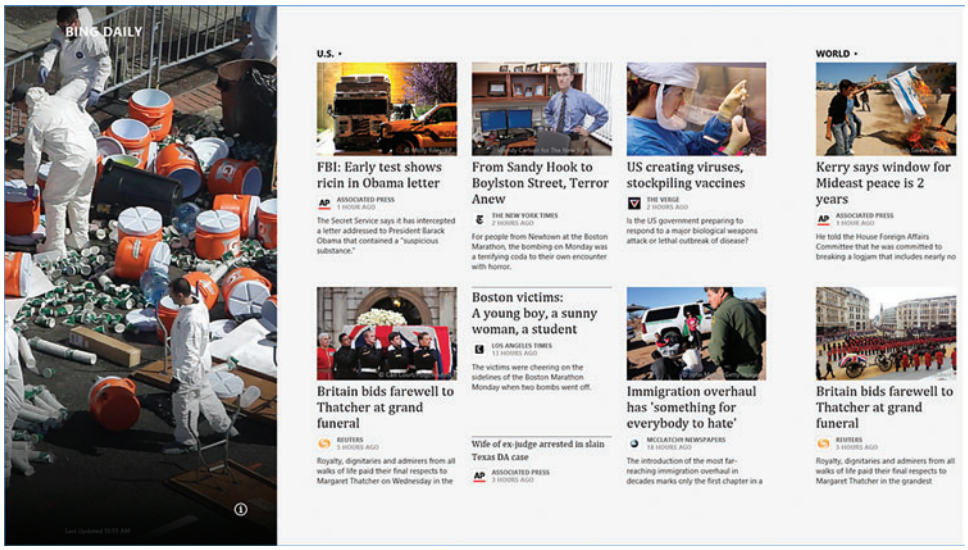


Figure 1 **New Windows UI Design in Practice**

1. Come up with an idea—a game, a productivity or information app, or just about anything appropriate for the format.
2. Download the tooling, SDKs and so forth. You need to be running Windows 8 and you'll need to download and install Visual Studio 2012 Express for Windows Desktop (free), as well as optional SDKs such as the Windows SDK for Windows 8 and the Advertising SDK.
3. Establish a Windows Store Developer Account at the Microsoft Dev Center Portal. This is where you sign up and enter information about your app, such as the name, pricing, in-app purchasing, a description and screenshots for the store (you'll have screenshots once you've completed step 5). You may be eligible for a free Windows Store account.
4. Consider whether your app will display ads. If so, you'll need an account at the Microsoft Advertising pubCenter. You'll get an AdUnitId, which will be used in an AdControl within the application.
5. Create a project in Visual Studio after selecting from various project templates and languages. When development is complete, you'll create an app package (essentially a zipped-up version of the application).
6. Upload your app package to the Dev Center Portal. You already entered information about the app (in step 3). Now you're uploading the finished application package.
7. Complete the remaining work at the Dev Center Portal. After you upload the app package, fill in the remaining two sections—the App Description and the Notes to testers.
8. Now you wait for certification. If the app fails compliance testing, you'll need to fix the issues.
9. Resubmit the app. It's not unusual to fail at certification your first time through. This document can help: bit.ly/rVo1WY.
10. Adjust and enhance the app. An app should be improved over time. Its revenue model may change as well. For example, you might choose to go to a trial model instead of a free model with ads. Huge scale is needed to make money with ads.

Now let's look at some of the details.

Idea Conception You can't really get started until you come up with a good idea. In my case, this was easy. I have a 2-year-old boy who much prefers to learn with a device than with a book. I realized that he loved two things—cars and music. So the goal of my first app was to use cars to teach him colors. With the second app, I wanted to show him the instruments in an orchestra, both in terms of how they look and how they sound.

By a significant margin, games dominate in popularity, both in terms of downloads and in terms of time spent in the application. The next most popular application category is social networking, followed by entertainment and utilities (read the slide deck, "The Future of Mobile," at read.bi/ZGIUV6 for more information). If your goal is to maximize popularity and, hence, monetization, you'll want to consider the type of application in which to invest your resources.

Design Windows 8 represents a major leap forward in UI design. It's fast and fluid to use, and optimized for mobile, touch-enabled form factors such as laptops, tablets and convertibles. The new design paradigm is formally known as Microsoft UI Design Language, also known as Modern UI. It focuses more on beautiful typography and less on fancy graphics, while putting content at the forefront for the end user rather than commands on the chrome. The cornerstone of application development with Modern UI design is a minimalistic approach with balance, symmetry and hierarchy as key pillars, as shown in the Bing News application in **Figure 1**. Forget about drop shadows, transparencies and 3D. All applications that get accepted to the Windows Store must conform to the Windows Store UX Design guidelines (see design.windows.com and bit.ly/R1mlWH for more design guidance).

Great for .NET Developers—and Others Although there's a new design paradigm, .NET developers will feel right at home because you'll use many familiar technologies. Much of the new framework, the Windows Runtime (WinRT), can be thought of as both a subset and superset of the Microsoft .NET Framework. You can access Windows Runtime with a variety of languages, such as C++/Component Extensions (CX), C#, Visual Basic .NET and JavaScript/TypeScript. Windows Runtime is a subset because it supports many, but not all, of the typical .NET elements, which amounts to about 1,800 types. But it's also a superset because it can utilize touch and natively supports sensors, such as a camera, ambient light sensor and accelerometer. C# developers will be happy to know that C# is a first-class citizen, minimizing the need to drop into C++ to gain additional efficiencies. Existing Silverlight and Windows Presentation Foundation (WPF) developers will feel most at home writing programs for Windows Runtime because of

Figure 2 **Links to Get You Started**

Windows 8	Not Free	bit.ly/Sar392
Visual Studio 2012 Express	Free	bit.ly/QLJsJT
Get a Developer License	Free	bit.ly/17SWdpd
Sign up for a Developer Account	$49 for Developers	bit.ly/Y4tTkK
Sign up for Ads in Your App	Free	bit.ly/L6xNAh

Figure 3 **Tutorials for All Developer Types**

Create your first Windows Store app using JavaScript	bit.ly/vVbVHC
Create your first Windows Store app using C# or Visual Basic	bit.ly/KvKxkt
Create your first Windows Store app using C++	bit.ly/11CnUIl
Create your first Windows Store app using DirectX	bit.ly/A5eZIF

the strong emphasis on XAML, the declarative markup language used to build UIs.

That said, C++ offers some strong advantages, in particular that you can leverage Direct2D and Direct3D. Direct2D is a 2D and vector graphics API that provides fast, high-quality graphics performance, taking advantage of hardware acceleration through compatible graphics cards, which frees the main CPU to perform other operations. Direct2D is built on top of Direct3D, which is the graphics API on the Xbox and Xbox 360 console systems. Direct3D also takes advantage of hardware acceleration and is used to render 3D graphics applications, typically games. If you're among the many developers who have existing C++ legacy code, you'll be happy to know it can be easily ported to a Windows Store application.

You'll need to choose the technology that best suits your skills and the requirements of your app. Web developers will typically choose HTML5/JavaScript and leverage the Windows Library for JavaScript (WinJS) APIs, while .NET developers will choose C# or Visual Basic. High-end game developers will choose C++, along with Direct2D/Direct3D. The one common theme across all these languages is the support for XAML.

Getting Set Up Getting your machine ready to produce Windows Store applications is a matter of downloading and installing the right software. Obviously, you'll need a copy of Windows 8 as the base OS. If you already have Windows 7, you'll probably find the upgrade to Windows 8 fairly seamless. Windows 8 still has a desktop that Windows 7 users will find familiar and that can be accessed at any time using the Windows Key+D shortcut.

Once you've installed Windows 8, you need to download developer tooling and sign up for an account at the Windows Store Developer Portal. Visual Studio 2012 is required, and it comes in a free Express version that has everything you need to build a full-featured Windows Store app. You can find a comparison of the various Visual Studio versions at bit.ly/Pzan9Y. **Figure 2** provides some links to get you started.

You can develop and test your Windows Store applications for free by simply signing up for a developer license with your Microsoft Account. If you wish to deploy to the Windows Store, you'll need to sign up for a developer account. Check Microsoft programs such as BizSpark (microsoft.com/bizspark), DreamSpark (bit.ly/QGbma7) and MSDN (bit.ly/2ludR3). If you plan to make money by showing advertisements, you'll need to sign up for at an additional portal known as the Microsoft Advertising pubCenter (see pubcenter.microsoft.com). One of the best and most thorough learning resources can be found on Channel 9 (bit.ly/VP7I00): "Windows Store apps for Absolute Beginners with C#." This 34-part series features video and sample code, and results in a full-featured application.

Learning How Once you set up your development environment, you'll want to learn how to design and develop Windows Store applications. I'll show you some of my personal journey developing Teach Kids Music and provide some follow-up learning resources later in this article. **Figure 3** provides links to some quick tutorials to get you started.

Monetization This is something you'll want to think about right from the beginning. There are many ways to monetize your Windows Store application. The first and most obvious way is to specify a purchase price. The Windows Store Dev Center is where you define the pricing details of your app as it appears in the Windows Store. You can charge anywhere from $1.49 to $999.99. You can even specify a trial period of one day to 30 days. The second way to monetize is through advertising, which works particularly well with game applications, where users spend significant time interacting with the app. As noted previously, to make use of advertising, you'll need to establish an account at the pubCenter. You then add an advertising control to the application by including an AdControl with an AdUnitId, which binds the ad to a specific application.

A growing trend and one of the more interesting ways to monetize your application is through in-app purchases. Similar

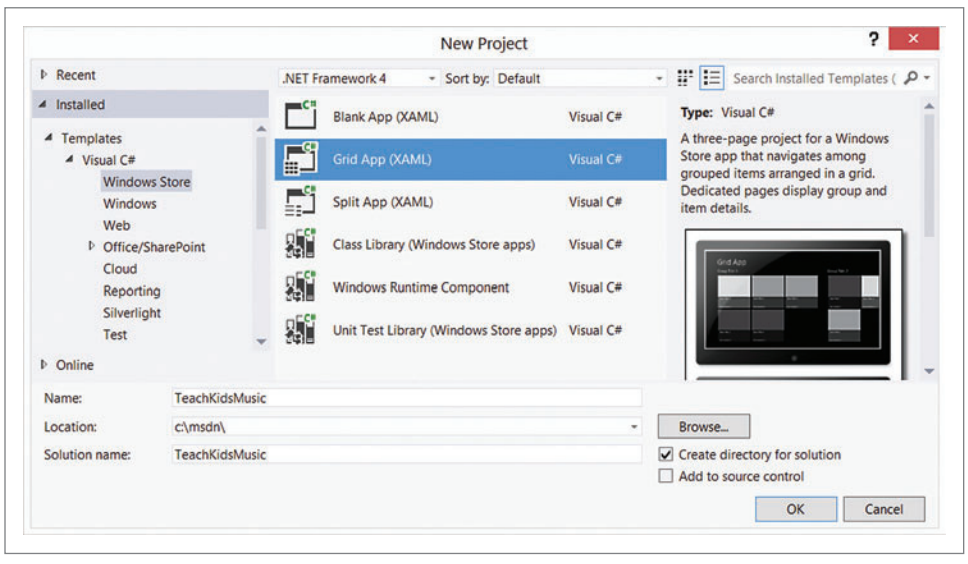


Figure 4 **Selecting the Grid App Template in the New Project Window**

PRECISELY PROGRAMMED FOR SPEED

DynamicPDF—Comprehensive PDF Solutions for .NET Developers

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

DynamicPDF

WWW.DYNAMICPDF.COM

TRY OUR PDF SOLUTIONS FREE TODAY!

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

ceTesoftware

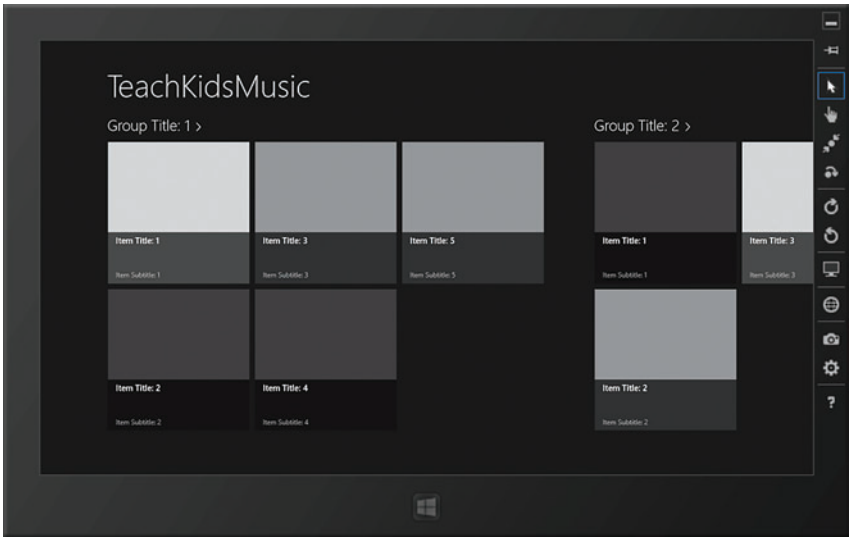


Figure 5 **The Grid App Template Running in the Simulator**

to Web storefronts, some apps are being created as vehicles to sell products. This means that app customers can make purchases directly from within a Windows Store application. For example, Rovio Entertainment Ltd., the creator of Angry Birds, made 45 percent of its 2012 revenue from in-app purchases, selling merchandise (toys, apparel and accessories) based on its popular games. Think about artists or clothing retailers displaying products and making them available for purchase by app users. But while selling merchandise is typical for in-app purchasing, a more common approach is to give away some aspect of an application for free, then offer new content or complementary features for purchase. To support in-app purchases in your application, you'll need to indicate a product ID, which can be obtained from the Dev Center Portal. The product ID isn't seen by customers; it's used as an internal reference to the offer in the app's program code. The Windows Store supports in-app purchases directly through the store or through third-party fulfillment services. Use the CurrentAppSimulator class (bit.ly/RyOImp) for testing and see "Verifying purchases using receipts" (bit.ly/UrK8jA) for more information on programmatically verifying services.

Creating an App

When creating Teach Kids Music, I found the Grid App template, shown in the New Project Window in **Figure 4**, to be quite useful.

The tooling built into Visual Studio 2012 is pretty powerful. A toolbar combo box lets you choose among the local machine (full screen), the simulator or a remote device (such as my Surface RT). The simulator makes it possible to run and test an application without physically deploying it to a device. Many developers are able to deploy an application to the Windows Store without ever physically deploying it to a Surface device. The simulator supports multiple resolutions as well as changing the orientation, and various touch modes.

Figure 5 shows the default Grid App template running in the simulator. Notice it includes sample data that can be changed to support the real data your application will need (which is exactly what I did when I created Teach Kids Music). When you create the Grid App template project, it includes a file called SampleDataSource.cs. This is the file you can edit with your own data.

Notice in **Figure 6** that Teach Kids Music closely resembles the look and feel of the default Grid App template.

The important point is that it's really easy to take one of the out-of-the-box templates and modify the code to create a compelling Windows Store application. This means you don't have to build an application from scratch (though that's an option with the "Blank" template). I built Teach Kids Music as a series of steps. The first step was to add my own data, which can be done by simply modifying SampleDataSource.cs. The parent-child relationship is represented in the code by two main objects, SampleDataItem (child) and SampleDataGroup (parent). You're free to modify these two classes to support the data model needed in your application. Because Teach Kids Music actually plays the sound of each instrument, I added properties in the SampleDataItem class to represent .mp3 files that were added to the project, as shown in **Figure 7**.

Modifying the Template-Generated Code

I made a number of other modifications that were straightforward. First, as you can see in **Figure 6**, I included several images of musical

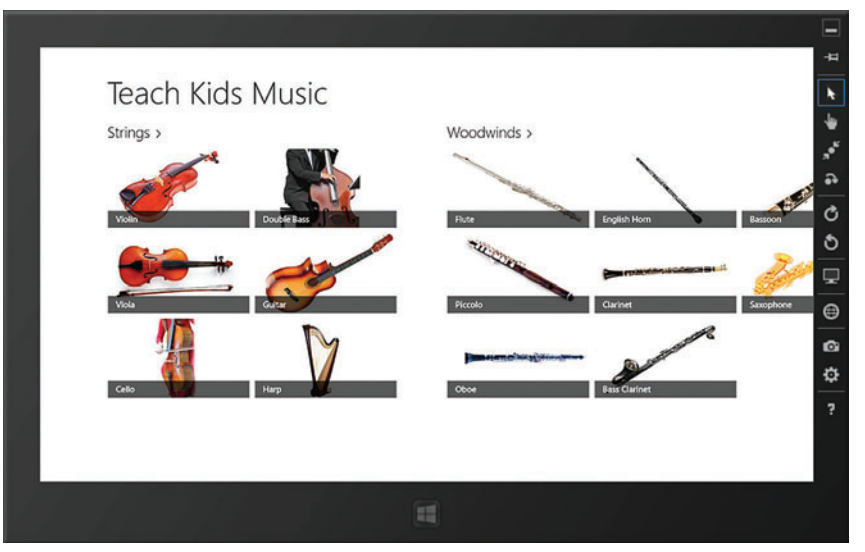


Figure 6 **Teach Kids Music Running in the Simulator**

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

ComponentOne Studio Enterprise | from $1,315.60

ComponentOne
a division of GrapeCity

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

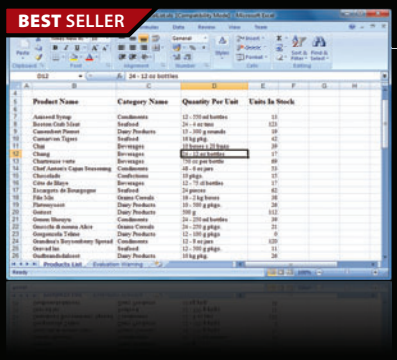
- Hundreds of UI controls for all .NET platforms including grids, charts, reports and schedulers
- Supports Visual Studio 2012 and Windows 8
- Now includes MVC 4 Scaffolding and new MVC 4 project templates (C# & VB)
- New Tile controls for WinForms, WPF, Silverlight, WinRT and Windows Phone
- Royalty-free deployment and distribution

Help & Manual Professional | from $583.10

ec software

Easily create documentation for Windows, the Web and iPad.

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



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

ASPOSE
Your File Format Experts

Every Aspose .NET component in one package.

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

GdPicture.NET | from $3,919.47

GdPicture
imaging technologies

All-in-one AnyCPU document-imaging and PDF toolkit for .NET and ActiveX.

- Document viewing, processing, printing, scanning, OMR, OCR, Barcode Recognition
- Annotate image and PDF within your Windows & Web applications
- Read, write and convert vector & raster images in more than 90 formats, including PDF
- Includes sample code for: .NET, VB6, Delphi, VC++, C++ Builder, VFP, HTML, Access...
- 100% royalty-free and world leading Imaging SDK

© 1996-2013 ComponentSource. All Rights Reserved. All prices correct at the time of press. Online prices may vary from those shown due to daily fluctuations & online discounts.

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

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

European Headquarters
ComponentSource
30 Greyfriars Road
Reading
Berkshire
RG1 1PE
United Kingdom

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

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

GSA Schedule Contract GS-35F-0188R

Figure 7. **Adding Properties in the SampleDataItem Class**

```
SampleDataGroup strings_group = new SampleDataGroup(
  "Strings"
  ,Strings"
    );

strings_group.Items.Add(new SampleDataItem(
  "Violin"
  ,"There are usually 30 violins in an orchestra" + "," +
   " more than any other instrument."
  ,"Images/Viola.jpg"
  ,"Images/ViolaThumb.jpg"
  ,"Sounds/Violin.mp3"
  ,"Images/PlayButtonBig.png"
  ,strings_group));

// And more ...

this.AllGroups.Add(strings_group);
```

instruments. To make the app beautiful, I went into Photoshop and removed the background. I made two sizes of the images, a thumbnail size for the images you see in **Figure 6** and a much bigger full-screen size that appears when a user clicks on a thumbnail. Second, I purchased a number of .mp3 files to represent the sound of each instrument. I used the free audio editor Audacity to enhance the audio and normalize the volume to be consistent across all instruments. I made a "Sounds" folder in the Visual Studio Solution Explorer and copied the .mp3 files to it. Third, I made some stylistic changes to the XAML code. I changed the screen background to white and shortened the description of each instrument to just one line (the name of the instrument). Finally, because I needed to play sound, I made use of the MediaElement control, which can be used to play sounds and video.

Figure 8 **Age Rating Indications for Apps**

Information Collected / Capability Accessed	Provide Access to Your Privacy Policy in the App's Settings as Displayed in the Windows Settings Charm	Can Age Rating Be Lower Than 12+?
SMS	Required	No
Text messages	Required	No
Location	Required	No
Microphone	Required	No
Webcam	Required	No
Documents Library	Required	No
Internet connection (incoming or outgoing)	Not required but encouraged	Yes
Requires a sign-in account where the name on the account is not required to be an e-mail address or user's name (in real life), but any made-up name	Not required but encouraged	Yes
Collects or transmits personal information: user's name, address, account number, e-mail address, phone number, contacts	Not required but encouraged	No
Collects or transmits images of computer desktop or screen shots	Not required but encouraged	No
Collects or transmits browsing history	Not required but encouraged	No

Accelerating Your Development Effort

The built-in Grid App template greatly accelerated my understanding of a number of important developer skills. I already knew a fair amount about C#, including such topics as LINQ, anonymous types, lambdas and so on. And I have a decent understanding of the basics of Visual Studio, such as using the debugger, adding content to a solution, and working with the XAML designers (including Blend) and codebehind. But the real benefit of the Grid App template is that I was able to master some of the more complex topics, such as data binding, change notifications, gridview controls, data templates, type converters, device-orientation changes, snapped state, semantic zoom, share contracts, search contracts, app bars, flyouts, lifetime management (saving and restoring state), saving user preferences, tile management, push notifications, toasts and in-app purchasing. The multipart video series on Channel 9 I mentioned earlier can help you start from ground zero and go all the way to expert.

Certification Requirements

Microsoft specifies a clear set of certification requirements (bit.ly/rVo1VY) for applications that get submitted to the Windows Store. The goal of these requirements is to support a diverse catalog of high-quality, engaging apps for Windows customers worldwide. Every app that gets submitted to the store must undergo significant scrutiny to make sure it conforms to the Windows Store app certification requirements. For example, and perhaps most obvious, Windows Store apps must provide value to the customer. The application must be fully functional. The image on an app's tile "must be reasonably related to the content of the app." Other obvious requirements include the app behaving predictably, without crashes or long launch times (no more than 5 seconds); the Windows App Certification Kit (bit.ly/13CDLnX) will perform automated tests on

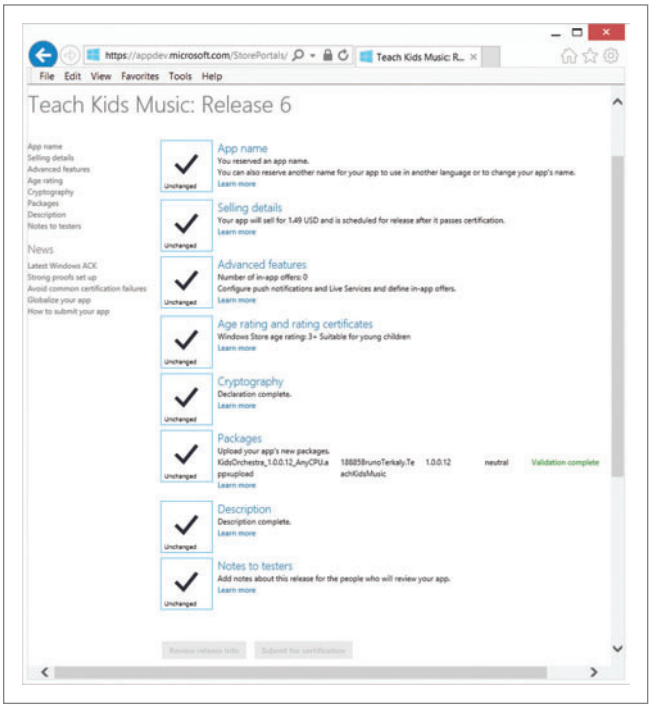


Figure 9 **Filling out App Information at the Windows Store Portal**

nsoftware

We didn't invent the Internet...

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

TOOLS • COMPONENTS • ENTERPRISE ADAPTERS

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

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

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

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

connectivity
powered by

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

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

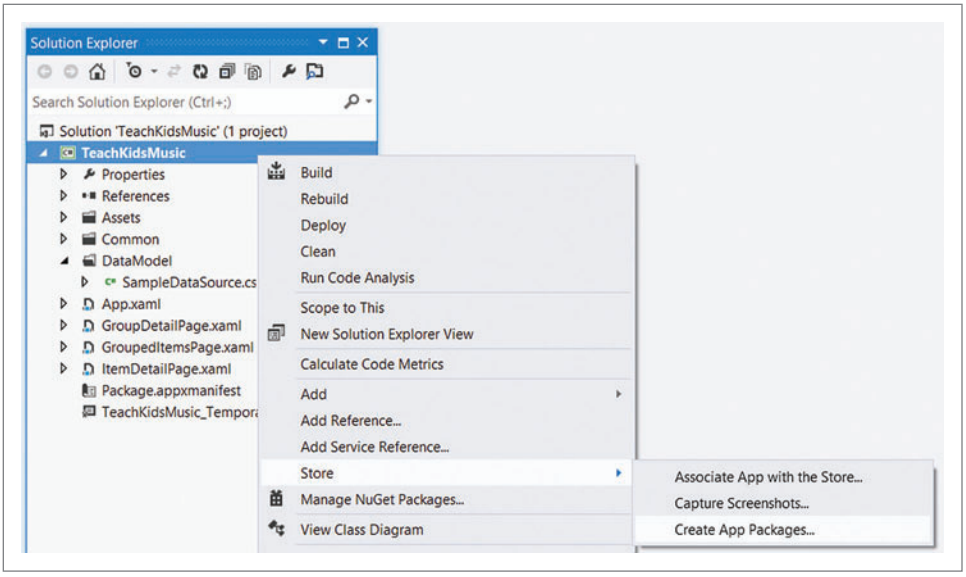


Figure 10 **Creating an App Package in Visual Studio 2012 Solution Explorer**

this. Apps that take more than 5 seconds to load or resume from suspend will be terminated by the runtime broker, so you shouldn't call out to a Web service when suspending and you should optimize your app's initial loading. If you find the initial loading exceeds the allotted time, review the MSDN Splash screen sample at bit.ly/GUrVKn—it has a creative approach to solving initial loading performance issues. Also, remember to test your app in snapped mode, both landscape and portrait modes, and on different resolutions. There's a lot to consider when preparing to release your app globally.

I've noticed two very common reasons for failed Windows 8 certification. The first relates to the app having an appropriate privacy policy (see section 4.1.1 of the requirements). If an app has the technical ability to transmit data, then it must provide access to a privacy policy in both the app's published Description page as well as in the app itself. The policy can be located either in the app's About section or as a separate privacy statement link from the settings flyout available from the Charms menu. Linking to external Web sites doesn't require Internet (client) capability. Many developers choose to have the privacy statement link go to a Web site instead of hosting another page within the app. It's important to note that the Internet (client) capability is turned on by default and this then requires a privacy statement even if your app doesn't use that capability. Capabilities that aren't used shouldn't be checked—review them in your Package.appxmanifest.

The other reason apps frequently fail certification relates to age ratings (see bit.ly/Ta4Rdq). You must assign a Windows Store age rating, indicating for whom the content is suitable. There are various standards bodies that define rating systems, such as the Pan European Game Information (PEGI), which provides guidelines on appropriate levels of violence, sexual content and explicit language. You'll be required to submit documentation from a third party if age rating is required. Your app must target 12+ if it collects or transmits personal information, or if it's an app not specifically designed for young kids (age rating categories 3+ and 7+), or an app that streams or consumes user-generated content. Most apps for those older than 12+ don't require an age rating. To help you determine if your app's age rating can be lower than 12+, look at **Figure 8** (note that this is provided only as guidance; the information may change over time).

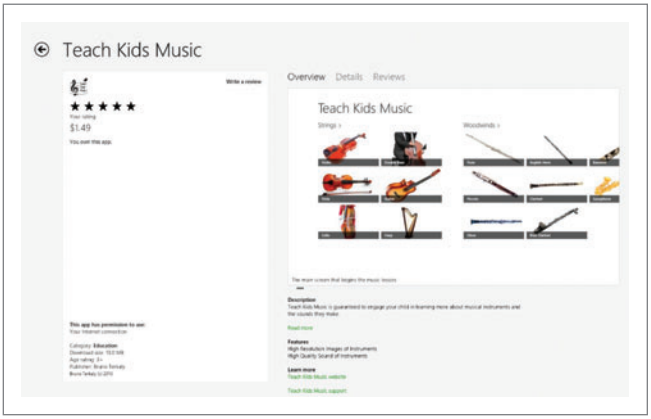


Figure 11 **Viewing an Application from the Windows Store**

Support

You'll want to take advantage of the free, in-person developer support options from Microsoft. Windows Store App Labs (WSAL) offers a place where you can come in and test your app on a variety of different hardware and get authoritative developer advice and assistance from a Windows 8 expert. The labs are located in more than 30 cities worldwide and they've helped with more than 7,000 apps to date. If you can't get to one of the lab locations, virtual options are available for remote review and assistance directly with a Microsoft engineer. Best of all—it's free! See www.windowsstore.com/applabs for more information and to sign up.

Creating an App Package

Once you've conceived, created and tested your Windows Store application, you're ready to create your app package. You no longer need to write code to install or uninstall Windows applications. There's no concept of patch files or setup executables. Instead, Windows Store apps are packaged and submitted. An app package can be thought of as a container or .zip file, with code, data and resources that conform to the Open Packaging Conventions (OPC) standards. In fact, if you rename your generated .appx to .zip, you can open it and look at the contents.

There are a few tasks to accomplish before you can create an app package, as shown in **Figure 9**. You'll need to work your way down the list to Packages before you can upload the package you create. This is where you go back to Visual Studio to physically create the package for upload.

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®

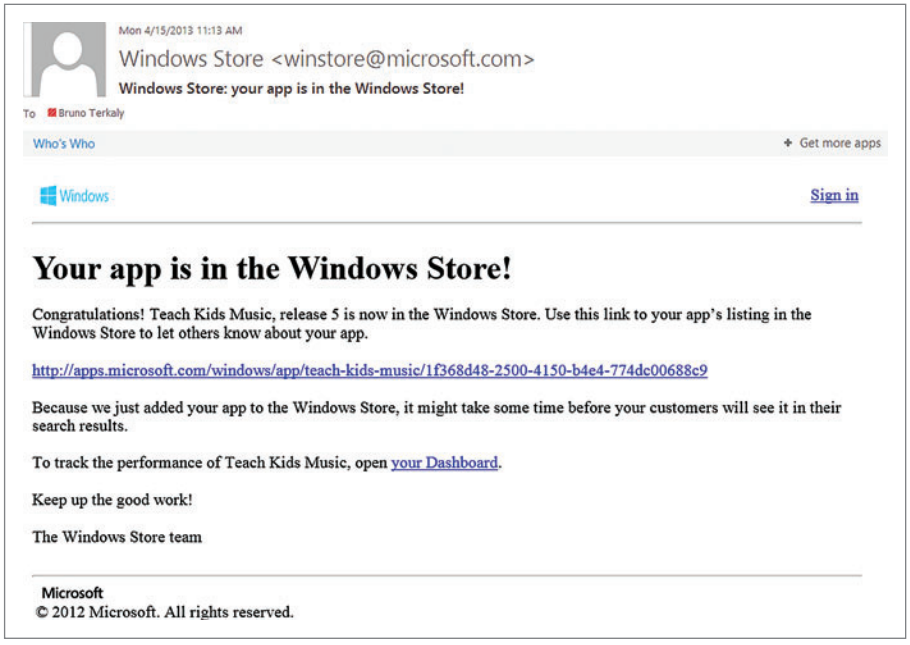
Download your free trial @
componentone.com/ar7

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

As **Figure 10** illustrates, a simple right-click on an app—in this case, the TeachKidsMusic project—can create an app package.

To create the package, you need to log in and associate your application with the Windows Store application you defined previously at the Windows Dev Center Portal. (This assumes you've already established a Windows Store account and have entered information about your application into the Dev Center Portal.) See bit.ly/WHnWq2 for more information on building app packages.

During the process, you'll be asked if you want to use the same technical tests the Windows Store runs during app certification. This means you can run a local copy of the Windows App Certification Kit before uploading a package, and it's very much advised. By running this kit locally you can find and fix any issues early on to increase the likelihood your app will pass technical certification.


Mon 4/15/2013 11:13 AM

Windows Store <winstore@microsoft.com>

Windows Store: your app is in the Windows Store!

To Bruno Terkaly

Who's Who + Get more apps

Windows Sign in

Your app is in the Windows Store!

Congratulations! Teach Kids Music, release 5 is now in the Windows Store. Use this link to your app's listing in the Windows Store to let others know about your app.

http://apps.microsoft.com/windows/app/teach-kids-music/1f368d48-2500-4150-b4e4-774dc00688c9

Because we just added your app to the Windows Store, it might take some time before your customers will see it in their search results.

To track the performance of Teach Kids Music, open your Dashboard.

Keep up the good work!

The Windows Store team

Microsoft
© 2012 Microsoft. All rights reserved.

Figure 12 **Receiving Approval**

The app package will be created in a sub-folder of your main project folder called AppPackages. The file you upload will be called something similar to YOURAPPNAME_1.0.0.1_AnyCPU.appxupload.

App Submission

If you look back at **Figure 9**, you'll see there are just a few remaining steps before your app submission is complete—the Description and Notes to testers sections need to be filled in. The Description section is very important because it's what a potential customer will see at the Windows Store (see **Figure 11**). You need to create a screenshot after running the app in the simulator, as discussed previously. The simulator has a built-in Copy Screenshot button. You save this image as a .png file and upload it to the Dev Center Portal.

The Description section includes some information critical to app certification—I mentioned earlier how problems with the privacy policy information are responsible for a great many app failures. You also use the Description section to include an optional Web site, e-mail address and promotional images (should the Windows Store choose to showcase your application).

You'll find a very helpful walk-through and discussion at the Microsoft Windows Store blog at bit.ly/zFZNAd.

The Notes to testers section is visible to testers only during the certification process, to indicate features or behavior that may not be immediately discoverable. For example, if the app must log into a service or get a username or password, you should include those notes to Microsoft testers. These notes can also be useful if an app uses background audio. In such cases, you should provide a sample that lets testers verify the audio file, making sure the sample takes less than a minute for a single tester to reproduce.

The final step in the entire process is clicking on the Submit for certification button, which becomes enabled only when you've completed all the sections and uploaded the app package at the Dev Center Portal. This process can take up to a few days to complete. If your app passes certification, you'll see the results shown in **Figure 12**. If it doesn't pass certification, you'll get a report telling you why. It's generally a simple matter of fixing the problem and resubmitting the app.

If the reasons for your app's failure are unclear, go to dev.windows.com, click Support and scroll to Help with your developer account, then indicate the problem type and category and choose the type of support you need.

It takes just 10 simple steps to get your app into the Windows Store. But that doesn't mean your work is over. Be sure to check your Developer Portal frequently for any error reports and to review your customer rating and feedback. Successful apps evolve over time and develop richer feature sets. Because there are many monetization strategies, many developers will continue to adjust their revenue model.

Successful apps also continue to drive awareness through a Web site or through social marketing. You can find the link to the Web portal for your app in your app details and link to that from various social media hubs. Also, if you have a Web site associated with your app, you can have Internet Explorer 10 show a "Get the app" pop-up to allow users to directly install your app (see bit.ly/AcEc1J for more information). ■

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. He has published two applications to the Windows Store: Teach Kids Music and Kids Car Colors. You can read his blog at blogs.msdn.com/b/brunoterkaly.*

Thanks to the following technical expert for reviewing this article: *Robert Evans (Microsoft)*

Spread Studio for .NET is our new cross-platform toolkit with controls for Windows Forms, ASP.NET, WPF, WinRT, and Silverlight in one amazing package.

Experience for yourself:

- 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

NEW VERSION 7

Spread Studio for .NET

ComponentOne®
a division of GrapeCity®

Download your free trial @
componentone.com/spread7

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

Enabling and Customizing ASP.NET Web API Services Security

Peter Vogel

For the most common scenario—JavaScript in a Web page accessing a Web API service on the same site—discussing security for ASP.NET Web API is almost redundant. Provided that you authenticate your users and authorize access to the Web Forms/Views holding the JavaScript that consumes your services, you've probably provided all the security your services need. This is a result of ASP.NET sending the cookies and authentication information it uses to validate page requests as part of any client-side JavaScript requests to your service methods. There's one exception (and it's an important one): ASP.NET doesn't automatically protect you against Cross-Site Request Forgery (CSRF/XSRF) attacks (more on that later).

In addition to CSRF, there are two scenarios when it does make sense to discuss securing your Web API services. The first scenario is when your service is consumed by a client other than a page in the same site as your ApiControllers. Those clients wouldn't have been authenticated through Forms Authentication and wouldn't have acquired the cookies and tokens that ASP.NET uses to control access to your services.

The second scenario occurs when you wish to add additional authorization to your services beyond what's provided through ASP.NET security. The default authorization ASP.NET provides is based on the identity ASP.NET assigns to the request during authentication. You might wish to extend that identity to authorize access based on something other than the identity's name or role.

Web API gives you a number of choices to address both scenarios. In fact, while I'll discuss security in the context of accepting Web API requests, because the Web API is based on the same ASP.NET foundation as Web Forms and MVC, the tools that I'll cover in this article are going to be familiar to anyone who has gone under the hood with security in Web Forms or MVC.

One caveat: While the Web API provides you with several choices for authentication and authorization, security begins with the host, either IIS or a host that you create when self hosting. If, for example, you want to ensure privacy in the communication between a Web API service and the client, then you should, at the very least, turn on SSL. This, however, is a responsibility of the site administrator, rather than the developer. In this article I'm going to ignore the host to concentrate on what a developer can—and should—do to secure a Web API service (and the tools that I'll discuss here work if SSL is turned on or off).

This article discusses:

- Preventing Cross-Site Request Forgery attacks
- Using Basic Authentication
- Consuming from a non-JavaScript client
- Working with client certificates
- Securing requests in a self-hosted service
- Creating a custom Web API message handler
- Extending authorization

Technologies discussed:

ASP.NET Web API

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!

5 Days of 65 Sessions and Workshops!

VISUAL STUDIO LIVE! REDMOND | AUGUST 19-23
MICROSOFT CAMPUS | vslive.com/redmond

Flip over for more details

Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS

REDMOND, WA | AUGUST 19-23, 2013
MICROSOFT CAMPUS

20 LIVE! CELEBRATING TWENTY YEARS

PRACTICAL & UNBIASED EDUCATION FOR DEVELOPERS:

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

Full agenda details available on vslive.com/redmond

REGISTER TODAY AND SAVE $400

Use promo code REDJUNTI before June 12

Scan the QR Code to Register and Save $400 Today!

vslive.com/redmond

EVENT SPONSOR Microsoft

PLATINUM SPONSOR esri

SUPPORTED BY Visual Studio | msdn magazine | Visual Studio MAGAZINE

PRODUCED BY 1105 MEDIA

Preventing Cross-Site Request Forgery Attacks

When a user accesses an ASP.NET site using Forms Authentication, ASP.NET generates a cookie that stipulates the user is authenticated. The browser will continue to send that cookie on every subsequent request to the site, no matter from where that request originates. This opens your site to CSRF attacks, as does any authentication scheme where the browser automatically sends authentication information previously received. If, after your site provides the browser with the security cookie, the user visits some malicious site, then that site can send requests to your service, piggy-backing on the authentication cookie the browser received earlier.

To prevent CSRF attacks, you'll need to generate antiforgery tokens at the server and embed them in the page to be used in your client-side calls. Microsoft provides the AntiForgery class with a GetToken method that will generate tokens specific to the user who made the request (who may, of course, be the anonymous user). This code generates the two tokens and puts them in the ASP.NET MVC ViewBag where they can be used in the View:

```
[Authorize(Roles="manager")]
public ActionResult Index()
{
  string cookieToken;
  string formToken;
  AntiForgery.GetTokens(null, out cookieToken, out formToken);
  ViewBag.cookieToken = cookieToken;
  ViewBag.formToken = formToken;
  return View("Index");
}
```

Any JavaScript calls to the server will need to return the tokens as part of the request (a CSRF site won't have these tokens and won't be able to return them). This code, in a View, dynamically generates a JavaScript call that adds the tokens to the request's headers:

```
$.ajax("http://phvis.com/api/Customers",{
  type: "get",
  contentType: "application/json",
  headers: {
    'formToken': '@ViewBag.formToken',
    'cookieToken': '@ViewBag.cookieToken' }});
```

A slightly more complex solution would let you use unobtrusive JavaScript by embedding the tokens in hidden fields in the View. The first step in that process would be to add the tokens to the ViewData dictionary:

```
ViewData["cookieToken"] = cookieToken;
ViewData["formToken"] = formToken;
```

Now, in the View, you can embed the data in hidden fields. The HtmlHelper's Hidden method just needs to be passed the value of a key in ViewDate to generate the right input tag:

```
@Html.Hidden("formToken")
```

The resulting input tag will use the ViewData key for the tag's name and id attributes and put the data retrieved from the ViewData dictionary into the tag's value attribute. The input tag generated from the previous code would look like this:

```
<input id="formToken" name="formToken" type="hidden" value="...token..." />
```

Your JavaScript code (kept in a separate file from the View) can then retrieve the values from the input tags and use them in your ajax call:

```
$.ajax("http://localhost:49226/api/Customers", {
  type: "get",
  contentType: "application/json",
  headers: {
    'formToken': $("#formToken").val(),
    'cookieToken': $("#cookieToken").val()}});
```

Figure 1 **Validating CSRF Tokens in a Service Method**

```
public HttpResponseMessage Get(){
  if (Request.Headers.TryGetValues("cookieToken", out tokens))
  {
    string cookieToken = tokens.First();
    Request.Headers.TryGetValues("formToken", out tokens);
    string formToken = tokens.First();
    AntiForgery.Validate(cookieToken, formToken);
  }
  else
  {
    HttpResponseMessage hrm =
      new HttpResponseMessage(HttpStatusCode.Unauthorized);
    hrm.ReasonPhrase = "CSRF tokens not found";
    return hrm;
  }
  // ... Code to process request ...
```

You can achieve the same goals in an ASP.NET Web Forms site by using the RegisterClientScriptBlock method on the ClientScriptManager object (retrievable from the Page's ClientScript property) to insert JavaScript code with the embedded tokens:

```
string CodeString = "function CallService(){" +
  "$.ajax('http://phvis.com/api/Customers',{" +
  "type: 'get', contentType: 'application/json'," +
  "headers: {'formToken': '" & formToken & "'," +
  "'cookieToken': '" & cookieToken & "'}});}"

this.ClientScript.RegisterClientScriptBlock(
  typeOf(this), "loadCustid", CodeString, true);
```

Finally, you'll need to validate the tokens at the server when they're returned by the JavaScript call. Visual Studio 2012 users who have applied the ASP.NET and Web Tools 2012.2 update will find the new Single-Page Application (SPA) template includes a ValidateHttpAntiForgeryToken filter that can be used on Web API methods. In the absence of that filter, you'll need to retrieve the tokens and pass them to the AntiForgery class's Validate method (the Validate method will throw an exception if the tokens aren't valid or were generated for a different user). The code in **Figure 1**, used in a Web API service method, retrieves the tokens from the headers and validates them.

Using the ValidateHttpAntiForgeryToken (rather than code inside the method) moves processing to earlier in the cycle (before, for example, model binding), which is a good thing.

Why No OAuth?

This article studiously ignores OAuth. The OAuth specification defines how tokens can be retrieved by a client from a third-party server to be sent to a service that will, in turn, validate the token with the third-party server. A discussion of how to access an OAuth token provider either from the client or the service is beyond the scope for this article.

The initial version of OAuth also isn't a good match for the Web API. Presumably, one of the primary reasons for using the Web API is to use lighter-weight requests based on REST and JSON. That goal makes the first version of OAuth an unattractive option for Web API services. The tokens specified by the first version of OAuth are bulky and XML-based. Fortunately, OAuth 2.0 introduced a specification for a lighter-weight JSON token that's more compact than the token from previous versions. Presumably, the techniques discussed in this article could be used to process any OAuth tokens sent to your service.

Basic Authentication

The first of the two primary responsibilities you have in securing a Web API service is authentication (the other responsibility being authorization). I'll assume other issues—privacy, for example—are handled at the host.

Ideally, both authentication and authorization will be performed as early as possible in the Web API pipeline to avoid spending processing cycles on a request you intend to deny. This article's authentication solutions are used very early in the pipeline—virtually as soon as the request is received. These techniques also allow you to integrate authentication with whatever user lists you're already maintaining. The authorization techniques discussed can be applied in a variety of places in the pipeline (including as late as in the service method itself) and can work with authentication to authorize requests based on some other criteria than the user's name or role.

You can support clients who haven't gone through the Forms Authentication by providing your own authentication method in a custom HTTP module (I'm still assuming here that you're not authenticating against Windows accounts but against your own list of valid users). There are two major benefits to using an HTTP module: a module participates in HTTP logging and auditing; also, modules are invoked very early in the pipeline. While these are both good things, modules do come with two costs: modules are global and are applied to all requests to the site, not just the Web API requests; also, to use authentication modules, you must host your service in IIS. Later in this article, I'll discuss using delegating handlers that are invoked only for Web API requests and are host-agnostic.

For this example in using an HTTP module, I assume that IIS is using Basic Authentication and the credentials used to authenticate a user are a username and password, sent by the client (in this article, I'll ignore Windows certification but will discuss using client certificates). I also assume that the Web API service that I'm protecting is secured using an Authorize attribute such as this, which specifies a user:

```
public class CustomersController : ApiController
{
  [Authorize(Users="Peter")]
  public Customer Get()
  {
```

The first step in creating a custom authorization HTTP module is to add a class to your service project that implements the IHttpModule and IDisposable interfaces. In the class's Init method you'll need to wire up two events from the HttpApplication object passed to the method. The method you attach to the AuthenticateRequest event will be called when the client's credentials are presented. But you must also wire up the EndRequest method in order to generate the message that causes the client to send you its credentials. You'll also need a Dispose method, but you don't need to put anything in it to support the code used here:

```
public class PHVHttpAuthentication : IHttpModule, IDisposable
{
  public void Init(HttpApplication context)
  {
    context.AuthenticateRequest += AuthenticateRequests;
    context.EndRequest += TriggerCredentials;
  }
  public void Dispose()
  {

  }
```

An HttpClient will send credentials in response to a WWW-Authenticate header that you include in the HTTP response. You should include that header when a request generates a 401 status code (ASP.NET will generate a 401 response code when the client is denied access to a secured service). The header must provide a hint as to the authentication method being used and the realm in which the authentication will apply (the realm can be any arbitrary string and is used to flag to the browser different areas on the server). The code to send that message is what you put in the method wired to the EndRequest event. This example generates a message that specifies that Basic authentication is being used within the PHVIS realm:

```
private static void TriggerCredentials(object sender, EventArgs e)
{
  HttpResponse resp = HttpContext.Current.Response;
  if (resp.StatusCode == 401)
  {
    resp.Headers.Add("WWW-Authenticate", @"Basic realm='PHVIS'");
  }
}
```

Within the method you've wired up to the AuthenticateRequest method, you'll need to retrieve the Authorization headers the client will send as a result of receiving your 401/WWW-Authenticate message:

```
private static void AuthenticateRequests(object sender,
  EventArgs e)
{
  string authHeader =
    HttpContext.Current. Request.Headers["Authorization"];
  if (authHeader != null)
  {
```

Once you've determined the client has passed Authorization header elements (and continuing with my earlier assumption that the site is using Basic Authentication), you need to parse out the data holding the username and password. The username and password are Base64-encoded and separated by a colon. This code retrieves the username and password into a two-position string array:

```
AuthenticationHeaderValue authHeaderVal =
  AuthenticationHeaderValue.Parse(authHeader);
if (authHeaderVal.Parameter != null)
{
  byte[] unencoded = Convert.FromBase64String(
    authHeaderVal.Parameter);
  string userpw =
    Encoding.GetEncoding("iso-8859-1").GetString(unencoded);
  string[] creds = userpw.Split(':');
```

As this code demonstrates, usernames and passwords are sent in clear-text. If you don't turn on SSL then your usernames and passwords can be easily captured (and this code works even if SSL is turned on).

The next step is to validate the username and password using whatever mechanism makes sense to you. Regardless of how you validate the request (the code I use in the following example is probably too simple), your final step is to create an identity for the user that will be used in the authorization processes later in the ASP.NET pipeline.

To pass that identity information through the pipeline, you create a GenericIdentity object with the name of the identity you want to assign to the user (in the following code I've assumed that identity is the username sent in the header). Once you've created the GenericIdentity object, you must put it in the Thread class's CurrentPrincipal property. ASP.NET also maintains a second security context in the HttpContext object and, if your host is IIS, you must support that by also setting the User property in the HttpContext's Current property to your GenericIdentity object:

Did we mention our remarkable human support?

developer SDKs to capture, view & annotate your documents

Atalasoft
A Kofax Company

.NET & Java Web Scanning & Imaging SDKs
www.atalasoft.com
1.413.572.4443

```
if (creds[0] == "Peter" && creds[1] == "pw")
{
  GenericIdentity gi = new GenericIdentity(creds[0]);
  Thread.CurrentPrincipal = new GenericPrincipal(gi, null);
  HttpContext.Current.User = Thread.CurrentPrincipal;
}
```

If you want to support role-based security, then you must pass an array of role names as the second parameter to the GenericPrincipal constructor. This example assigns every user to the manager and admin roles:

```
string[] roles = "manager,admin".Split(',');
Thread.CurrentPrincipal = new GenericPrincipal(gi, roles);
```

To integrate your HTTP module into your site's processing, in your project's web.config file, use the add tag within the modules element. The add tag's type attribute must be set to a string consisting of the fully qualified class name followed by the assembly name of your module:

```
<modules>
  <add name="myCustomerAuth"
    type="SecureWebAPI.PHVHttpAuthentication, SecureWebAPI"/>
</modules>
```

The GenericIdentity object you created will work with the ASP.NET Authorize attribute. You can also access the GenericIdentity from inside a service method to perform authorization activities. You could, for example, provide different services for logged-in and anonymous users by determining if a user has been authenticated by checking the GenericIdentity object IsAuthenticated property (IsAuthenticated returns false for the Anonymous user):

```
if (Thread.CurrentPrincipal.Identity.IsAuthenticated)
{
```

You can retrieve the GenericIdentity object more simply through the User property:

```
if (User.Identity.IsAuthenticated)
{
```

Building a Compatible Client

In order to consume services protected by this module, a non-JavaScript client must provide an acceptable username and password. To provide those credentials using the .NET HttpClient, you first create an HttpClientHandler object and set its Credentials property to a NetworkCredential object holding the username and password (or set the HttpClientHandler object's UseDefaultCredentials property to true in order to use the current user's Windows credentials). You then create your HttpClient object, passing the HttpClientHandler object:

```
HttpClientHandler hch = new HttpClientHandler();
hch.Credentials = new NetworkCredential ("Peter", "pw");
HttpClient hc = new HttpClient(hch);
```

With that configuration done, you can issue your request to the service. The HttpClient won't present the credentials until it's denied access to the service and has received the WWW-Authenticate message. If the credentials provided by the HttpClient aren't acceptable, the service returns an HttpResponseMessage with the StatusCode of its Result set to "unauthenticated."

The following code calls a service using the GetAsync method, checks for a successful result and (if it doesn't get one) displays the status code returned from the service:

```
hc.GetAsync("http://phvis.com/api/Customers").ContinueWith(r =>
{
  HttpResponseMessage hrm = r.Result;
  if (hrm.IsSuccessStatusCode)
  {
    // ... Process response ...
  }
  else
  {
    MessageBox.Show(hrm.StatusCode.ToString());
  }
});
```

Assuming that you bypass the ASP.NET login process for non-JavaScript clients, as I did here, no authentication cookies will be created and each request from the client will be validated individually. To reduce the overhead on repeatedly validating the credentials provided by the client, you should consider caching credentials you retrieve at the service (and using your Dispose method to discard those cached credentials).

Working with Client Certificates

In an HTTP module, you retrieve a client certificate object (and ensure that it's present and valid) with code such as this:

```
System.Web.HttpClientCertificate cert =
  HttpContext.Current.Request.ClientCertificate;
if (cert!= null && cert.IsPresent && cert.IsValid)
{
```

Further along in the processing pipeline—in a service method, for example—you retrieve the certificate object (and check that one exists) with this code:

```
X509Certificate2 cert = Request.GetClientCertificate();
if (cert!= null)
{
```

If a certificate is valid and present, you can additionally check for specific values in the certificate's properties (for example, subject or issuer).

To send certificates with an HttpClient, your first step is to create a WebRequestHandler object instead of an HttpClientHandler (the WebRequestHandler offers more configuration options than the HttpClientHandler):

```
WebRequestHandler wrh = new WebRequestHandler();
```

You can have the HttpClient automatically search the client's certificate stores by setting the WebRequestHandler object's ClientCertificateOptions to the Automatic value from the ClientCertificateOption enum:

```
wrh.ClientCertificateOptions = ClientCertificateOption.Manual;
```

Figure 2 **Creating a Custom Principal with Additional Properties**

```
public class PHVPrincipal: IPrincipal
{
  public PHVPrincipal(string Name, string Region)
  {
    this.Name = Name;
    this.Region = Region;
  }
  public string Name { get; set; }
  public string Region { get; set; }

  public IIdentity Identity
  {
    get
    {
      return new GenericIdentity(this.Name);
    }
    set
    {
      this.Name = value.Name;
    }
   }

   public bool IsInRole(string role)
   {
     return true;
   }
```

OPEN YOUR WINDOWS TO EXTERNAL DATA

It used to be that data was largely an internal matter. Today though, we have the opportunity to harness a wealth of external data to shape new revenue streams and build customer insight. **FATDB** is a NoSQL database for Windows that is ideal for capturing unstructured, external data. FatDB allows you to build compelling applications combining data from both FatDB and SQL Server.

To find out more and download FatDB visit **FATCLOUD.COM**.

FAT

By default, however, the client must explicitly attach certificates to the WebRequestHandler from code. You can retrieve the certificate from one of the client's certificate stores as this example does, which retrieves a certificate from the CurrentUser's store using the issuer's name:

```
X509Store certStore;
X509Certificate x509cert;
certStore = new X509Store(StoreName.My, StoreLocation.CurrentUser);
certStore.Open(OpenFlags.OpenExistingOnly | OpenFlags.ReadOnly);
x509cert = certStore.Certificates.Find(
  X509FindType.FindByIssuerName, "PHVIS", true)[0];
store.Close();
```

If the user has been sent a client certificate that, for some reason, isn't going to be added to the user's certificate store, then you can create an X509Certificate object from the certificate's file with code like this:

```
x509cert = new X509Certificate2(@"C:\PHVIS.pfx");
```

Regardless of how the X509Certificate is created, the final steps at the client are to add the certificate to the WebRequestHandler ClientCertificates collection and then use the configured WebRequestHandler to create the HttpClient:

```
wrh.ClientCertificates.Add(x509cert);
hc = new HttpClient(wrh);
```

Authorizing in a Self-Hosted Environment

While you can't use an HttpModule in a self-hosted environment, the process for securing requests early in the processing pipeline of a self-hosted service is the same: Get the credentials from the request, use that information to authenticate the request and create an identity to pass to the current thread's CurrentPrincipal property. The simplest mechanism is to create a username and password validator. To do more than just validate a username and password combination, you can create a delegating handler. I'll first look at integrating a username and password validator.

To create a validator (still assuming that you're using Basic Authentication), you must create a class that inherits from UserNamePasswordValidator (you'll need to add a reference to the System.IdentityModel library to your project). The only method from the base class that you need to override is the Validate method, which will be passed the username and password sent to the service by the client. As before, once you've validated the username and password, you must create a GenericPrincipal object and use it to set the CurrentPrincipal property on the Thread class (because you're not using IIS as your host, you don't set the HttpContext User property):

```
public class PHVValidator :
  System.IdentityModel.Selectors.UserNamePasswordValidator
{
  public override void Validate(string userName, string password)
  {
    if (userName == "Peter" && password == "pw")
    {
      GenericIdentity gi = new GenericIdentity(username, null);
      Thread.CurrentPrincipal = gi;
    }
```

The following code creates a host for a controller called Customers with an endpoint of http://phvis.com/MyServices, and specifies a new validator:

```
partial class PHVService : ServiceBase
{
  private HttpSelfHostServer shs;
  protected override void OnStart(string[] args)
  {
    HttpSelfHostConfiguration hcfg =
      new HttpSelfHostConfiguration("http://phvis.com/MyServices");
    hcfg.Routes.MapHttpRoute("CustomerServiceRoute",
      "Customers", new { controller = "Customers" });
    hcfg.UserNamePasswordValidator = new PHVValidator;
    shs = new HttpSelfHostServer(hcfg);
    shs.OpenAsync();
```

Message Handlers

To do more than validate the username and password, you can create a custom Web API message handler. Message handlers have several benefits compared to an HTTP module: message handlers aren't tied to IIS, so security applied in a message handler will work with any host; message handlers are only used by the Web API, so they provide a simple way to perform authorization (and assign identities) for your services using a process different from that used with your Web site pages; and you can also assign message handlers to specific routes so that your security code is only invoked where it's needed.

The first step in creating a message handler is to write a class that inherits from DelegatingHandler and override its SendAysnc method:

```
public class PHVAuthorizingMessageHandler: DelegatingHandler
{
  protected override System.Threading.Tasks.Task<HttpResponseMessage>
    SendAsync(HttpRequestMessage request,
      System.Threading.CancellationToken cancellationToken)
  {
```

Within that method (and assuming that you're creating a per-route handler) you can set the DelegatingHandler's InnerHandler property so that this handler can be linked into the pipeline with other handlers:

```
HttpConfiguration hcon = request.GetConfiguration();
InnerHandler = new HttpControllerDispatcher(hcon);
```

For this example, I'm going to assume that a valid request must have a simple token in its querystring (quite simple: a name/value pair of "authToken=xyx"). If the token is missing or not set to xyx, the code returns a 403 (Forbidden) status code.

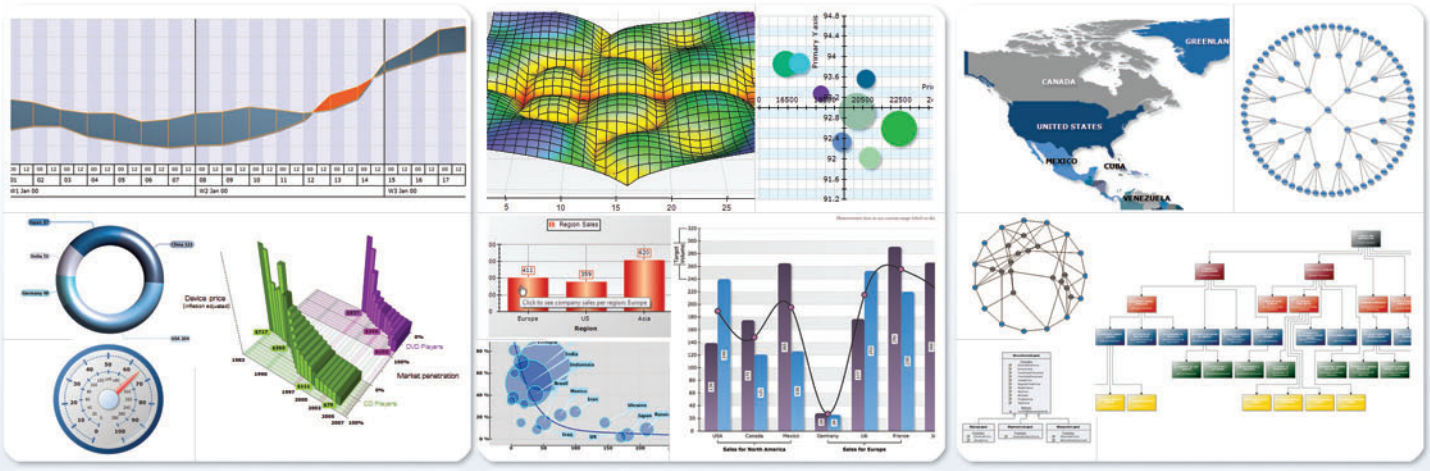
I first turn the querystring into a set of name/value pairs by calling the GetQueryNameValuePairs method on the HttpRequestMessage object passed to the method. I then use LINQ to retrieve the token (or null if the token is missing). If the token is missing or invalid, I create an HttpResponseMessage with the appropriate HTTP status code, wrap it in a TaskCompletionSource object and return it:

Figure 3 **Filtering a Custom Principal Object**

```
public class RegionAuthorizeAttribute : System.Web.Http.AuthorizeAttribute
{
  public string Region { get; set; }
  protected override bool IsAuthorized(HttpActionContext actionContext)
  {
    PHVPrincipal phvPcp = Thread.CurrentPrincipal as PHVPrincipal;
    if (phvPcp != null && phvPcp.Region == this.Region)
    {
      return true;
    }
    else
    {
      actionContext.Response =
        new HttpResponseMessage(
          System.Net.HttpStatusCode.Unauthorized)
        {
          ReasonPhrase = "Invalid region"
        };
      return false;
    }
  }
}
```

Nevron Data Visualization

The leading data visualization components for a wide range of .NET platforms.
14+ years of refinement, complete feature sets, highly customizable design and great support.



Nevron Vision for .NET

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

Nevron Vision for SharePoint

The leading data visualization web parts for SharePoint 2007, 2010 and 2013.
Helps you convert your SharePoint pages into interactive dashboards and reports.

Nevron Vision for SSRS

The leading data visualization report items for SSRS 2005, 2008, 2008R2 and 2012.
Helps you deliver deeper data insights with more engaging looks.

Nevron components integrate seamlessly in Web and Desktop .NET applications, SQL Server Reporting Services 2005/2008/2008R2 and 2012 reports and SharePoint 2007/2010/2013 portals and deliver an unmatched set of enterprise-grade features. That is why Nevron is the trusted vendor by many Fortune 500 companies for their most demanding data visualization needs.

Make sure that your data is making the visual statement
it deserves by downloading your free evaluation copy
from www.nevron.com today.

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

```
string usingRegion = (from kvp in request.GetQueryNameValuePairs()
                      where kvp.Key == "authToken"
                      select kvp.Value).FirstOrDefault();
if (usingRegion == null || usingRegion != "xyx")
{
  HttpResponseMessage resp =
    new HttpResponseMessage(HttpStatusCode.Forbidden);
  TaskCompletionSource tsc =
    new TaskCompletionSource<HttpResponseMessage>();
  tsc.SetResult(resp);
  return tsc.Task;
}
```

If the token is present and set to the right value, I create a GenericPrincipal object and use it to set the Thread's CurrentPrincipal property (to support using this message handler under IIS, I also set the HttpContext User property if the HttpContext object isn't null):

```
Thread.CurrentPrincipal = new GenericPrincipal(
  Thread.CurrentPrincipal.Identity.Name, null);
if (HttpContext.Current != null)
{
  HttpContext.Current.User = Thread.CurrentPrincipal;
}
```

With the request authenticated through the token and the identity set, the message handler calls the base method to continue processing:

```
return base.SendAsync(request, cancellationToken);
```

If your message handler is to be used on every controller, you can add it to the Web API processing pipeline like any other message handler. However, to limit your handler to being used only on specific routes, you must add it through the MapHttpRoute method. First, instantiate your class and then pass it as the fifth parameter to MapHttpRoute (this code requires an Imports/using statement for System.Web.Http):

```
routes.MapHttpRoute(
  "ServiceDefault",
  "api/Customers/{id}",
  new { id = RouteParameter.Optional },
  null,
  new PHVAuthorizingMessageHandler());
```

Rather than set the InnerHandler within the DelegatingHandler, you can set the InnerHandler property to the default dispatcher as part of defining your route:

```
routes.MapHttpRoute(
  "ServiceDefault",
  "api/{controller}/{id}",
  new { id = RouteParameter.Optional },
  null,
  new PHVAuthorizingMessageHandler
  {InnerHandler = new HttpControllerDispatcher(
    GlobalConfiguration.Configuration)});
```

Now, instead of your InnerHandler setting being spread among multiple DelegatingHandlers, you're managing it from the single location where you define your routes.

Extending the Principal

If authorizing requests by name and role isn't sufficient, you can extend the authorization process by creating your own principal class by implementing the IPrincipal interface. However, to take advantage of a custom principal class, you'll need to create your own custom authorization attribute or add custom code to your service methods.

For example, if you have a set of services that can only be accessed by users from a specific region, you could create a simple principal class that implements the IPrincipal interface and adds a Region property, as shown in **Figure 2**.

To take advantage of this new principal class (which will work with any host), you just need to instantiate it and then use it to set the CurrentPrincipal and User properties. The following code looks for a value in the request's query string associated with the name "region." After retrieving that value, the code uses it to set the principal's Region property by passing the value to the class's constructor:

```
string region = (from kvp in request.GetQueryNameValuePairs()
                 where kvp.Key == "region"
                 select kvp.Value).FirstOrDefault();
Thread.CurrentPrincipal = new PHVPrincipal(userName, region);
```

If you're working in the Microsoft .NET Framework 4.5, rather than implementing the IPrincipal interface, you should inherit from the new ClaimsPrincipal class. ClaimsPrincipal supports both claims-based processing and integration with Windows Identity Foundation (WIF). That, however, is out of scope for this article (I'll address that topic in an upcoming article on claims-based security).

Authorizing a Custom Principal

With a new principal object in place you can create an authorization attribute that takes advantage of the new data carried by the principal. First, create a class that inherits from System.Web.Http.AuthorizeAttribute and overrides its IsAuthorized method (this is a different process from the ASP.NET MVC practice where you create new Authorization attributes by extending System.Web.Http.Filters.AuthorizationFilterAttribute). The IsAuthorized method is passed an HttpActionContext, whose properties can be used as part of your authorization process. However, this example just needs to extract the principal object from the Thread's CurrentPrincipal property, cast it to the custom principal type and check the Region property. If authorization succeeds, the code returns true. If authorization fails, you need the ActionContext Response property to create a custom response before returning false, as shown in **Figure 3**.

Your custom authorization filter can be used just like the default ASP.NET Authorize filter. Because this filter has a Region property, that property must be set to the acceptable region for this method as part of decorating a service method with it:

```
[RegionAuthorize(Region = "East")]
public HttpResponseMessage Get()
{
```

For this example, I've chosen to inherit from the AuthorizeAttribute because my authorization code is purely CPU bound. If my code needed to access some network resource (or do any I/O at all), a better choice would've been to implement the IAuthorizationFilter interface because it supports making asynchronous calls.

As I said at the start of this article: The typical Web API scenario doesn't require additional authorization, except to protect against CSFR exploits. But when you do need to extend the default security system, the Web API provides numerous choices throughout the processing pipeline where you can integrate whatever protection you need. And it's always better to have choices. ■

Peter Vogel *is a principal at PH&V Information Services, specializing in ASP.NET development with expertise in service-oriented architecture, XML, database and UI design.*

Thanks to the following technical experts for reviewing this article: *Dominick Baier (thinktecture GmbH & Co KG), Barry Dorrans (Microsoft) and Mike Wasson (Microsoft)*

Does your Team do more than just track bugs?

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

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

Track all your project tasks in one database so you can work together to get projects done.

- Quality Control / Compliance Tracking
- Project Management
- End User Accessible Service Desk Portal
- Bugs and Features
- Action Items
- Sales and Marketing
- Help Desk

Native Smart Card Login Support including Government and DOD

New in Team 2.11

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

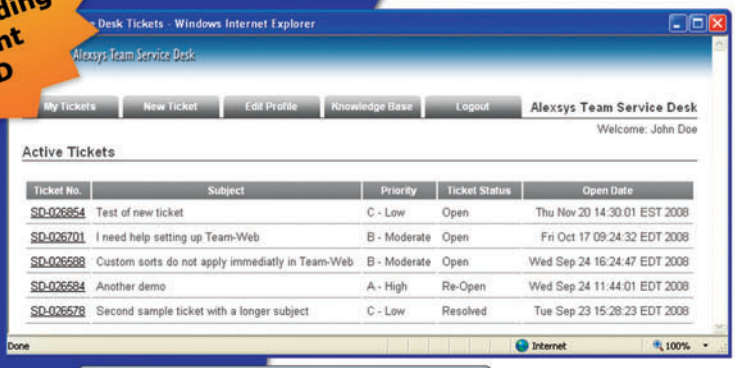
Service Desk Features

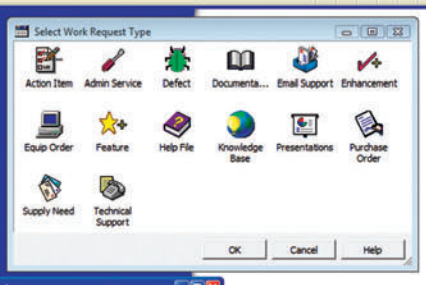
- Fully Secure
- Unlimited Users Self Registered or Active Directory
- Integrated into Your Web Site
- Fast/AJAX Dynamic Content
- Unlimited Service Desks
- Visual Service Desk Builder

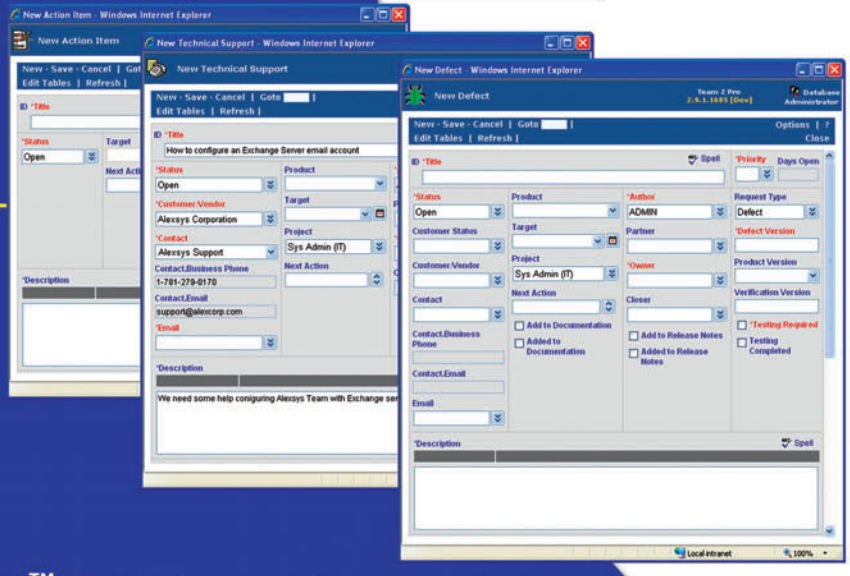
Team 2 Features

- Windows and Web Clients
- Multiple Work Request Forms
- Customizable Database
- Point and Click Workflows
- Role Based Security
- Clear Text Database
- Project Trees
- Time Recording
- Notifications and Escalations
- Outlook Integration

ZDNet Editors' Pick







Free Trial and Single User FreePack™ available at www.alexcorp.com.
FreePack™ includes a free single user Team Pro and Team-Web license.
Need more help? Give us a call at 1-888-880-ALEX (2539).

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

Exploring the JavaScript API for Office: Mail Apps

Angela Chu-Hatoun

This article is the fourth part of an in-depth look at the JavaScript API for Office, focusing on the portion of the API available to mail apps supported by Outlook and Outlook Web App. I assume you have some general understanding of apps for Office. If in doubt, reading the Dev Center documentation page, "Overview of apps for Office" (bit.ly/12nBWHG), will set you on the right path. Part 1 of this series, "Exploring the New JavaScript API for Office" (msdn.microsoft.com/magazine/jj891051), provides a high-level overview of the object model, which has the Office object as the root object. The Mailbox object, hanging off from Office.context, offers an entry point for most of the mail app-specific functionality in the object model, and this is where I'll pick up the discussion.

This article discusses:

- Contextual activation
- Using well-known entities
- Data storage
- Working with attachments and forms
- Authentication
- Exchange Web Services

Technologies discussed:

JavaScript API for Office

Code download available at:

code.msdn.microsoft.com/Mail-apps-for-Outlook-2b20fc16

The key takeaway of this article is what you, as a developer, can do with the JavaScript API for Office in mail apps. I've also provided an online companion article in which I walk through a sample mail app, complete with source code, at msdn.microsoft.com/magazine/dn205107. Now, I'll discuss various features of the JavaScript API for Office, starting out with some of the more basic, commonly used techniques and then moving on to more advanced concepts.

Fundamental Features of the JavaScript API for Office

The Mailbox object provides access to the user profile, the item currently selected by the user, forms to display the item, and a subset of Exchange Web Services (EWS) to manipulate items and folders in the mailbox. The Item object represents the selected message or appointment item. Through that object, you can further access built-in properties, custom properties and attachments of that item.

Controlling Activation Based on Item Type You can define rules in the manifest to determine when to activate a mail app. The item currently selected by the user and specified by Mailbox.item can be a Message (including meeting requests, responses and cancellations) or an Appointment object. Depending on your scenario, you can restrict the mail app to activate for only a specific type of item. The following XML example shows an ItemIs activation rule that restricts the mail app to activate only for messages:

```
<Rule xsi:type="ItemIs" ItemType="Message" />
```

Using the JavaScript API for Office, you can use the itemType property to verify the type of the selected item:

```
Office.context.mailbox.item.itemType
```

Item Access and Properties One basic function offered by mail apps is accessing the currently selected item and its built-in properties.

The following JavaScript example shows how to access the selected item and its built-in property, subject:

```
// The initialize function is required for all apps.
Office.initialize = function () {
  // Check for the DOM to load.
  $(document).ready(function () {
    var item = Office.context.mailbox.item;
    var subject = item.subject;
    // Continue processing the subject.
  });
}
```

Well-Known Entities Exchange Server recognizes certain entities that are available regardless of whether you used entities to activate the mail app. Outlook and Outlook Web App extract these entities if they exist in the subject or body of the selected item and make them available through the following JavaScript API methods:

- getEntities method
- getEntitiesByType(entityType) method
- getFilteredEntitiesByName(name) method

Supported entities include address, contact, email address and more.

Note that Outlook and Outlook Web Apps extract strings only in English regardless of the default locale you specify in the app manifest. They can't extract entities in the Sent Items folder. And they can extract meeting suggestions in messages but not in appointments.

Getting Matches That Result in Contextual Activation You can specify activation rules that depend on regular expression matches (ItemHasRegularExpressionMatch rules) or entity matches (ItemHasKnownEntity rules) in the selected item. You can use the following methods to get regular expression matches. These methods are available on the parent Item object, which can be a message or appointment:

- getRegExMatches method
- getRegExMatchesByName(name) method

You can use the methods listed in the Well-Known Entities section to get specific entity matches. Note that you can always use these methods to get any entity matches regardless of the type of activation rules you use for the mail app.

Examples The following is an example of an ItemHasRegularExpressionMatch rule, named VideoURL, which activates the mail app if the selected item is a message and the message body contains a URL for a video on YouTube:

```
<Rule xsi:type="RuleCollection" Mode="And">
  <Rule xsi:type="ItemIs" ItemType="Message"/>
  <Rule xsi:type="ItemHasRegularExpressionMatch"
    RegExName="VideoURL"
    RegExValue="http://www\.youtube\.com/watch\?v=[a-zA-Z0-9_-]{11}"
    PropertyName="Body"/>
</Rule>
```

The following JavaScript code sample shows how to use the Item.getRegExMatches method to get the matches of the preceding VideoURL regular expression:

```
Office.initialize = function () {
  // Check for the DOM to load.
  $(document).ready(function () {
    // Get an array of string matches for the regular expression VideoURL,
    // as specified in the manifest.
    var matches = Office.context.mailbox.item.getRegExMatches().VideoURL;
    // Continue processing the matches for the regular expression.
    });
}
```

The following is an example of an ItemHasKnownEntity rule that activates the mail app if an address is present in the subject or body of a message:

```
<Rule xsi:type="RuleCollection" Mode="And">
  <Rule xsi:type="ItemIs" ItemType="Message"/>
  <Rule xsi:type="ItemHasKnownEntity" EntityType="Address" />
</Rule>
```

Note that Outlook and Outlook Web Apps extract strings only in English regardless of the default locale you specify in the app manifest.

The following JavaScript code sample shows how to use the getEntitiesByType method to get an array of strings that are postal addresses in the subject or body of the currently selected message:

```
Office.initialize = function () {
  // Check for the DOM to load.
  $(document).ready(function () {
    var item = Office.context.mailbox.item;
    // Get an array of strings that represent postal
    // addresses in the current item.
    var addresses = item.getEntitiesByType(
      Office.MailboxEnums.EntityType.Address);
    // Continue processing the array of addresses.
  });
}
```

Activating a mail app based on contexts—especially regular expression matches or the existence of well-known entities—is powerful and can be complicated to debug. See the MSDN Library article, "Troubleshooting mail apps activation," at bit.ly/11COH30 for tips on how to debug activation issues.

Data Storage Per Item Per App The CustomProperties object lets you store item-specific data that only your mail app can access as a custom property on the item in a subsequent session. Such data is represented as key-value pairs. If the app is designed to run on

Figure 1 **Working with Custom Properties for the Current Item**

```
Office.initialize = function () {
  var mailbox = Office.context.mailbox;
  mailbox.item.loadCustomPropertiesAsync(customPropsCallback);
}

function customPropsCallback(asyncResult) {
  if (asyncResult.status == Office.AsyncResultStatus.Succeeded) {
    var customProps = asyncResult.value;
    var myProp = customProps.get("myProp");

    customProps.set("otherProp", "value");
    customProps.saveAsync(saveCallback);
  }
  else {
    write (asyncResult.error.message);
  }
}

function saveCallback(asyncResult) {
  // Callback method to save custom properties.
}
```

Figure 2 **Checking to See If User Has Selected an Appointment**

```
var myOm;
var myItem;

Office.initialize = function () {
  $(document).ready(function () {
    myOm = Office.context.mailbox;
    // Get the selected item.
    myItem = myOm.item;
    // Display the selected appointment.
    displaySelectedAppointment();
  });
}

function displaySelectedAppointment() {
  // Display selected item only if the item is an appointment.
  if (myitem.itemType == Office.MailboxEnums.ItemType.Appointment)
  {
    myOm.displayAppointmentForm(myItem.itemId);
  }
  else
  {
    // Handle condition as appropriate.
  }
}
```

multiple Outlook clients and form factors, the custom properties roam with the supported Outlook client and form factor.

The custom properties for an item are available through the Item.loadCustomPropertiesAysnc method. This is an asynchronous method that takes a callback method as a parameter. When the custom properties are loaded, they're available through the asyncResult.value property, which is passed to the callback method as an input parameter. **Figure 1** shows how to load, get, set and save custom properties for the current item.

Data Storage Per Mailbox Per App The RoamingSettings object lets you store custom data that your mail app can access for the same mailbox regardless of whether the mail app is running on a desktop, tablet or smartphone. The following JavaScript example shows how to get mailbox-specific data in the initialize event handler of the mail app:

```
var _mailbox;
var _settings;
Office.initialize = function () {
  // Check for the DOM to load using the jQuery ready function.
  $(document).ready(function () {
    // Initialize instance variables to access API objects.
    _mailbox = Office.context.mailbox;
    _settings = Office.context.roamingSettings;
    // Continue with app-specific code.
  });
}
```

Figure 3 **Assigning Values to Specific Fields in a New Appointment Form**

```
var myOm;
Office.initialize = function () {
  $(document).ready(function () {
    myOm = Office.context.mailbox;
    // After the DOM is loaded, can display the
    // pre-populated new appointment form.
    displayFormForNewAppointment();
  });
}

function displayFormForNewAppointment(){
  var formParameters =
  {
    "requiredAttendees" : ["wendy@contoso.com", "derek@contoso.com"],
    "optionalAttendees" : ["shane@contoso.com", "michael@contoso.com"],
    "start" : new Date("September 27, 2012 11:15:00"),
    "end" : new Date("("September 27, 2012 12:30:00"),
    "location" : "Conf room 200",
    "resources" : ['sound', 'lighting', 'recording'],
    "subject" : "Next rehearsal",
    "body" : "This is about the next rehearsal."
  }

  // Display a form to create an appointment with
  // the specified fields in the form.
  myOm.displayNewAppointmentForm(formParameters);
}
```

User Profile You can access the user's profile using the Office.context.mailbox.userProfile property. Through the UserProfile object, you can get the display name, SMTP address and the local time zone of the signed-in user. The following JavaScript code sample shows how to access the UserProfile.emailAddress property of the current user signed in to Outlook or Outlook Web App:

```
Office.initialize = function () {
  // Check for the DOM to load.
  $(document).ready(function () {
    var userProfile = Office.context.mailbox.userProfile;
    var SMTPAddress = userProfile.emailAddress;
    // Continue with processing the user's SMTP address.
  });
}
```

Displaying Outlook Forms In Outlook a user can specify forms to display appointments and messages by default. Using the following methods in the JavaScript API for Office, a mail app can display an appointment and message respectively, using the same forms that Outlook uses:

- Mailbox.displayAppointmentForm(itemId)
- Mailbox.displayMessageForm(itemId)

The code in **Figure 2** checks to see if the user has selected an appointment and displays it with the Outlook default appointment form.

The Mailbox.displayNewAppointmentForm method allows you to assign values to certain fields in an appointment form for a new meeting and display the form for the user to fill in the appointment. When calling this method, specify these fields as a list of string-value pairs, as shown in **Figure 3**.

The methods that display reply forms to an appointment or message offer the most room for customization:

- Appointment.displayReplyAllForm(htmlBody) method
- Appointment.displayReplyForm(htmlBody) method
- Message.displayReplyAllForm(htmlBody) method
- Message.displayReplyForm(htmlBody) method

With each of these methods, you can provide an HTML string that represents the body of the reply form. The example in **Figure 4** displays a reply-all form for a message.

Advanced Concepts

Getting Item Attachments A mail app can use the JavaScript API for Office and EWS callback tokens to access attachments of the currently selected message or appointment in the user's mailbox. The use of the callback token allows back-end, server-side code of the mail app—or third-party Web services—to call the EWS operation, GetAttachment, to actually get the specific attachments.

The following describes how a mail app gets attachments (this process is also illustrated in **Figure 5**):

WINDOWS FORMS | WPF | ASP.NET

FLOW TYPE LAYOUT REPORTING

Reuse MS Word documents or templates as your reporting templates.

Easy database connection with master-detail nested blocks.

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

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

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

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

TX TEXT CONTROL® word processing components

Visual Studio Microsoft Partner

US +1 855-533-8398
EU +49 421-4270671-0

WWW.TEXTCONTROL.COM

Figure 4 **Specifying the HTML for a Reply-All Form for a Message Item**

```
var myOm;
var myItem;

Office.initialize = function () {
  // Check for the DOM to load.
  $(document).ready(function () {
    myOm = Office.context.mailbox;
    // Get the selected item.
    myItem = myOm.item;
    // After the DOM is loaded, display
    // the reply form.
    displayReply();
  });
}

function displayReply(){
  // Display a reply form to the sender and recipients only
  // if the selected item is a message.
  if (myitem.itemType == Office.MailboxEnums.ItemType.Message) {
      myItem.displayReplyAllForm
      ("<html><head><head><body>This is the body of " +
       "the email reply.</body></html>");
  }
}
```

1. A mail app uses Mailbox.getCallbackTokenAsync to request a callback token, uses Item.attachments to get metadata about the attachments of the selected appointment or message (including attachments IDs), and uses Mailbox.ewsUrl to get the URL of the EWS endpoint for the current e-mail account.
2. The mail app passes the callback token, attachment IDs and URL of the EWS endpoint to the server-side code.
3. The server-side code calls the EWS GetAttachment operation to get the actual attachments from that Exchange store.

Note that the server-side code should access EWS by directly creating EWS requests. You can't use the Mailbox.makeEwsRequestAsync method to access the GetAttachment operation. As of this writing, accessing attachments is supported if the mail app runs on Outlook Web App. The feature isn't available if the mail app is running in Outlook.

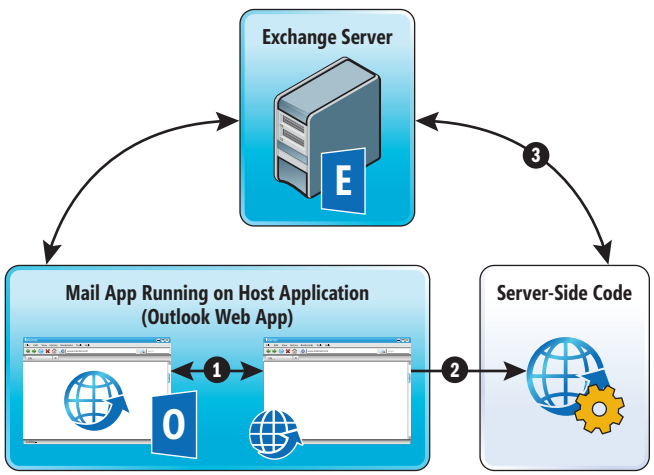


Figure 5 **Mail Apps Accessing Attachments from an Exchange Server**

See the MSDN code sample, "Mail apps for Office: Get attachments from an Exchange server" (bit.ly/11dG9fS), for an example of how to access attachments in a mail app.

User Token for Single Sign-On Your mail app can use common authentication Web techniques to authenticate and authorize users. If you want to support single sign-on authentication for your users across Outlook and Outlook Web App, you can use Exchange identity tokens by calling the Mailbox.getUserIdentityTokenAsync method.

Single sign-on is based on an Exchange Server assigning an Exchange identity token for the user's e-mail account on that Exchange Server. An Exchange identity token contains multiple fields, among which are a signature for validating the token and a unique identifier representing the Exchange e-mail account.

In general, a mail app that authenticates users can interact with a Web service—which can be server-side code for the mail app—or a third-party Web service, which uses its own Identity Provider (IdP) to validate user identity in a federated identity system. The key to single sign-on is the mapping that the Web service maintains between the unique ID (from the Exchange identity token) and the user identity (which is known to the IdP).

Figure 6 provides a high-level description of using Exchange identity tokens for authentication.

The following summarizes the process:

1. The mail app calls Mailbox.getUserIdentityTokenAsync to request an Exchange identity token for the user, specifying a callback method that runs when the asynchronous get operation completes.
2. The callback function gets the Exchange identity token and passes it to the Web service for authentication.
3. The Web service gets a public key from the Exchange Server to validate the token.
4. Depending on whether the user has signed in before, the Web service proceeds as follows:
   a. If it's the first time the user signs in to the mail app, no prior mapping exists for that account, and the Web service responds to the mail app to prompt the user for credentials. Upon receiving the credentials, the mail app passes them to the Web service.
   b. The service then creates a mapping between the unique ID provided by the token and the user identity from the credentials that are well-known to the IdP. Using the credentials, the service can log in the user.
   c. On subsequent times: When the user opens the mail app in Outlook or Outlook Web App from a desktop, tablet or browser on a smartphone, the following steps occur:
      1. The mail app calls getUserIdentityTokenAsync and gets the same unique ID in the token
      2. The mail app passes the token to the Web service.
      3. The Web service validates the token.
      4. The Web service then looks up the mapping table, finds the user and authorizes access.

Once you validate the token, you can keep the user logged in, even when the user opens the mail app from another host application or device.

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.

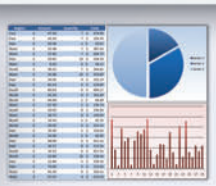
Excel Reporting for ASP.NET, WinForms, WPF and Silverlight

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

Visual Studio Microsoft Partner

SpreadsheetGear

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

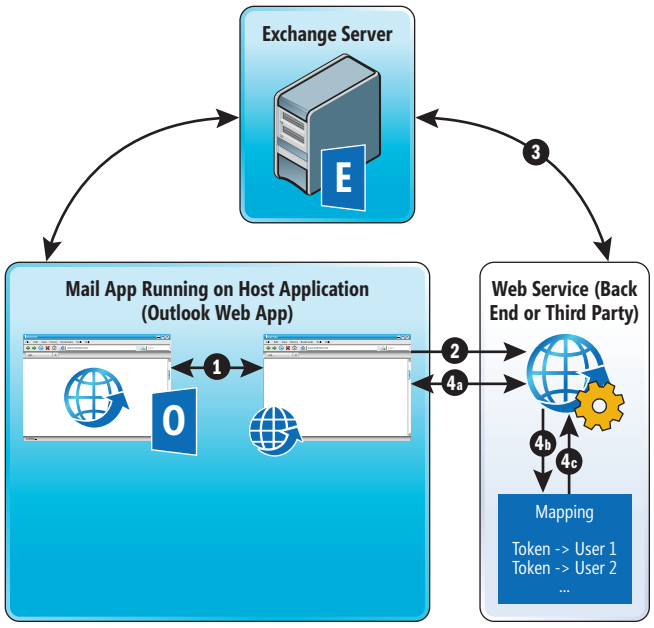


Figure 6 **Authentication Using Exchange Identity Tokens**

The details for authentication and validation are fairly complex. You can use an EWS Managed API Validation Library to simplify the process. Using that Validation Library, a Web service would create an object, extract and put the token into the object, verify if the signature is valid, and if it's valid, then put the unique identifier in the mapping database.

For more information about using Exchange identity tokens for authentication, see the following Dev Center documentation:

- "Authenticating a mail app by using Exchange identity tokens" (bit.ly/ZYaZ9v)
- "Inside the Exchange identity token" (bit.ly/ZxRogq)
- "How to: Call a service by using an identity token in Exchange" (bit.ly/12jqxcU)
- "How to: Use the Exchange token validation library" (bit.ly/11akmEg)
- "How to: Validate an Exchange identity token" (bit.ly/15im6SO)
- "How to: Authenticate a user with an identity token for Exchange" (bit.ly/13gZ36T)

For code samples:

- You can refer to "Mail apps for Outlook: Use a client identity token" sample in C# for Visual Studio 2012 (bit.ly/ZxS85b) for a C# example of a mail app and a Web service that use Exchange identity tokens for authentication.
- See "Mail apps for Outlook: Validate a client identity token using the .NET Framework" (bit.ly/17j9FSZ) for an example of how to use the .NET Framework to validate Exchange client identity tokens.

Access to a Subset of EWS Using the Mailbox.makeEwsRequestAsync method, mail apps can access a subset of EWS operations to create, find, get, update, move, or send an item or folder. For example, you can use the CreateItem operation to create an appointment, message or contact. You can use the FindItem operation to look up items in a mailbox. And you can use the SendItem operation to send a message or meeting invitation. The supported subset of operations pertains to only the user's mailbox. Other EWS operations that pertain to an entire organization—such as accessing the Global Address List or iterating through every person in the organization—are off limits to mail apps. Also, because the supported subset of operations is powerful, mail apps must request read/write mailbox permission in the app manifest, and only administrators can install mail apps requiring such permission.

When you call the Mailbox.makeEwsRequestAsync method, you're requesting EWS on the Exchange Server that hosts the user's mailbox. Before calling makeEwsRequestAsync, specify input arguments for the following parameters:

- data parameter: Create an XML SOAP request for the EWS operation you intend to call.
- callback parameter: A callback method that would handle the response from the operation.
- userContext parameter: Any input data for the callback method.

When the operation finishes, the apps for Office framework calls the callback method with one argument, which is an AsyncResult object, similar to the other asynchronous methods that you've seen in the first part of this series. The callback method can access the AsyncResult.value property to get the XML SOAP response that contains data relevant to the operation. The callback method can also access the AsyncResult.asyncContext property to access any other input parameter passed in through the userContext parameter. The callback method then parses the XML in the SOAP response to get the data relevant for its purposes.

For a complete list of supported EWS operations and further details, see the Dev Center documentation, "Calling Web services from a mail app for Outlook" (bit.ly/12jtH0z).

For an example of a mail app that demonstrates calling the GetItem operation, see "Mail apps for Outlook: Make an EWS request" sample in C# for Visual Studio 2012 (bit.ly/Zv3hEQ).

Wrapping Up

That wraps up my discussion of building mail apps, as well the four-part exploration of the JavaScript API for Office. For the previous installments in this series, and the online companion article for this installment, see:

- "Exploring the New JavaScript API for Office" at msdn.microsoft.com/magazine/jj891051
- "Exploring the JavaScript API for Office: Data Access and Events" at msdn.microsoft.com/magazine/jj991976
- "Exploring the JavaScript API for Office: Data Binding and Custom XML Parts" at msdn.microsoft.com/magazine/dn166930
- "Exploring the JavaScript API for Office: A Sample Mail App" at msdn.microsoft.com/magazine/dn205107. ■

Angela Chu-Hatoun *started as a software developer and switched to writing for her greater interest in explaining how software works. She has been a programmer writer in the Office Division for 12 years, and writes for developers about creating apps for Office and other solutions for Outlook. She enjoys reading about current affairs, gardening, travelling, food and fashion.*

Thanks to the following technical experts for reviewing this article: *Andrew Salamatov (Microsoft) and Tim Wan (Microsoft)*

Extreme Performance & Linear Scalability

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

Enterprise Distributed Cache

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

ASP.NET Optimization in Web Farms

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

Runtime Data Sharing

- Powerful event notifications for pub/sub data sharing

Download a 60-day FREE trial today!

FREE EXPRESS EDITION

NCache™

Distributed Cache for .NET & Java

www.alachisoft.com

1-800-253-8195

Extending Visual Studio Team Explorer 2012

Mike Fourie

In this article I'll discuss new extensibility points provided in Microsoft Visual Studio Team Explorer 2012. I'll build a working sample to highlight the architecture of Team Explorer 2012 while hopefully providing you with some inspiration to build extensions that will be beneficial to you, your team and perhaps even the wider community.

Since Visual Studio 2005, Team Explorer has provided the primary interface for interacting with the various team-centric features provided in Visual Studio such as Work Items, SharePoint Documents, Reports, Builds and Source Control.

The subsequent releases of Visual Studio 2008 and Visual Studio 2010 provided minor updates to Team Explorer, but the basic Tree View navigation paradigm remained in place—until Visual Studio 2012 arrived, that is. Visual Studio 2012 introduced a completely redesigned Team Explorer and UX. Gone is the Tree View, which has been replaced by a collection of Pages that provides access to the same features we've become accustomed to, as well as some new features like My Work.

Also, note that with Visual Studio 2012, you can now work entirely in managed code without any concerns about dealing with COM internals.

This article discusses:

- An overview of Team Explorer 2012
- Additional sources of information
- Team Explorer architecture
- Building an example extension app
- Debugging the app
- Project organization

Technologies discussed:

Microsoft Visual Studio 2012

Code download available at:

archive.msdn.microsoft.com/mag201306Rangers

Team Explorer 2012 Overview

Team Explorer 2012 comprises several Pages, each of which can be split into Sections and host Navigation Items and Navigation Links. The default page is called Home and it provides easy access to the other pages provided in Team Explorer, namely My Work, Pending Changes, Work Items, Builds, Web Access, Team Members and Settings.

Figure 1 shows the Home page and the various Navigation Items and Links provided.

Figure 2 shows the use of Sections, which you can add to Pages.

If this is the first time you've seen Team Explorer 2012, you'll no doubt agree that the design is completely different from the previous Tree View-based experience provided in Visual Studio 2005, 2008 and 2010.

Given the simplicity and efficiency of the previous Tree View implementations, you might wonder why there's been such a radical redesign. After working in the new Page-based Team Explorer you might feel—and I think with some justification—that the navigation isn't as efficient as the simple Tree View that it replaced. It certainly takes some getting used to, though hopefully by the end of this article you'll understand that the current loss in navigation efficiency is outweighed by the powerful extensibility features that the new Page design provides and the reduction in modal dialogs that are prevalent in previous versions. Note that I say "current" loss in navigation efficiency, as I expect the design to

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

be tweaked throughout the course of the quarterly updates Microsoft is now providing for Visual Studio. In fact, Visual Studio 2012 Update 2 has a handy new Connect Page that displays every Team Foundation Server (TFS), Team Project Collection and Team Project to which you've previously connected, making navigating TFS connections much easier.

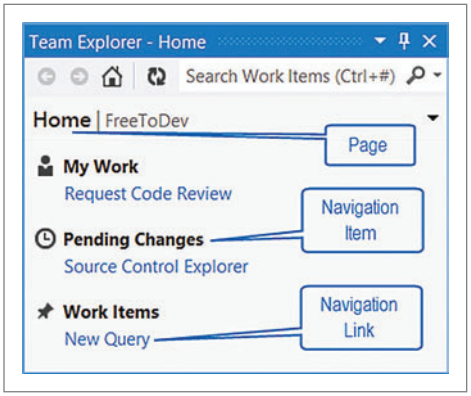


Figure 1 **The Team Explorer 2012 Home Page**

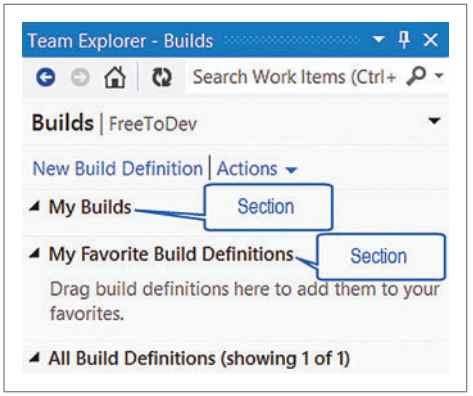


Figure 2 **Sections in the Team Explorer Builds Page**

Existing Team Explorer Extensibility Resources

In addition to this article, you should be aware of a few other resources that can help you find your way around extending Team Explorer. MSDN has API-level documentation covering the Microsoft.TeamFoundation.Controls namespace that hosts all the objects I'll extend.

The Visual Studio Developer Center has a sample, "Extending Team Explorer in Visual Studio 2012" (bit.ly/Zp1OFf), that was published by the team that developed Team Explorer 2012. The MyHistory sample extension provided with this article is based on that sample and I recommend you read it first.

Team Explorer Architecture

Before you start creating extensions for Team Explorer it will be beneficial to understand a little about how Team Explorer is architected.

The extensions you create need to implement the interface for the extension point you're targeting. These interfaces are implemented in the Microsoft.TeamFoundation.Controls namespace and are defined as follows:

- Page: ITeamExplorerPage (bit.ly/11FvPyE)
- Section: ITeamExplorerSection (bit.ly/Y6Y22X)
- Navigation Item: ITeamExplorerNavigationItem (bit.ly/11Hh2Cm)
- Navigation Link: ITeamExplorerNavigationLink (bit.ly/17XFHEp)

Extensions you create are discovered using the Managed Extensibility Framework (MEF). To aid discovery, you must decorate your class with the appropriate attribute as defined here:

- Page: [TeamExplorerPage("guid")] (bit.ly/17kBzQD)
- Section: [TeamExplorerSection("guid", "parentPageGuid", priority)] (bit.ly/12nMxCC)
- Navigation Item: [TeamExplorerNavigationItem("guid", priority)] (bit.ly/Y6YQ7S)
- Navigation Link: [TeamExplorerNavigationLink("guid", "parentNavigationItemGuid", priority)] (bit.ly/12KFghj)

For your extension to be discovered, your assembly needs to be in one of the "well-known" extension locations for Visual Studio. These locations are defined in the "Master PkgDef" file, located at <VsInstallRootFolder>\Common7\IDE\devenv.pkgdef.

The recommended packaging mechanism for your extensions is to use a Visual Studio Extension (VSIX). This mechanism will take care of all the deployment plumbing for you. If you'd like to read more on discovery, see The Visual Studio Blog post, "How VSIX extensions are discovered and loaded in VS 2010" (bit.ly/xTJrSv).

To illustrate the extension lifetime, navigation, context management and other features available to your extension, let's take a look at the My History sample provided with this article.

Getting Started with the Sample App

You can download the full working sample for this article, but I believe it's important to highlight a few key points to help you successfully get started with building your own extension.

To create an extension you're going to need Visual Studio 2012—and in particular, the Visual Studio 2012 SDK—to create the VSIX project. When you create a VSIX project, Visual Studio takes you straight to the .vsixmanifest file shown in **Figure 3**. I've added my own license file to the project and completed a few fields as shown.

You can choose to add your extension in a separate assembly or host all your code within the VSIX project. If you choose to host all your code in the VSIX, it's important to manually tweak the .csproj file so the extension assembly is included in the VSIX package. By default, the VSIX project creates only the VSIX package. When you deploy your extension, it will show as installed; however, your assembly won't be deployed, thus it won't be discovered. To include the assembly, you need to open the .csproj file in a text editor and set the project settings as shown in the following code (this change can't be made via the UI):

```
<IncludeAssemblyInVSIXContainer>true</IncludeAssemblyInVSIXContainer>
<CopyBuildOutputToOutputDirectory>true</CopyBuildOutputToOutputDirectory>
```

Within the manifest file you need to configure Assets so the extension can deploy your extension appropriately. On the Assets tab, add a new Microsoft.VisualStudio.VsPackage for the project. This tells the VSIX that you're creating a Visual Studio package. Also, add an Asset of type Microsoft.VisualStudio.MefComponent, which provides the name of the extension assembly in the VSIX package.

A VSIX project has no References by default, so you need to add all those that are required. At a minimum you're likely to need the following references:

- Microsoft.TeamFoundation.Client
- Microsoft.TeamFoundation.Controls
- Microsoft.TeamFoundation.VersionControl.Client
- Microsoft.TeamFoundation.VersionControl.Controls
- Microsoft.VisualStudio.Shell.Interop
- Microsoft.VisualStudio.Shell.Interop.10.0
- Microsoft.VisualStudio.Shell.Interop.8.0

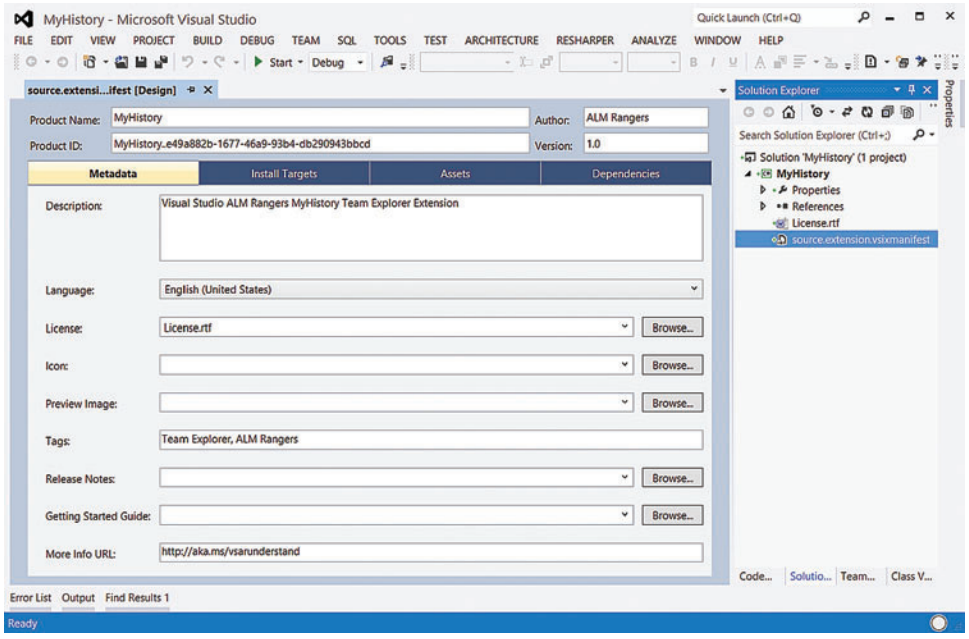


Figure 3 **Details of the .vsixmanifest File**

- Microsoft.VisualStudio.Shell.Interop.9.0
- Microsoft.VisualStudio.TeamFoundation.VersionControl
- Microsoft.VisualStudio.OLE.Interop
- Microsoft.VisualStudio.Shell.11.0
- Microsoft.VisualStudio.Shell.Immutable.10.0
- PresentationCore
- PresentationFramework
- System
- System.ComponentModel.Composition
- System.Core
- System.Drawing
- System.Windows.Forms
- System.Xaml
- System.Xml

Although you can create your extension in managed code, you'll likely still need to integrate with Visual Studio via COM, so you'll most likely also need COM references to EnvDTE and EnvDTE80.

Once you've done all that, you're ready to start creating content to extend Team Explorer.

Debugging

If this is your first adventure using VSIX projects, you may be wondering how to debug them. In Visual Studio 2012 the new project is now created with the required Debug information, so you can simply hit F5 and a new "experimental" instance of Visual Studio (bit.ly/12KRCGq) will start with your extension loaded.

You can set breakpoints in your code and debug like any other app, although perhaps a little slower than you may be used to for simpler applications. Somehow my project managed to lose the settings, so I've shown them in **Figure 4**. Basically the Start Action is set to Start external program with the value C:\Program Files (x86)\Microsoft Visual Studio 11.0\Common7\IDE\devenv.exe and the Command line arguments are set to /rootsuffix Exp.

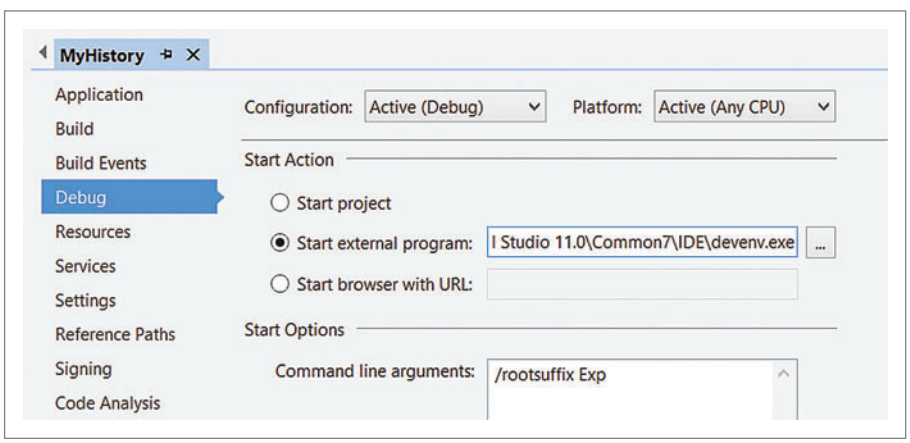


Figure 4 **VSIX Debug Settings**

Walk-Through of the My History Sample

Developers often need to multitask (that's the fancy word for randomization!) and sometimes forget what they did just a few minutes or days ago. The My History extension aims to help improve your multitasking experience by providing a list of Shelvesets, Changesets, Work Items and Projects/Solutions with which you've recently worked. Shown in **Figure 5**, it consists of a new Page along with multiple Sections, Navigation Items and Navigation Links.

At this point I'd recommend installing the extension to see how it extends Team Explorer, then come back to this article to go through some principal parts of the extension code.

The Code in My History

The solution is organized into a set of folders for easier navigation and discovery.

The Base folder contains a set of classes based on those provided in the aforementioned "Extending Team Explorer in Visual Studio 2012" sample. The Internals and Resources folders contain various helper classes. You're most interested in the Sections folder, which contains the implementation of the objects from which you need to derive. **Figure 6** shows how the classes are related.

To create an entry point to the new Page you need to add a Navigation Link to one of the existing sections on the Home Page.

Visual Studio
PARTNERS

Extended & empowered by partners.

Get more out of Visual Studio by using tools and extensions developed by Visual Studio Partners.

Find Partners

vsipprogram.com/partners

Visual Studio
Partner

Visual Studio Partners are recognized experts who use the highly extensible framework of Visual Studio to create innovative products and solutions that support specialized needs for developing quality enabled, agile applications and services.

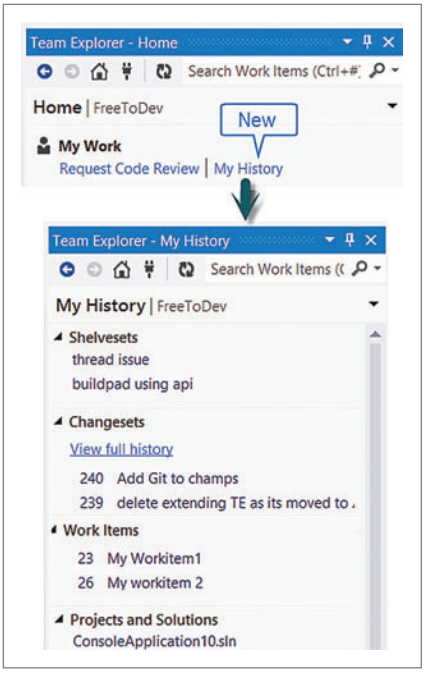


Figure 5 **The My History Extension**

Figure 7 shows the code for the custom Navigation Link contained in MyHistoryNavigationLink.cs. This code produces the My History link shown in **Figure 5**.

When the link is clicked, the Execute method is run and Team Explorer is made to navigate to your custom My History Page. Let's take a closer look at the first line in **Figure 7**. The TeamExplorerNavigationLink class attribute is used to associate the link with the MyWork section, and the priority of 200 means that link will likely be displayed last, assuming someone hasn't installed another extension with a higher number (lower priority).

Figure 8 shows the code in MyHistoryPage.cs, which provides the addition of a new Team Explorer Page. Note the attribute usage and unique ID used.

You have an entry point to your new Page, and you can get to your new Page; now you need to get content onto the Page. I'll cover the Changesets Section in detail and leave it to you to take a look at the other implemented Sections in your own time. All follow the same basic pattern.

In the accompanying code download is the ChangesetsSection.cs class, which is too long to list in this article. In this class you register a new ITeamExplorerSection with the MyHistory Page, using the TeamExplorerSection class attribute. Note that the value of 20 in the class attribute is the priority of the section. The lower the priority, the higher up in the Page the Section gets rendered. The class declares an ObservableCollection of Changesets to which your UI is bound. In the constructor of the class you define additional visual and behavioral settings. It's important to link the class to its rendered content, which is a new instance of your User Control, ChangesetsSectionView.xaml.

Note that in the Initialize method you check for context so you can reload it from memory rather than re-query it. The SaveContext method saves necessary context information for future use, and the ContextChanged method can be used to refresh the data if the user changes projects or collections. In all cases, if you need to re-query it, you do so in an async fashion to avoid blocking the UI thread. The ViewChangesetDetails method is called from

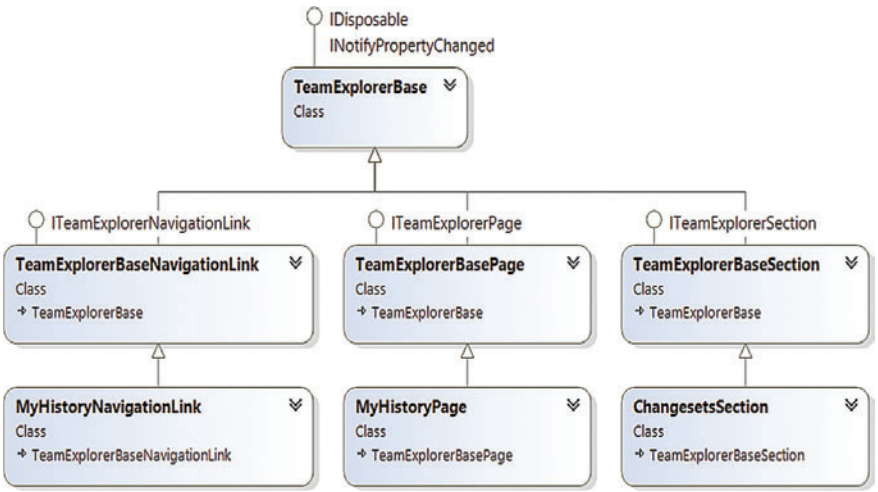


Figure 6 **Diagram of MyHistory Classes**

Figure 7 **The My History Navigation Link**

```
[TeamExplorerNavigationLink(MyHistoryNavigationLink.LinkId,
  TeamExplorerNavigationItemIds.MyWork, 200)]
public class MyHistoryNavigationLink : TeamExplorerBaseNavigationLink
{
  public const string LinkId = "e49a882b-1677-46a9-93b4-db290943bbcd";

  [ImportingConstructor]
  public MyHistoryNavigationLink([Import(typeof(SVsServiceProvider))]
    IServiceProvider serviceProvider) : base(serviceProvider)
  {
    this.Text = "My History";
    this.IsVisible = true;
    this.IsEnabled = true;
  }

  public override void Execute()
  {
    try
    {
      ITeamExplorer teamExplorer = GetService<ITeamExplorer>();
      if (teamExplorer != null)
      {
        teamExplorer.NavigateToPage(new Guid(MyHistoryPage.PageId), null);
      }
    }
    catch (Exception ex)
    {
      this.ShowNotification(ex.Message, NotificationType.Error);
    }
  }
```

Figure 8 **The MyHistory Page**

```
/// <summary>
/// MyHistory Page. We're extending Team Explorer by adding a new page and
/// therefore use the TeamExplorerPage attribute and pass in our unique ID
/// </summary>
[TeamExplorerPage(MyHistoryPage.PageId)]
public class MyHistoryPage : TeamExplorerBasePage
{
  // All Pages must have a unique ID; use the Tools | Create
  // GUID menu in Visual Studio to create your own GUID
  public const string PageId = "BAC5373E-1BE5-4A10-97F5-AC278CA77EDF";

  public MyHistoryPage()
  {
    // Set the page title
    this.Title = "My History";
  }
}
```

Figure 9 **The Changesets Section UI**

Figure 10 **The Changesets User Control Code**

```
public partial class ChangesetsSectionView
{
  public static readonly DependencyProperty ParentSectionProperty =
    DependencyProperty.Register("ParentSection",
    typeof(ChangesetsSection), typeof(ChangesetsSectionView));

  public ChangesetsSection ParentSection
  {
    get { return (ChangesetsSection)GetValue(ParentSectionProperty); }
    set {SetValue(ParentSectionProperty, value);}
  }

  private void HistoryLink_Click(object sender, RoutedEventArgs e)
  {
    this.ParentSection.ViewHistory();
  }

  private void ChangesetList_MouseDoubleClick(
    object sender, MouseButtonEventArgs e)
  {
    if (e.ChangedButton == MouseButton.Left &&
      changesetList.SelectedItems.Count == 1)
    {
      this.ViewChangesetDetails();
    }
  }
}
```

your User Control. This method interacts with Team Explorer and provides it with the information it needs to render the Changeset details in the correct Page.

The final piece in the puzzle is to create your user control to display the content. The Changesets section provides a double-clickable list of the last 10 check-ins the user did, each with a tooltip showing a few pertinent details and a hyperlink to take the user to the full Changeset history, as shown in **Figure 9**.

The code for the user control is fairly basic, and it really just handles the user events, which make calls through to the ParentSection to interact with Team Explorer. **Figure 10** provides an abbreviated view over some of the codebehind. Note the use of the DependencyProperty class (bit.ly/11FJNQW), which helps to bind the user control to the class you've registered with Team Explorer.

Awareness, Insight and a Starting Point

There's a lot more code to look through in the sample, which, together with what I've covered in this article, hopefully provides you with:

- An awareness of the extensibility Team Explorer 2012 provides.
- An insight into the vast API Visual Studio provides with which you can interact (be sure to read the *MSDN Magazine* article, "Version Control in the TFS Client Object Model" [msdn.microsoft.com/magazine/jj883959], if you're interested in more API-level reading matter).
- A starting point and the confidence to create your first extension.

Thanks for taking the time to read this, and please look for more articles from the ALM Rangers (aka.ms/vsarunderstand). ■

Mike Fourie *is an independent consultant with more than 13 years of software development experience who specializes in build and deployment automation. He's a Microsoft ALM MVP and Distinguished ALM Ranger. He can be reached via his blog at freetodev.com. You can also follow him on Twitter at twitter.com/mikefourie.*

Thanks to the following technical experts for reviewing this article: *Chad Boles (Microsoft), Jeff Bramwell (Farm Credit Services of America), Willy-Peter Schaub (Microsoft) and Hamid Shahid (Consultant)*

JAMES MCCAFFREY

Amoeba Method Optimization Using C#

The goal of numerical optimization is to solve an equation. There are many calculus-based deterministic techniques available; however, some difficult problems, especially in the areas of machine learning and artificial intelligence, can't be solved easily using classical optimization techniques. In such situations, alternatives such as amoeba method optimization can be of value. Amoeba method optimization, which I'll discuss in this article, is similar in some respects to particle swarm optimization, which I described in the August 2011 issue of *MSDN Magazine* (msdn.microsoft.com/magazine/hh335067).

The best way to understand what amoeba method optimization is and to see where I'm headed is to examine **Figure 1**, which shows a demo program in action. The goal in this program is to find the values of x and y that minimize a relatively simple standard benchmark problem called Rosenbrock's function. This function has a known solution at x = 1.0 and y = 1.0 when the value of the function is 0.

The demo program creates a virtual amoeba that contains three random potential solutions. The best of these initial solutions isn't very good, at x = -0.66 and y = 5.43, yielding a function value of 2499.52. The demo program calls a solve method and, behind the scenes, it uses the amoeba method to iteratively find better and better solutions. After 50 iterations, the algorithm succeeds in finding the optimal solution.

```
file:///C:/AmoebaOptimization/bin/Debug/AmoebaOptimization.EXE

Begin amoeba method optimization demo

Solving Rosenbrock's function f(x,y) = 100*(y-x^2)^2 + (1-x)^2
Function has a minimum at x = 1.0, y = 1.0 when f = 0.0

Creating amoeba with size = 3
Setting maxLoop = 50

Initial amoeba is:

[0] [ -0.66  5.43 ]  val = 2499.5203
[1] [  3.15 -1.34 ]  val = 12704.1758
[2] [ -5.03 -7.79 ]  val = 109280.5356

Beginning reflect-expand-contract solve loop

At t = 10 curr best solution = [  2.07  4.13 ]  val = 3.1564
At t = 20 curr best solution = [  1.52  2.31 ]  val = 0.2720
At t = 30 curr best solution = [  1.29  1.65 ]  val = 0.0990
At t = 40 curr best solution = [  1.10  1.20 ]  val = 0.0131
At t = 50 curr best solution = [  1.00  1.00 ]  val = 0.0000

Solve complete

Final amoeba is:

[0] [  1.00  1.00 ]  val = 0.0000
[1] [  1.00  1.00 ]  val = 0.0000
[2] [  1.00  1.01 ]  val = 0.0001

Best solution found:

[  1.00  1.00 ]  val = 0.0000

End amoeba method optimization demo
```

Figure 1 **Amoeba Method Optimization Demo**

In the sections that follow I present and explain the complete source code for the demo program. The code is available at archive.msdn.microsoft.com/mag201306TestRun. This article assumes you have at least intermediate-level programming skills with a modern procedural language. I coded the demo using C# but you shouldn't have too much trouble refactoring the demo to another language, such as Visual Basic .NET or Python.

The Amoeba Method Algorithm

The amoeba method optimization algorithm I present here is based on the 1965 research paper, "A Simplex Method for Function Minimization," by J.A. Nelder and R. Mead.

The key parts of this algorithm are illustrated in **Figure 2**. At any point in time, there are several possible solutions. In most cases, amoeba optimization uses three solutions, colored red in the figure. Each solution has an associated objective function value, so there will be a worst solution (the highest function value because the goal is to minimize), a best solution (the smallest function value) and other(s).

The algorithm is an iterative process. At each step, it attempts to replace the worst solution with a new, better solution from among three candidates: a reflected point, an expanded point and a contracted point. Each of these candidates lies along a line from the worst point through the centroid—a point that's in the middle of all points except the worst point. In the usual case with three solutions, the centroid will lie halfway between the best point and the other (non-worst) point.

If neither the reflected point, nor the expanded point, nor the contracted point is better than the current worst solution, the amoeba shrinks itself by moving all points, except for the best point, halfway toward the best point. Some research papers call this process multiple contraction.

When graphed over time, if the three current solution points are connected by a line (as with the black dashed line in **Figure 2**), the

Code download available at archive.msdn.microsoft.com/mag201306TestRun.

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...

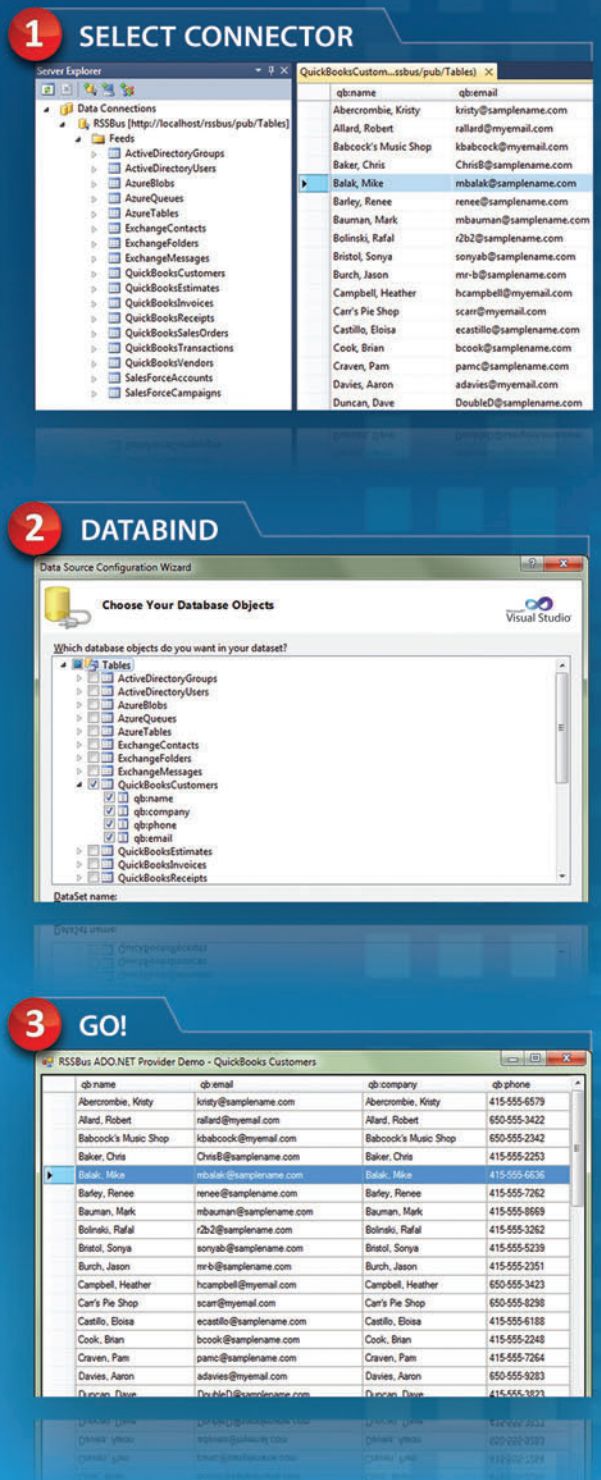
The RSSBus Data Providers give your .NET applications the power to databind (just like SQL) to Amazon, PayPal, eBay, QuickBooks, FedEx, Salesforce, MS-CRM, Twitter, SharePoint, Windows Azure, and much more! Leverage your existing knowledge to deliver cutting-edge WinForms, ASP.NET, and Windows Mobile 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 ...



rssbus

Download a free 30-day trial online at: **www.rssbus.com**

© Copyright 2013 RSSBus Inc. All rights reserved. All trademarks and registered trademarks are the property of their respective owners.

solutions form a triangle, and their movement resembles that of an amoeba crawling through its environment. In mathematical terms, a triangle on a plane is called a simplex, so this optimization algorithm, in addition to being called the amoeba method or the Nelder-Mead method, is sometimes called the simplex method.

Overall Program Structure

I coded the amoeba optimization demo program as a single C# console application. I used Visual Studio 2010 (any version of Visual Studio should work) and created a new project named AmoebaOptimization. After the project loaded, in the Solution Explorer window I renamed file Program.cs to the more descriptive AmoebaProgram.cs, which automatically renamed class Program. I deleted all unneeded template-generated using statements except for the single statement that references the top-level System namespace.

The entire program structure, with some comments and WriteLine statements removed, is listed in **Figure 3**.

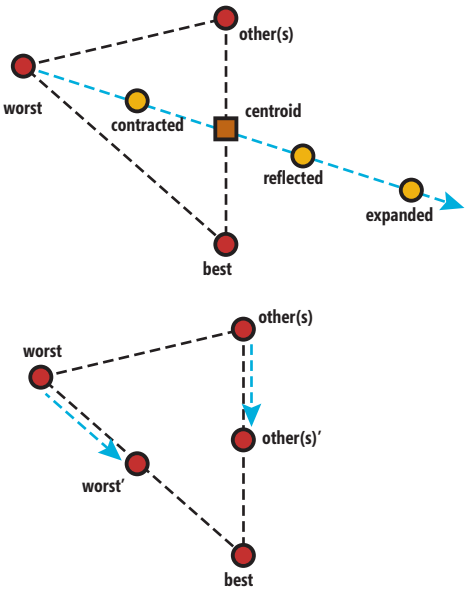


Figure 2 **Amoeba Optimization Primitive Operations**

The Objective Function

Amoeba method optimization is most often used to solve a numerical minimization problem. The function to minimize is generally called a cost function or objective function. The demo program in **Figure 1** is solving a dummy mathematical benchmark problem called Rosenbrock's function. The function has two input variables, x and y, and is defined as $f(x,y) = 100 * (y - x^2)^2 + (1 - x)^2$. The function has a solution of x = 1.0, y = 1.0, which gives a value of 0.0. **Figure 4** shows a three-dimensional plot of Rosenbrock's function.

In real-world situations, amoeba optimization is used to find the solution to difficult problems that are based on data. For example, suppose you're trying to predict stock market prices. You might come up with a list of factors you believe are predictors, and an equation, but you need to determine a set of numeric constants for the equation that minimize the error on a set of training data that has known results.

The objective function for the demo program is defined in the main program class as:

```
public static double ObjectiveFunction(
  double[] vector, object dataSource)
{
  double x = vector[0];
  double y = vector[1];
  return 100.0 * Math.Pow((y - x * x), 2) +
    Math.Pow(1 - x, 2);
}
```

I use a dummy input parameter named dataSource to indicate that in most situations the objective function depends on some external data source, such as a text file or SQL table. Because the function is declared using the public and static modifiers, the function is visible to all code in the demo program.

The Solution Class

Amoeba optimization maintains a collection of possible solutions, defined in a Solution class:

```
public class Solution : IComparable<Solution>
{
  public double[] vector;
  public double value;

  static Random random = new Random(1);

  public Solution(int dim, double minX, double maxX) { . . }
  public Solution(double[] vector) { . . }
  public int CompareTo(Solution other) { . . }
  public override string ToString() { . . }
}
```

The class derives from the IComparable interface so that Solution objects can be automatically sorted. A Solution object has just two significant fields: one is an array of double named vector that holds the numeric values of the solution, and the other is the value of the objective function. I use public scope member fields

Figure 3 **Amoeba Optimization Program Structure**

```
using System;
namespace AmoebaOptimization
{
  class AmoebaProgram
  {
    static void Main(string[] args)
    {
      try
      {
        Console.WriteLine("\nBegin amoeba method optimization demo\n");

        int dim = 2;  // problem dimension
        int amoebaSize = 3;  // number potential solutions
        double minX = -10.0;
        double maxX = 10.0;
        int maxLoop = 50;

        Console.WriteLine("Creating amoeba with size = " + amoebaSize);
        Amoeba a = new Amoeba(amoebaSize, dim, minX, maxX, maxLoop);

        Console.WriteLine("\nInitial amoeba is: \n");
        Console.WriteLine(a.ToString());

        Solution sln = a.Solve();

        Console.WriteLine("Final amoeba is: \n");
        Console.WriteLine(a.ToString());

        Console.WriteLine("\nBest solution found: \n");
        Console.WriteLine(sln.ToString());

        Console.WriteLine("\nEnd amoeba method optimization demo\n");
      }
      catch (Exception ex)
      {
        Console.WriteLine(ex.Message);
      }
    }

    public static double ObjectiveFunction(
      double[] vector, object dataSource) { . . }
  }

  public class Solution : IComparable<Solution> { . . }
  public class Amoeba { . . }
}
```

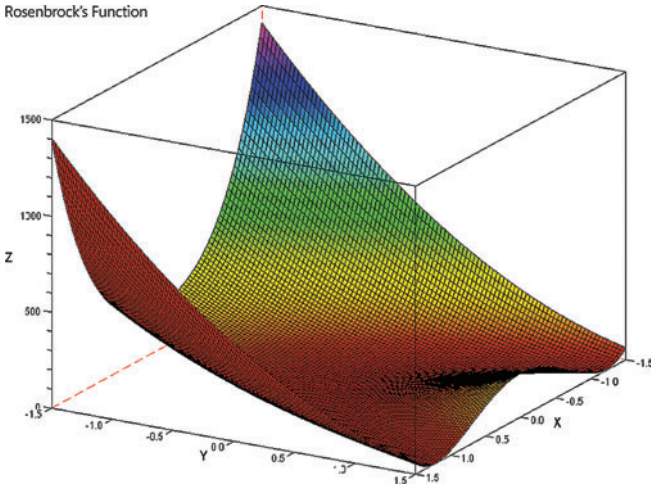


Figure 4 **Graph of the Objective Function**

and remove all error checking for simplicity. The static Random object allows the code to generate random solutions.

The first Solution constructor creates a random solution:

```
public Solution(int dim, double minX, double maxX)
{
  this.vector = new double[dim];
  for (int i = 0; i < dim; ++i)
  this.vector[i] = (maxX - minX) * random.NextDouble() + minX;
  this.value = AmoebaProgram.ObjectiveFunction(this.vector, null);
}
```

The constructor accepts a problem dimension, and limits for each vector component. The dimension for the demo program is 2 because Rosenbrock's function has two input variables, x and y. After allocating space for member field vector, the constructor assigns random values between minX and maxX to the vector array, and then calls the globally accessible objective function to compute the value field.

The second Solution constructor creates a solution from a specified array of double:

```
public Solution(double[] vector)
{
  this.vector = new double[vector.Length];
  Array.Copy(vector, this.vector, vector.Length);
  this.value = AmoebaProgram.ObjectiveFunction(this.vector, null);
}
```

Because the Solution class derives from the IComparable interface, the class must implement a CompareTo method. CompareTo is defined so that Solution objects will be automatically sorted from best (smaller) to worst (larger) values of the objective function:

```
public int CompareTo(Solution other)
{
  if (this.value < other.value) return -1;
  else if (this.value > other.value) return 1;
  else return 0;
}
```

For visualization and debugging purposes, the Solution class defines a simple ToString method using string concatenation:

```
public override string ToString()
{
  string s = "[ ";
  for (int i = 0; i < this.vector.Length; ++i) {
    if (vector[i] >= 0.0) s += " ";
    s += vector[i].ToString("F2") + " ";
  }
  s += "]  val = " + this.value.ToString("F4");
  return s;
}
```

The Amoeba Class

The Amoeba class is essentially an array of Solution objects plus a Solve method that uses the amoeba method algorithm. The structure of the Amoeba class is listed in **Figure 5**.

I declare all fields and methods using public scope for simplicity and for easier debugging during development. The amoebaSize field specifies the number of potential solutions in the Amoeba object. By far the most common value is 3, but you may want to experiment with larger values. The dim field represents the number of variables in the objective function that must be solved for—two in the case of Rosenbrock's function.

Array solutions holds the potential Solution objects. Although it's not clear from the declaration, array solutions must be sorted at all times, from best solution (smallest value field) to worst solution. Fields minX and maxX constrain the initial values in each Solution object. These values will vary from problem to problem.

Fields alpha, beta and gamma are constants that are used by helper methods called by the Solve method. Field maxLoop limits the number of iterations of the processing loop in Solve.

The single Amoeba constructor creates an array of amoebaSize Solution objects, each of which has a vector field of size dim. All of the work is performed by method Solve; all the other methods in class Amoeba are helper methods.

The Amoeba constructor is defined as:

```
public Amoeba(int amoebaSize, int dim, double minX, double maxX, int maxLoop)
{
  this.amoebaSize = amoebaSize;
  this.dim = dim;
  this.minX = minX; this.maxX = maxX;
  this.alpha = 1.0; this.beta = 0.5; this.gamma = 2.0;
  this.maxLoop = maxLoop;

  this.solutions = new Solution[amoebaSize];
  for (int i = 0; i < solutions.Length; ++i)
    solutions[i] = new Solution(dim, minX, maxX);

  Array.Sort(solutions);
}
```

Figure 5 **The Amoeba Class**

```
public class Amoeba
{
  public int amoebaSize;  // Number of solutions
  public int dim;         // Problem dimension
  public Solution[] solutions;  // Potential solutions (vector + value)

  public double minX;
  public double maxX;

  public double alpha;  // Reflection
  public double beta;   // Contraction
  public double gamma;  // Expansion

  public int maxLoop;   // Limits main solving loop

  public Amoeba(int amoebaSize, int dim, double minX,
    double maxX, int maxLoop) { . . }
  public Solution Centroid() { . . }
  public Solution Reflected(Solution centroid) { . . }
  public Solution Expanded(Solution reflected, Solution centroid) { . . }
  public Solution Contracted(Solution centroid) { . . }
  public void Shrink() { . . }
  public void ReplaceWorst(Solution newSolution) { . . }
  public bool IsWorseThanAllButWorst(Solution reflected) { . . }
  public Solution Solve() { . . }
  public override string ToString() { . . }
}
```

Figure 6 **The Amoeba Optimization Solve Algorithm**

```
generate amoebaSize random solutions
while not done loop
  compute centroid
  compute reflected
  if reflected is better than best solution then
    compute expanded
    replace worst solution with better of reflected, expanded
  else if reflected is worse than all but worst then
    if reflected is better than worst solution then
      replace worst solution with reflected
    end if
    compute contracted
    if contracted is worse than worst
      shrink the amoeba
    else
      replace worst solution with contracted
    end if
  else
    replace worst solution with reflected
  end if
end loop
return best solution found
```

Fields alpha, beta and gamma control the behavior of the Solve method and are assigned hardcoded values of 1.0, 0.5 and 2.0, respectively. Research has shown that these values generally give good results, but you might want to experiment. After array solutions has been allocated, a random Solution object is assigned to each cell. The Array.Sort method sorts the solutions from best value to worst.

The Amoeba class has a simple ToString method for visualization and easier debugging:

```
public override string ToString()
{
  string s = "";
  for (int i = 0; i < solutions.Length; ++i)
    s += "[" + i + "] " + solutions[i].ToString() +
      Environment.NewLine;
  return s;
}
```

The Algorithm Primitives

A key aspect of the amoeba optimization algorithm is that the current worst solution is replaced—if it leads to a better set of solutions—by a so-called reflected point, expanded point or contracted point.

Amoeba class helper method Centroid creates a Solution object that is in some sense a middle solution between all solutions in the amoeba except for the worst solution (the worst solution is the one with the largest solution value because the goal is to minimize the objective function, and it will be located at index amoebaSize-1):

```
public Solution Centroid()
{
  double[] c = new double[dim];
  for (int i = 0; i < amoebaSize - 1; ++i)
    for (int j = 0; j < dim; ++j)
      c[j] += solutions[i].vector[j];  // Accumulate sum of each component

  for (int j = 0; j < dim; ++j)
    c[j] = c[j] / (amoebaSize - 1);

  Solution s = new Solution(c);
  return s;
}
```

Helper method Reflected creates a Solution object that's in the general direction of better solutions. Constant alpha, typically set to 1.0, controls how far away from the centroid to move to yield the reflected solution. Larger values of alpha generate reflected points that are farther from the centroid:

```
public Solution Reflected(Solution centroid)
{
  double[] r = new double[dim];
  double[] worst = this.solutions[amoebaSize - 1].vector;  // Convenience only
  for (int j = 0; j < dim; ++j)
    r[j] = ((1 + alpha) * centroid.vector[j]) - (alpha * worst[j]);
  Solution s = new Solution(r);
  return s;
}
```

Helper method Expanded creates a Solution object that's even farther from the centroid than the reflected solution. Constant gamma, typically set to 2.0, controls how far the reflected point is from the centroid:

```
public Solution Expanded(Solution reflected, Solution centroid)
{
  double[] e = new double[dim];
  for (int j = 0; j < dim; ++j)
    e[j] = (gamma * reflected.vector[j]) + ((1 - gamma) * centroid.vector[j]);
  Solution s = new Solution(e);
  return s;
}
```

Helper method Contracted creates a Solution object that's roughly in between the worst solution and the centroid. Constant beta, typically set to 0.50, controls how close to the worst solution the contracted point is:

```
public Solution Contracted(Solution centroid)
{
  double[] v = new double[dim];  // Didn't want to reuse 'c' from centroid routine
  double[] worst = this.solutions[amoebaSize - 1].vector;  // Convenience only
  for (int j = 0; j < dim; ++j)
    v[j] = (beta * worst[j]) + ((1 - beta) * centroid.vector[j]);
  Solution s = new Solution(v);
  return s;
}
```

Figure 7 **The Solve Method**

```
public Solution Solve()
{
  int t = 0;  // Loop counter
  while (t < maxLoop)
  {
    ++t;
    Solution centroid = Centroid();
    Solution reflected = Reflected(centroid);

    if (reflected.value < solutions[0].value)
    {
      Solution expanded = Expanded(reflected, centroid);
      if (expanded.value < solutions[0].value)
        ReplaceWorst(expanded);
      else
        ReplaceWorst(reflected);
      continue;
    }

    if (IsWorseThanAllButWorst(reflected) == true)
    {
      if (reflected.value <= solutions[amoebaSize - 1].value)
        ReplaceWorst(reflected);

      Solution contracted = Contracted(centroid);

      if (contracted.value > solutions[amoebaSize - 1].value)
        Shrink();
      else
        ReplaceWorst(contracted);

      continue;
    }

    ReplaceWorst(reflected);

  } // loop
  return solutions[0];  // Best solution
}
```

Helper method ReplaceWorst replaces the current worst solution, located at index amoebaSize-1, with a different solution (the reflected, expanded or contracted point):

```
public void ReplaceWorst(Solution newSolution)
{
  for (int j = 0; j < dim; ++j)
    solutions[amoebaSize-1].vector[j] = newSolution.vector[j];
  solutions[amoebaSize - 1].value = newSolution.value;
  Array.Sort(solutions);
}
```

If neither the reflected, nor the expanded, nor the contracted point gives a better set of solutions, the amoeba algorithm shrinks the current solution set. Every solution point, except for the best point at index 0, is moved halfway from its current location toward the best point:

```
public void Shrink()
{
  for (int i = 1; i < amoebaSize; ++i)  // start at [1]
  {
    for (int j = 0; j < dim; ++j) {
      solutions[i].vector[j] =
        (solutions[i].vector[j] + solutions[0].vector[j]) / 2.0;
      solutions[i].value = AmoebaProgram.ObjectiveFunction(
        solutions[i].vector, null);
    }
  }
  Array.Sort(solutions);
}
```

The Solve Method

The amoeba optimization solve algorithm is given in **Figure 6**, in high-level pseudo-code.

Even though the algorithm is short, it's a bit trickier than it first appears and you'll likely have to examine it very closely if you want to modify it for some reason. Helper method IsWorseThanAllButWorst makes the Solve method quite a bit neater. The helper examines a Solution object and returns true only if the Solution object (always the reflected solution in the algorithm) is worse (has a greater objective function value) than all other solutions in the amoeba, except for possibly the worst solution (located at index amoebaSize-1):

```
public bool IsWorseThanAllButWorst(Solution reflected)
{
  for (int i = 0; i < amoebaSize - 1; ++i) {
    if (reflected.value <= solutions[i].value)
// Found worse solution
      return false;
  }
  return true;
}
```

With all the helper methods in place, the Solve method, listed in **Figure 7**, is rather short. The processing loop in Solve exits after maxLoop iterations. In general, a good value of maxLoop varies from problem to problem and must be determined through trial and error. An alternative or additional stopping condition is to exit the processing loop when the average error of the solutions in the amoeba drops below some problem-dependent value.

Customizing the Code

The example code and explanation presented in this article should get you up and running if you want to experiment with or use amoeba method optimization in a software system. There are several modifications you might want to consider. In the main algorithm loop, before computing the centroid and reflected solutions, I often compute a purely random solution and check to see if this random solution is better than the current worst solution. This approach helps prevent the optimization algorithm from getting stuck in a local minimum solution.

Another customization possibility is to compute multiple reflected points. Instead of a single reflected point that lies on the line between the current worst solution and the centroid, you can compute additional reflected points that lie on different lines. This approach also helps avoid local minima traps. ■

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 experts for reviewing this article: *Darren Gehring (Microsoft) and Mark Marron (Microsoft)*

RACHEL APPEL

Use TypeScript in Modern Apps

The original intent of JavaScript was for Document Object Model (DOM) manipulation in a small DOM tree. Over time, however, JavaScript has become so popular that it's now a mainstream language for any kind of app, from small marketplace apps to apps for the enterprise. As the popularity of JavaScript continues to grow, a rise in the number of tools and languages required to support its developers is inevitable, with TypeScript being one such language.

What Is TypeScript and How Does It Work?

TypeScript is a superset of JavaScript that allows you to write and generate JavaScript code that acts more strongly typed and object-oriented, but retains the flexibility that developers love (or sometimes hate) about JavaScript. TypeScript boosts the viable range of use of JavaScript into the realm of enterprise apps, Web sites and apps where JavaScript has historically run amok due to lack of tools in this space.

Tsc.exe, an open source TypeScript compiler/code generator, is available for download at typescriptlang.org. TypeScript is a standalone compiler, so you can open up a command prompt and execute tsc.exe with the proper arguments at any time, like so:

```
tsc.exe --out outputfile.js inputfile.ts
```

You write TypeScript code, then spin it through the compiler and out comes production JavaScript. Although TypeScript is a code generator, it doesn't output unnecessary code (as is often done for the sake of a visual design tool), mangle variable names or change the variable order. This means it's easier to debug the final product because it's straight-up JavaScript.

JavaScript is already an object-oriented language, but its prototypal syntax is off-putting to many developers. To solve this problem, TypeScript adds features to JavaScript such as classes and interfaces, which are proposed features of the ECMAScript 6 (ES6) standard. This makes TypeScript a code generator covered in syntactic sugar that in most cases cuts down the amount of JavaScript to maintain. For example, the following code uses the prototypal syntax:

```
function Animal(name, species, habitat) {
  this.name = name;
  this.species = species;
  this.habitat = habitat;
}
Animal.prototype.sayHello = function(){
  console.log("RAWR!");
}
var animal =
  new Animal("Fluffy", " Velociraptor ", "Everywhere. Run and hide.");
animal.sayHello();
```

The preceding sample starts with a constructor function, a heavily used JavaScript pattern without the surrounding class definition that you'd normally see in other object-oriented languages. You define what is similar to instance members of classes inside constructor functions by using the *this* keyword. Outside the constructor function lies the actual prototype method that binds JavaScript methods to classes. Classes in TypeScript allow you to write the same code as in the preceding sample but with a more natural syntax, as shown in **Figure 1**.

To many developers the TypeScript code sample in **Figure 1** is more readable than the traditional JavaScript equivalent. Clearly, the code serves as a class definition and list of members, and it shows the types of arguments. TypeScript also provides type checking, interfaces, static compile-time checking, lambda-style expressions and goodies usually found in compiled—not interpreted—languages. These extensions to the JavaScript language are beneficial, as they keep you from falling into common coding pitfalls.

> You write TypeScript code, then spin it through the compiler and out comes production JavaScript.

Other common JavaScript challenges arise when too many variables that have public scope exist in the JavaScript global namespace, causing global namespace pollution (which seems to happen all too frequently). Fortunately, TypeScript helps in this case because it implements modules that behave like namespaces and create closures that prevent global clutter. Modules in TypeScript come in two flavors: internal and external. Internal modules contain code declared in the current file, and you must import external modules by adding ///<reference path='path/reference-file.ts' /> to the top of the current code file. You can declare modules with the module keyword and need only curly brackets to close. A TypeScript module looks something like this:

```
module outerModule {
  "use strict";
  module innerModule {
    export function aFunction { s: string };
    export var variable = 1;
  }
}
```

Figure 1 **A TypeScript Class**

```
class Animal
{
  name: string;
  species: string;
  habitat: string;
  constructor(name: string, species: string, habitat: string)
  {
    this.name = name;
    this.species = species;
    this.habitat = habitat;
  }
  sayhello()
  {
    Console.log("RAWR");
  }
}
```

And the emitted JavaScript looks like this:

```
var outerModule;
(function (outerModule) {
  "use strict";
  var innerModule;
  (function (innerModule) {
    function aFunction() { s: string }
    innerModule.aFunction = aFunction;
    innerModule.variable = 1;
  })(innerModule || (innerModule = {}));
})(outerModule || (outerModule = {}));
```

The preceding code creates a singleton module instance accessible from anywhere in a Windows Store app in the outerModule namespace. As you can see, modules render as anonymous functions (that is, Immediately Invoked Function Expressions, or IIFEs). Any member marked with the *export* directive has global scope, which is equivalent to C# members marked with the *internal* keyword (that is, project-wide).

Configure and Build Windows Store Apps with TypeScript

Tight integration exists between TypeScript and Visual Studio, but TypeScript ships separately, so you need to install the following tools in addition to Visual Studio 2012 Express, Pro or Ultimate editions:

- TypeScript (bit.ly/QHxbEZ)
- Web Essentials for Visual Studio (bit.ly/MpJlHh)

> Tight integration exists between TypeScript and Visual Studio, but TypeScript ships separately.

After installing the extensions, you can find a TypeScript project template for Visual Studio located just underneath the JavaScript node in the New Project dialog box. This particular built-in template is an HTML client Web app template with the appropriate TypeScript assets built in, so you need do nothing else for this template to work.

Even with tight integration between Visual Studio and TypeScript, at the time of this writing there are no built-in Windows Store project templates with TypeScript (only the client Web template previously mentioned); however, the TypeScript documentation says they're coming soon. In the meantime, if you're building Windows Store apps with JavaScript, you can use any of the current JavaScript project templates, such as Blank, Grid, Split and more. TypeScript automatically works in them all, but you need to make some minor modifications to the project to get things going.

To integrate TypeScript into an existing Windows Store app, you need to copy the following declaration files to a folder, for example <project-root>\tslib:

- lib.d.ts
- winjs.d.ts
- winrt.d.ts

These files are available at the TypeScript download page at typescript.codeplex.com. Notice that the file extensions for the listed files end in .d.ts, in which the "d" stands for declaration. These files contain type declarations for popular frameworks like jQuery or the native Windows Runtime (WinRT) and Windows Library for JavaScript (WinJS) libraries. **Figure 2** contains a sample of the winjs.d.ts file, outlining commonly used WinJS methods. Notice the file is full of public declarations used by Visual Studio or other tools for compile-time checks. Because TypeScript at this point is still somewhat immature, there might be some missing, but you can add them in yourself.

If you write WinJS apps you should be quite familiar with these method stubs, including WinJS.Binding.List and WinJS.xhr objects—all the WinRT/WinJS library stubs are at your disposal. These definition files allow IntelliSense to work in Visual Studio.

Figure 2 **Examining the winjs.d.ts Definition File**

```
declare module WinJS {
  export function strictProcessing(): void;
  export module Binding {
    export function as(data: any): any;
    export class List {
      constructor (data: any[]);
      public push(item: any): any;
      public indexOf(item: any): number;
      public splice(index: number, count: number, newelems: any[]): any[];
      public splice(index: number, count: number): any[];
      public splice(index: number): any[];
      public createFiltered(predicate: (x: any) => bool): List;
      public createGrouped(keySelector: (x: any) => any, dataSelector:
        (x: any) => any): List;
        public groups: any;
        public dataSource: any;
        public getAt: any;
      }
      export var optimizeBindingReferences: bool;
  }
  export module Namespace {
    export var define: any;
    export var defineWithParent: any;
  }
  export module Class {
    export function define(constructor: any, instanceMembers: any): any;
    export function derive(
      baseClass: any, constructor: any, instanceMembers: any): any;
    export function mix(constructor: any, mixin: any): any;
  }
  export function xhr(options: { type: string; url: string; user: string;
    password: string; headers: any; data: any;
    responseType: string; }): WinJS.Promise;
  export module Application {
    export interface IOHelper {
      exists(filename: string): bool;
      readText(fileName: string, def: string): WinJS.Promise;
      readText(fileName: string): WinJS.Promise;
      writeText(fileName: string, text: string): WinJS.Promise;
      remove(fileName: string): WinJS.Promise;
    }
// More definitions
```

Adding a .ts file to any folder in the project causes Visual Studio to automatically create corresponding .js and .min.js (minified) companion files. TypeScript rebuilds these files each time you save a .ts file in Visual Studio.

In most of the Windows Store JavaScript templates a folder named *pages* contains subfolders with the collective .html, .css and .js assets needed for each page. In addition to these files are more JavaScript files in the \js folder, such as data.js, default.js and navigator.js. You must integrate TypeScript into these files by performing the following steps for each:

1. Add declaration references to the top of each file, for example, ///<reference path='path/reference-file.ts' />.
2. Rename the .js files to a .ts extension.
3. Modify existing JavaScript to corresponding TypeScript language constructs, that is, modules, classes, declarations and so on.

For example, to integrate \js\data.js, you need to insert references at the top of the file and convert the top-level function to a module like the code in **Figure 3**. If you rename data.js to data.ts, Visual Studio will create the corresponding .js and mapping files on file save.

The code in **Figure 3** acts as a top-level namespace (module) using the project's name, TypeScriptApp, sticking with standard conventions familiar to Visual Studio users.

Of course, you can always leave the current JavaScript from the templates unchanged and the code will still run as expected, but it isn't consistent in its style or syntax, making maintenance difficult.

Visual Studio Options for TypeScript

There are settings you need to know about for optimal use of TypeScript, especially in Windows Store apps. Because there's no need for the minified .js files in Windows Store apps, you can change the "Minify generated JavaScript" option to False in the Web Essentials tab from the Visual Studio Tools | Options dialog box and delete any that might exist. Minified files improve performance in Web sites only—not client apps—because they decrease overall required bandwidth across the Internet.

Another necessary change for use with TypeScript in Windows Store apps is to set encoding to "Re-save JS with UTF-8 BOM" (see bit.ly/ceKkYq) in the Tools | Options dialog. Setting to UTF-8 BOM (byte order mark) makes the app perform better at startup, adhering to Windows Store process lifecycle management guidelines (see my Windows 8 Special Issue column, "The Windows Store App Lifecycle," at msdn.microsoft.com/magazine/jj660301 for more on these guidelines). In addition to performance, this encoding spec is required for passing Windows Store certification and publishing your app.

Figure 3 **TypeScript Transformation from data.js to data.ts**

```
// Original code in data.js
(function () {
  "use strict";
  var list = new WinJS.Binding.List();
  var groupedItems = list.createGrouped(
    function groupKeySelector(item) { return item.group.key; },
    function groupDataSelector(item) { return item.group; }
  );
  // TODO: Replace the data with your real data
  // You can add data from asynchronous sources
  // whenever it becomes available
  generateSampleData().forEach(function (item) {
    list.push(item);
  });
  // ... More data-access code
})();

// The modified data.ts file
///<reference path='../ts/winjs.d.ts' />
///<reference path='../ts/winrt.d.ts' />
  module TypeScriptApp {
    "use strict";
    var list = new WinJS.Binding.List();
    var groupedItems = list.createGrouped(
      function groupKeySelector(item) { return item.group.key; },
      function groupDataSelector(item) { return item.group; }
  );
  // TODO: Replace the data with your real data.
  // You can add data from asynchronous sources whenever it becomes available
  generateSampleData().forEach(function (item) {
    list.push(item);
  });
  // ... More data-access code
}
```

> If you're building Windows Store apps with JavaScript, you can use any of the current JavaScript project templates.

JavaScript developers know that tools like source maps are a necessity for debugging deployed code or source code from JavaScript generators. That's because source maps are files that map code to other code, often between standard-sized development files and minified and combined production files that aren't easily debugged. Set the "Generate Source Map" to True in the Visual Studio options to create source maps between TypeScript and JavaScript to enable TypeScript debugging. This option applies the --sourcemap compiler switch, which in turn creates the maps at compile time.

The TypeScript compiler by default compiles to ECMAScript 3 (ES3)-compliant code; however, you can compile to ECMAScript 5 (ES5) if you change the "Compile to ECMAScript 3" option to False in Visual Studio, which sets the tsc compiler flag --target to generate ES5 code. Developers who want to use property-style syntax or any ES5 or proposed ES6 TypeScript features should set this switch.

Extra Benefits

JavaScript is here to stay and it's more popular than ever, so those who move to JavaScript from the land of compiled languages now have TypeScript to help them write and manage application-scale JavaScript code. JavaScript developers benefit from extra type checking and compiler services to which they don't normally have access when writing straight-up JavaScript. ■

Rachel Appel *is a developer evangelist at Microsoft New York City. Reach her via her Web site at rachelappel.com or by e-mail at rachel.appel@microsoft.com. You can also follow her latest updates on Twitter at twitter.com/rachelappel.*

Thanks to the following technical expert for reviewing this article: *Christopher Bennage (Microsoft)*

79% of mobile app developers plan to **integrate HTML5** in the near future.

75% of North American Internet users use browsers that are mostly **HTML5 compatible**, up from 57% in 2011

7,412,919 websites are using the HTML5 DocType

HTML

By 2016...

MORE THAN 50% of Mobile Apps will be **Hybrid**

63% of mobile developers are "**very interested**" in using **HTML5** to build their apps as of Q4 2012

Don't get left behind: infragistics.com/**igniteui**

IGNITEUI™
INFRAGISTICS JQUERY CONTROLS

Stats source: infragistics.com/sources

Copyright 1996-2013 Infragistics, Inc. All rights reserved. Infragistics and NetAdvantage are registered trademarks of Infragistics, Inc. The Infragistics logo is a trademark of Infragistics, Inc. All other trademarks or registered trademarks are the respective property of their owners.

INFRAGISTICS DESIGN / DEVELOP / EXPERIENCE

CHARLES PETZOLD

An Introduction to Audio Processing Objects

The XAudio2 component of DirectX is much more than just a way to play sounds and music in a Windows 8 application. I've come to view it rather as a versatile construction set of sound processing. Through the use of multiple IXAudio2SourceVoice and IXAudio2SubmixVoice instances, programmers can split sounds into separate pipelines for customized processing, and then combine them to merge in the final IXAudio2MasteringVoice.

As I demonstrated in the previous installment of this column (msdn.microsoft.com/magazine/dn198248), XAudio2 allows standard audio filters to be applied to source and submix voices. These filters attenuate frequency ranges and consequently alter the harmonic content and timbre of the sounds.

But much more powerful is a generalized facility that provides access to the actual audio streams passing through the voices. You can simply analyze this audio stream, or modify it.

This facility is known informally as an "audio effect." More formally it involves the creation of an Audio Processing Object (APO), also known as a cross-platform APO, or XAPO, when it can be used with Xbox 360 applications as well as Windows.

XAudio2 includes two predefined APOs for common tasks. The XAudio2CreateVolumeMeter function creates an APO that allows a program to dynamically obtain the peak amplitude of an audio stream at intervals convenient to the application. The XAudio2CreateReverb function creates an APO that applies echo or reverberation to a voice based on 23 runtime parameters—and by "runtime" in this context I mean parameters that can be dynamically changed while the APO is actively processing audio. In addition, a library known as XAPOFX provides echo and reverberation effects, as well as a volume limiter and a four-band equalizer.

> APOs can be applied to any type of XAudio2 voice.

An APO implements the IXAPO interface, and an APO with runtime parameters implements the IXAPOParameters interface. But an easier approach to creating your own APOs involves deriving from the CXAPOBase and CXAPOParametersBase classes, which implement these interfaces and handle much of the overhead.

Code download available at archive.msdn.microsoft.com/mag201306DXF.

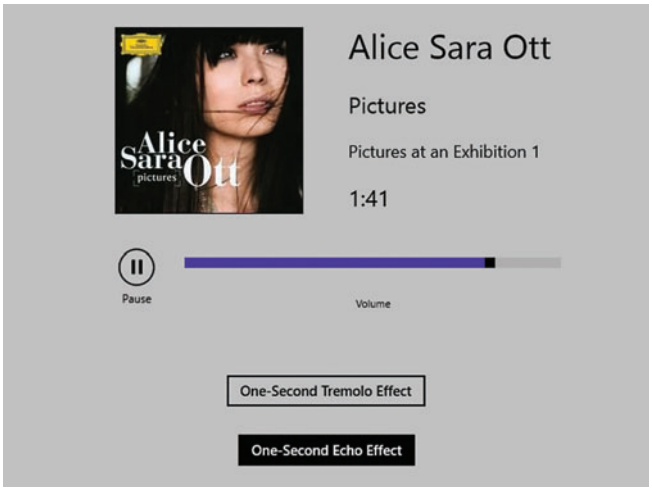


Figure 1 **The SimpleEffectDemo Program with Two Audio Effects**

Deriving from these two classes is the strategy I'll be using in this article. In addition to the other header files and important libraries I've discussed in previous columns, projects that implement APOs need a reference to the xapobase.h header file and the xapobase.lib import library.

Using Audio Processing Objects

Before discussing the internals of an APO class, let me show you how to apply the effects to XAudio2 voices. The SimpleEffectDemo project in the downloadable code for this column allows you to load a file from your Windows 8 music library and play it. It's similar to code I've shown in previous columns: The file is loaded and decoded using Media Foundation classes, and played with XAudio2. SimpleEffectDemo creates only two XAudio2 voices: a source voice for generating the audio and the required mastering voice that funnels the audio to the sound hardware.

SimpleEffectDemo also contains two non-parametered CXAPOBase derivatives called OneSecondTremoloEffect (which applies a tremolo, or wavering of volume, based on simple amplitude modulation) and OneSecondEchoEffect. **Figure 1** shows the program running with a loaded music file. Each of the two effects is enabled or disabled by a ToggleButton. The screenshot shows the echo effect enabled but the tremolo effect disabled.

APOs can be applied to any type of XAudio2 voice. When applied to source voices or submix voices, the audio processing occurs *after* the built-in filters you set with SetFilterParameters,

Figure 2 **Applying Two Audio Effects to a Mastering Voice**

```
// Create tremolo effect
ComPtr<OneSecondTremoloEffect> pTremoloEffect = new OneSecondTremoloEffect();

// Create echo effect
ComPtr<OneSecondEchoEffect> pEchoEffect = new OneSecondEchoEffect();

// Reference those effects with an effect descriptor array
std::array<XAUDIO2_EFFECT_DESCRIPTOR, 2> effectDescriptors;

effectDescriptors[0].pEffect = pTremoloEffect.Get();
effectDescriptors[0].InitialState = tremoloToggle->IsChecked->Value;
effectDescriptors[0].OutputChannels = 2;

effectDescriptors[1].pEffect = pEchoEffect.Get();
effectDescriptors[1].InitialState = echoToggle->IsChecked->Value;
effectDescriptors[1].OutputChannels = 2;

// Reference that array with an effect chain
XAUDIO2_EFFECT_CHAIN effectChain;
effectChain.EffectCount = effectDescriptors.size();
effectChain.pEffectDescriptors = effectDescriptors.data();
hresult = pMasteringVoice->SetEffectChain(&effectChain);

if (FAILED(hresult))
  throw ref new COMException(hresult, "pMasteringVoice->SetEffectChain failure");
```

but before the filters applied to audio sent to other voices using SetOutputFilterParameters.

I've chosen to apply these two effects to the mastering voice. The code that instantiates the effects and attaches them to the mastering voice is shown in **Figure 2**. Each effect is referenced with an XAUDIO2_EFFECT_DESCRIPTOR. If there's more than one effect (as is the case here), you use an array of such structures. That structure (or array) is then referenced by an XAUDIO2_EFFECT_CHAIN structure, which is passed to the SetEffectChain method supported by all XAudio2 voices. The order of the effects matters: In this case the echo effect will get an audio stream that already has the tremolo effect applied.

After the SetEffectChain call, the effect instances should not be further referenced by the program. XAudio2 has already added a reference to these instances, and the program can release its own copies, or ComPtr can do that for you. From here on, the effects are identified by indices—in this case 0 for the tremolo effect and 1 for the echo effect. You might want to use an enumeration for those constants.

For both of the effects, I've set the InitialState field of the XAUDIO2_EFFECT_DESCRIPTOR to a ToggleButton checked status. This governs whether the effect is initially enabled or disabled. The effects are later enabled and disabled by the Checked and Unchecked handlers for the two ToggleButton controls, as shown in **Figure 3**.

Figure 3 **Enabling and Disabling Audio Effects**

```
void MainPage::OnTremoloToggleChecked(Object^ sender, RoutedEventArgs^ args)
{
  EnableDisableEffect(safe_cast<ToggleButton^>(sender), 0);
}

void MainPage::OnEchoToggleChecked(Object^ sender, RoutedEventArgs^ args)
{
  EnableDisableEffect(safe_cast<ToggleButton^>(sender), 1);
}

void MainPage::EnableDisableEffect(ToggleButton^ toggle, int index)
{
  HRESULT hresult = toggle->IsChecked->Value ? pMasteringVoice->EnableEffect(index) :
                                               pMasteringVoice->DisableEffect(index);
  if (FAILED(hresult))
    throw ref new COMException(hresult, "pMasteringVoice->Enable/DisableEffect " +
                                        index.ToString());
}
```

Instantiation and Initialization

Both OneSecondTremoloEffect and OneSecondEchoEffect derive from CXAPOBase. Perhaps the first puzzlement you'll encounter when deriving from this class is dealing with the CXAPOBase constructor. This constructor requires a pointer to an initialized XAPO_REGISTRATION_PROPERTIES structure, but how does this structure get initialized? C++ requires that a base class constructor complete before any code in the derived class is executed.

This is a bit of a quandary, which you can solve by defining and initializing the structure as a global variable, or a static field, or within a static method. I prefer the static field approach in this case, as you can see in the OneSecondTremoloEffect.h header file in **Figure 4**.

The RegistrationProperties field is initialized in the code file (coming up shortly). A pointer to it is passed to the CXAPOBase constructor. Very often a CXAPOBase derivative will also define a field of type WAVEFORMATEX (as this one does) or WAVEFORMATEXTENSIBLE (in the general case) for saving the waveform format of the audio stream passing through the effect.

Notice also the __declspec ("declaration specifier") at the bottom of the file that associates the OneSecondTremoloEffect class with a GUID. You can generate a GUID for your own effects classes from the Create GUID option on the Tools menu in Visual Studio.

A CXAPOBase derivative must override the Process method and usually overrides the LockForProcess method as well. The LockForProcess method allows the APO to perform initialization based on a particular audio format, which includes the sampling rate, the number of channels and the sample data type. The Process method actually performs the analysis or modification of the audio data.

Figure 4 **The OneSecondTremoloEffect.h Header File**

```
#pragma once

class OneSecondTremoloEffect sealed : public CXAPOBase
{
private:
  static const XAPO_REGISTRATION_PROPERTIES RegistrationProps;
  WAVEFORMATEX waveFormat;
  int tremoloIndex;

public:
  OneSecondTremoloEffect() : CXAPOBase(&RegistrationProps),
                             tremoloIndex(0)
  {
  }

protected:
  virtual HRESULT __stdcall LockForProcess(
    UINT32 inpParamCount,
    const XAPO_LOCKFORPROCESS_BUFFER_PARAMETERS  *pInpParams,
    UINT32 outParamCount,
    const XAPO_LOCKFORPROCESS_BUFFER_PARAMETERS  *pOutParam) override;

  virtual void __stdcall Process(
    UINT32 inpParameterCount,
    const XAPO_PROCESS_BUFFER_PARAMETERS *pInpParams,
    UINT32 outParamCount,
    XAPO_PROCESS_BUFFER_PARAMETERS *pOutParams,
    BOOL isEnabled) override;
};

class __declspec(uuid("6FB2EBA3-7DCB-4ADF-9335-686782C49911"))
                      OneSecondTremoloEffect;
```

Figure 5 **The OneSecondTremoloEffect.cpp File**

```
#include "pch.h"
#include "OneSecondTremoloEffect.h"

const XAPO_REGISTRATION_PROPERTIES
OneSecondTremoloEffect::RegistrationProps =
{
  __uuidof(OneSecondTremoloEffect),
  L"One-Second Tremolo Effect",
  L"Coded by Charles Petzold",
  1,      // Major version number
  0,      // Minor version number
  XAPOBASE_DEFAULT_FLAG | XAPO_FLAG_INPLACE_REQUIRED,
  1,      // Min input buffer count
  1,      // Max input buffer count
  1,      // Min output buffer count
  1       // Max output buffer count
};

HRESULT OneSecondTremoloEffect::LockForProcess(
  UINT32 inpParamCount,
  const XAPO_LOCKFORPROCESS_BUFFER_PARAMETERS  *pInpParams,
  UINT32 outParamCount,
  const XAPO_LOCKFORPROCESS_BUFFER_PARAMETERS  *pOutParams)
{
  waveFormat = * pInpParams[0].pFormat;

  return CXAPOBase::LockForProcess(inpParamCount, pInpParams,
                                   outParamCount, pOutParams);
}

void OneSecondTremoloEffect::Process(UINT32 inpParamCount,
  const XAPO_PROCESS_BUFFER_PARAMETERS *pInpParams,
  UINT32 outParamCount,
  XAPO_PROCESS_BUFFER_PARAMETERS *pOutParams,
  BOOL isEnabled)
{
  XAPO_BUFFER_FLAGS flags = pInpParams[0].BufferFlags;
  int frameCount = pInpParams[0].ValidFrameCount;
  const float * pSrc = static_cast<float *>(pInpParams[0].pBuffer);
  float * pDst = static_cast<float *>(pOutParams[0].pBuffer);
  int numChannels = waveFormat.nChannels;

  switch(flags)
  {
  case XAPO_BUFFER_VALID:
    for (int frame = 0; frame < frameCount; frame++)
    {
      float sin = 1;

      if (isEnabled)
      {
        sin = fabs(DirectX::XMScalarSin(DirectX::XM_PI * tremoloIndex /
                                        waveFormat.nSamplesPerSec));
        tremoloIndex = (tremoloIndex + 1) % waveFormat.nSamplesPerSec;
      }

      for (int channel = 0; channel < numChannels; channel++)
      {
        int index = numChannels * frame + channel;
        pDst[index] = sin * pSrc[index];
      }
    }
    break;

  case XAPO_BUFFER_SILENT:
    break;
  }

  pOutParams[0].ValidFrameCount = pInpParams[0].ValidFrameCount;
  pOutParams[0].BufferFlags = pInpParams[0].BufferFlags;
}
```

Figure 5 shows these two methods as well as the initialization of the RegistrationProperties field. Notice that the first field of XAPO_REGISTRATION_PROPERTIES is the GUID identified with the class.

In theory, APOs can deal with multiple input buffers and multiple output buffers. However, APOs are currently restricted to one input buffer and one output buffer. This restriction affects the last four fields of the XAPO_REGISTRATION_PROPERTIES structure, and the parameters to the LockForProcess and Process method. For both methods, inpParamCount and outParamCount are always equal to 1, and the pointer arguments always point to just one instance of the indicated structure.

> Most often when working with XAudio2, you're dealing with samples that are 16-bit integers or 32-bit floating-point values.

At the rate of 100 calls per second, the Process method of an APO receives an input buffer of audio data and prepares an output buffer. It's possible for APOs to perform format conversions—for example, to change the sampling rate between the input and output buffer, or the number of channels, or the data type of the samples.

These format conversions can be difficult, so you can indicate in the sixth field of the XAPO_REGISTRATION_PROPERTIES structure which conversions you're not prepared to implement. The XAPOBASE_DEFAULT_FLAG indicates that you don't wish to perform conversions of the sampling rate, the number of channels, the sample bit sizes or the frame sizes (the number of samples in each Process call).

The format of the audio data passing through the APO is available from the parameters to the LockForProcess override in the form of a standard WAVEFORMATEX structure. Commonly, LockForProcess is only called once. Most APOs need to know the sampling rate and number of channels, and it's best to generalize your APO for any possible values.

Also crucial is the data type of the samples themselves. Most often when working with XAudio2, you're dealing with samples that are 16-bit integers or 32-bit floating-point values. Internally, however, XAudio2 prefers using floating-point data (the C++ float type), and that's what you'll see in your APOs. If you'd like, you can verify the sample data type in the LockForProcess method. However, it's also my experience that the wFormatTag field of the WAVEFORMATEX structure does not equal WAVE_FORMAT_IEEE_FLOAT as might be expected. Instead, it's WAVE_FORMAT_EXTENSIBLE (the value 65534), which means that you're really dealing with a WAVEFORMATEXTENSIBLE structure, in which case the SubFormat field indicates the data type KSDATAFORMAT_SUBTYPE_IEEE_FLOAT.

If the LockForProcess method encounters an audio format it can't deal with, it should return an HRESULT indicating an error, perhaps E_NOTIMPL to indicate "not implemented."

Figure 6 **The Process Method in OneSecondEchoEffect**

```
void OneSecondEchoEffect::Process(UINT32 inpParamCount,
  const XAPO_PROCESS_BUFFER_PARAMETERS *pInpParams,
  UINT32 outParamCount,
  XAPO_PROCESS_BUFFER_PARAMETERS *pOutParams,
  BOOL isEnabled)
{
  const float * pSrc = static_cast<float *>(pInpParams[0].pBuffer);
  float * pDst = static_cast<float *>(pOutParams[0].pBuffer);
  int frameCount = pInpParams[0].ValidFrameCount;
  int numChannels = waveFormat.nChannels;
  bool isSourceValid = pInpParams[0].BufferFlags == XAPO_BUFFER_VALID;

  for (int frame = 0; frame < frameCount; frame++)
  {
    for (int channel = 0; channel < numChannels; channel++)
    {
      // Get sample based on XAPO_BUFFER_VALID flag
      int index = numChannels * frame + channel;
      float source = isSourceValid ? pSrc[index] : 0.0f;

      // Combine sample with contents of delay buffer and save back
      int delayBufferIndex = numChannels * delayIndex + channel;
      float echo = 0.5f * source + 0.5f * delayBuffer[delayBufferIndex];
      delayBuffer[delayBufferIndex] = echo;

      // Transfer to destination buffer
      pDst[index] = isEnabled ? echo : source;
    }
    delayIndex = (delayIndex + 1) % delayLength;
  }

  pOutParams[0].BufferFlags = XAPO_BUFFER_VALID;
  pOutParams[0].ValidFrameCount = pInpParams[0].ValidFrameCount;
}
```

Processing the Audio Data

The LockForProcess method can spend whatever time it needs for initialization, but the Process method runs on the audio-processing thread, and it must not dawdle. You'll discover that for a sampling rate of 44,100 Hz, the ValidFrameCount field of the buffer parameters equals 441, indicating that Process is called 100 times per second, each time with 10 ms of audio data. For two-channel stereo, the buffer contains 882 float values with the channels interleaved: left channel followed by right channel.

The BufferFlags field is either XAPO_BUFFER_VALID or XAPO_BUFFER_SILENT. This flag allows you to skip processing if there's no actual audio data coming through. In addition, the isEnabled parameter indicates if this effect has been enabled via the EnableEffect and DisableEffect methods that you've already seen.

Figure 7 **The Static Create Method for OneThirdOctaveEqualizerEffect**

```
OneThirdOctaveEqualizerEffect * OneThirdOctaveEqualizerEffect::Create()
{
  // Create and initialize three effect parameters
  OneThirdOctaveEqualizerParameters * pParameterBlocks =
    new OneThirdOctaveEqualizerParameters[3];

  for (int i = 0; i < 3; i++)
    for (int band = 0; band < 26; band++)
      pParameterBlocks[i].Amplitude[band] = 1.0f;

  // Create the effect
  return new OneThirdOctaveEqualizerEffect(
    &RegistrationProps,
    (byte *) pParameterBlocks,
    sizeof(OneThirdOctaveEqualizerParameters),
    false);
}
```

If the buffer is valid, the OneSecondTremoloEffect APO loops through the frames and the channels, calculates an index for the buffer, and transfers float values from the source buffer (pSrc) to the destination buffer (pDst). If the effect is disabled, a factor of 1 is applied to the source values. If it's enabled, a sine value is applied, calculated using the zippy XMScalarSin function from the DirectX Math library.

At the end of the Process method, the ValidFrameCount and BufferFlags are set on the output parameters structure to the corresponding values of the input parameters structure.

Although the code treats the input and output buffers as separate objects, this is not actually the case. Among the flags you can set in the XAPO_REGISTRATION_PROPERTIES structure are XAPO_FLAG_INPLACE_SUPPORTED (which is included in the XAPOBASE_DEFAULT_FLAG) and XAPO_FLAG_INPLACE_REQUIRED. The word "inplace" means that the pointers to the input and output buffers—called pSrc and pDst in my code—are actually equal. There's only one buffer used for both input and output. You should definitely be aware of that fact when writing your code.

But watch out: It's my experience that if those flags are removed, separate buffers are indeed present, but only the input buffer is valid for both input and output.

Saving Past Samples

The tremolo effect merely needs to alter samples. An echo effect needs to save previous samples because the output of a one-second echo effect is the current audio plus audio from one second ago.

This means that the OneSecondEchoEffect class needs to maintain its own buffer of audio data, which it defines as an std::vector of type float and sizes during the LockForProcess method:

```
delayLength = waveFormat.nSamplesPerSec;
int numDelaySamples = waveFormat.nChannels *
                      waveFormat.nSamplesPerSec;
delayBuffer.resize(numDelaySamples);
```

This delayBuffer vector is sufficient to hold one second of audio data, and it's treated as a revolving buffer. The LockForProcess method initializes the buffer to 0 values, and initializes an index to this buffer:

```
delayIndex = 0;
```

Figure 8 **The Calculation of Equalizer Filter Constants**

```
Q = 4.318f;       // One-third octave

static float frequencies[26] =
{
  20.0f, 25.0f, 31.5f, 40.0f, 50.0f, 63.0f, 80.0f, 100.0f, 125.0f,
  160.0f, 200.0f, 250.0f, 320.0f, 400.0f, 500.0f, 630.0f, 800.0f, 1000.0f,
  1250.0f, 1600.0f, 2000.0f, 2500.0f, 3150.0f, 4000.0f, 5000.0f, 6300.0f
};

for (int band = 0; band < 26; band++)
{
  float frequency = frequencies[band];
  float omega = 2 * 3.14159f * frequency / waveFormat.nSamplesPerSec;
  float alpha = sin(omega) / (2 * Q);

  a0[band] = 1 + alpha;
  a1[band] = -2 * cos(omega);
  a2[band] = 1 - alpha;
  b0[band] = Q * alpha;       // == sin(omega) / 2;
  b1[band] = 0;
  b2[band] = -Q * alpha;      // == -sin(omega) / 2;
}
```

Figure 6 shows the Process method in OneSecondEchoEffect. Because the echo effect must continue after the source audio has completed, you can no longer skip processing when the XAPO_BUFFER_SILENT flag indicates no input audio. Instead, after the sound file is finished, the output audio must continue to play the tail end of the echo. The variable named source is therefore either the input audio or the value 0, depending on the existence of the XAPO_BUFFER_SILENT flag. Half of this source value is combined with half the value stored in the delay buffer, and the result is saved back into the delay buffer. At any time, you're hearing half the current audio, plus one-quarter of the audio from one second ago, plus one-eighth of the audio from two seconds ago and so forth. You can adjust the balance for different effects, including an echo that gets louder with each repetition.

Try setting the length of the delay buffer to one-tenth of a second:

```
delayLength = waveFormat.nSamplesPerSec / 10;
```

Now you get more of a reverb effect than a distinct echo. Of course, in a real APO, you'll want programmatic control over these various parameters (and others as well), which is why the real echo/reverb APO is controlled by an XAUDIO2FX_REVERB_PARAMETERS structure with 23 fields.

An APO with Parameters

Most APOs allow their behavior to be altered with runtime parameters that can be set programmatically. The SetEffectParameters method is defined for all the voice classes and references a particular APO with an index. A parametered APO is a little trickier to implement, but not much.

In the previous installment of this column, I demonstrated how to use the built-in bandpass filter implemented in the XAudio2 source and submix voices to create a 26-band graphic equalizer, in which each band affects one-third octave of the total audio spectrum. That GraphicEqualizer program effectively split the sound into 26 parts for the application of these filters, and then recombined those audio streams. This technique might have seemed somewhat inefficient.

It's possible to implement an entire graphic equalizer algorithm in a single APO, and to get the same effect as the previous program with just one source voice and one mastering voice. This is what I've done in the GraphicEqualizer2 program. The new program looks the same and sounds the same as the earlier program, but internally it's quite different.

One of the issues in passing parameters to an APO is thread synchronization. The Process method runs in the audio-processing thread, and parameters are likely being set from the UI thread. Fortunately, the CXAPOParametersBase class performs this synchronization for you.

You first need to define a structure for the parameters. For the 26-band equalizer effect, the structure contains just one field that's an array of 26 amplitude levels:

```
struct OneThirdOctaveEqualizerParameters
{
  std::array<float, 26> Amplitude;
};
```

Figure 9 **The Process Method in OneThirdOctaveEqualizerEffect**

```
void OneThirdOctaveEqualizerEffect::Process(UINT32 inpParamCount,
  const XAPO_PROCESS_BUFFER_PARAMETERS *pInpParam,
  UINT32 outParamCount,
  XAPO_PROCESS_BUFFER_PARAMETERS *pOutParam,
  BOOL isEnabled)
{
  // Get effect parameters
  OneThirdOctaveEqualizerParameters * pEqualizerParams =
    (OneThirdOctaveEqualizerParameters *) CXAPOParametersBase::BeginProcess();

  // Get buffer pointers and other information
  const float * pSrc = static_cast<float *>(pInpParam[0].pBuffer);
  float * pDst = static_cast<float *>(pOutParam[0].pBuffer);
  int frameCount = pInpParam[0].ValidFrameCount;
  int numChannels = waveFormat.nChannels;

  switch(pInpParam[0].BufferFlags)
  {
  case XAPO_BUFFER_VALID:
    for (int frame = 0; frame < frameCount; frame++)
    {
      for (int channel = 0; channel < numChannels; channel++)
      {
        int index = numChannels * frame + channel;

        // Do very little if filter is disabled
        if (!isEnabled)
        {
          pDst[index] = pSrc[index];
          continue;
        }

        // Get previous inputs
        float x = pSrc[index];
        float xp = pxp[channel];
        float xpp = pxpp[channel];

        // Initialize accumulated value
        float accum = 0;

        for (int band = 0; band < 26; band++)
        {
          int bandIndex = numChannels * band + channel;

          // Get previous outputs
          float yp = pyp[bandIndex];
          float ypp = pypp[bandIndex];

          // Calculate filter output
          float y = (b0[band] * x + b1[band] * xp + b2[band] * xpp
                                  - a1[band] * yp - a2[band] * ypp) / a0[band];

          // Accumulate amplitude-adjusted filter output
          accum += y * pEqualizerParams->Amplitude[band];

          // Save previous output values
          pypp[bandIndex] = yp;
          pyp[bandIndex] = y;
        }

        // Save previous input values
        pxpp[channel] = xp;
        pxp[channel] = x;

        // Save final value adjusted for filter gain
        pDst[index] = accum / Q;
      }
    }
    break;

  case XAPO_BUFFER_SILENT:
    break;
  }

  // Set output parameters
  pOutParam[0].ValidFrameCount = pInpParam[0].ValidFrameCount;
  pOutParam[0].BufferFlags = pInpParam[0].BufferFlags;

  CXAPOParametersBase::EndProcess();
}
```

Within the program, the members of this array are calculated from the decibel values of the sliders.

To initialize CXAPOParametersBase, you need to pass an array of three of the parameter structures to its constructor. CXAPOParametersBase uses this block of memory to perform the thread synchronization.

We've again encountered the problem of passing initialized data to a base class constructor from a derived class. The solution I chose this time was to define the derived class constructor as protected and instantiate the class from a public static method named Create, which is shown in **Figure 7**.

The digital biquad filters implemented in XAudio2 (which are emulated in this APO) involve the following formula:

$$y = (b0 \cdot x + b1 \cdot x' + b2 \cdot x'' - a1 \cdot y' - a2 \cdot y'') / a0$$

In this formula, x is the input sample, x' is the previous input sample, and x" is the sample before that. The output is y, y' is the previous output and y" is the output before that.

An equalizer effect thus needs to save two previous input values for each channel, and two previous output values for each channel and each band.

The six constants in this formula depend on the type of filter; the cutoff frequency (or in the case of a bandpass filter, the center frequency) relative to the sampling rate; and Q, the filter quality. For a one-third-octave graphic equalizer, each filter has a Q corresponding to a bandwidth of one-third octave, or 4.318. Each band has a unique set of constants that are calculated in the LockForProcess method with the code shown in **Figure 8**.

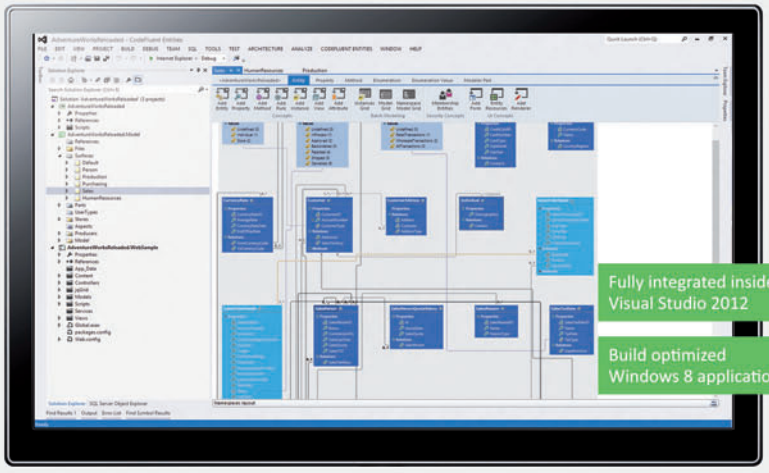
During the Process method, the APO obtains a pointer to the current parameters structure with a call to CXAPOParametersBase::BeginProcess, in this case casting the return value to a structure of type OneThirdOctaveEqualizerParameters. At the end of the Process method, a call to CXAPOParametersBase::EndProcess releases the method's hold on the parameters structure. The complete Process method is shown in **Figure 9**.

One characteristic of programming that I've always liked is that problems often have multiple solutions. Sometimes a different solution is more efficient in some way, and sometimes not. Certainly replacing 26 IXAudio2SubmixVoice instances with a single APO is a radical change. But if you think this change is reflected in vastly improved performance, you're wrong. The Windows 8 Task Manager reveals that the two GraphicEqualizer programs are approximately equivalent, suggesting that splitting an audio stream into 26 submix voices isn't so crazy after all. ■

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 experts for reviewing this article: *Duncan McKay (Microsoft) and James McNellis (Microsoft)*

Save your time!
Stop writing repetitive code and focus on what matters



Generate rock-solid foundations for your .NET applications

CodeFluent Entities

A centralized model
Don't ever repeat yourself. A single model, from Visual Studio drives all your developments from database to UI.

Continuous generation
Generate continuously throughout developments without losing code or data.

Flexibility
Support multiple databases, languages, user interfaces and architectures by simply turning features on and off from your model.

Migration & interoperability
Import existing databases and create .NET applications on top of them or migrate them to new ones.

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
SoftFluent

DAVID S. PLATT

Getting Heisenberged

My college physics lab partner and I once hung a sign over our office couch saying, "Heisenberg may have slept here, but not for long."

We were referring, of course, to Heisenberg's Uncertainty Principle, a foundation of modern physics. Most laypeople misunderstand it as simply saying you never know anything precisely. Despite its simple mathematical formulation ($\Delta P \Delta X \geq h/4\pi$), the Uncertainty Principle is remarkably subtle. Heisenberg says that certain complementary properties are entwined such that knowing more about one means you necessarily know less about the other.

For example, the more precisely you determine the position of some particle, the less precisely you can determine its momentum, and vice versa. A similar restriction connects a particle's energy and the time over which it is measured. Heisenberg and his contemporaries Einstein (relativity) and Gödel (incompleteness) postulated a limit on human knowledge regardless of the precision of instruments.

> Geeks bump into Heisenberg when they start trying to figure out what users want in their programs.

Geeks bump into Heisenberg when they start trying to figure out what users want in their programs. You can't do it via ESP. You can only measure it in different ways. As in physics, each type of measurement shows certain parts of the total picture, but hides others as it does so.

For example, data telemetry is considered the ultimate in user tracking. You can instrument your entire program, and because it's automated you can do it for many users at low cost. You can record what users actually do instead of what they can remember doing or are willing to admit to doing. But telemetry can't show you other important information, and if you forget that it exists you'll get hurt.

Microsoft got hammered in press reviews and user interviews for removing the Start menu from Windows 8. Microsoft made this decision because telemetry reported that users rarely used the Start menu—they placed icons on the taskbar for frequently used programs, and typed into the search box for infrequently used ones. Nevertheless, after removing the menu, Microsoft found that its mere presence had reassured users, who knew where to turn if they got stuck.

It's like the way terminally ill cancer patients often request a lethal prescription but rarely use it. They crave not so much the relief of actual pain, but relief from the fear of forthcoming unbearable pain. (Hey, here's a good riddle: "How is the Windows 7 Start menu like a lethal dose of barbiturates?" Cut me in for a share if you ever win anything with it.)

Telemetry can't tell you that sort of thing.

For another example, UX testing books such as Steve Krug's "Rocket Surgery Made Easy" (New Riders, 2009) tell you to start laboratory UX testing early and often. Here you can ask the users *why* they did such and such a thing. "Did you notice this button over here?" "What did you think it would do?" "Why didn't you click it like *you should have, you ^*&%^$ idiot*?!" and so on. But you can't exercise very much of your UX that way, because it takes too long and costs too much. This type of testing shows you depth, but not breadth.

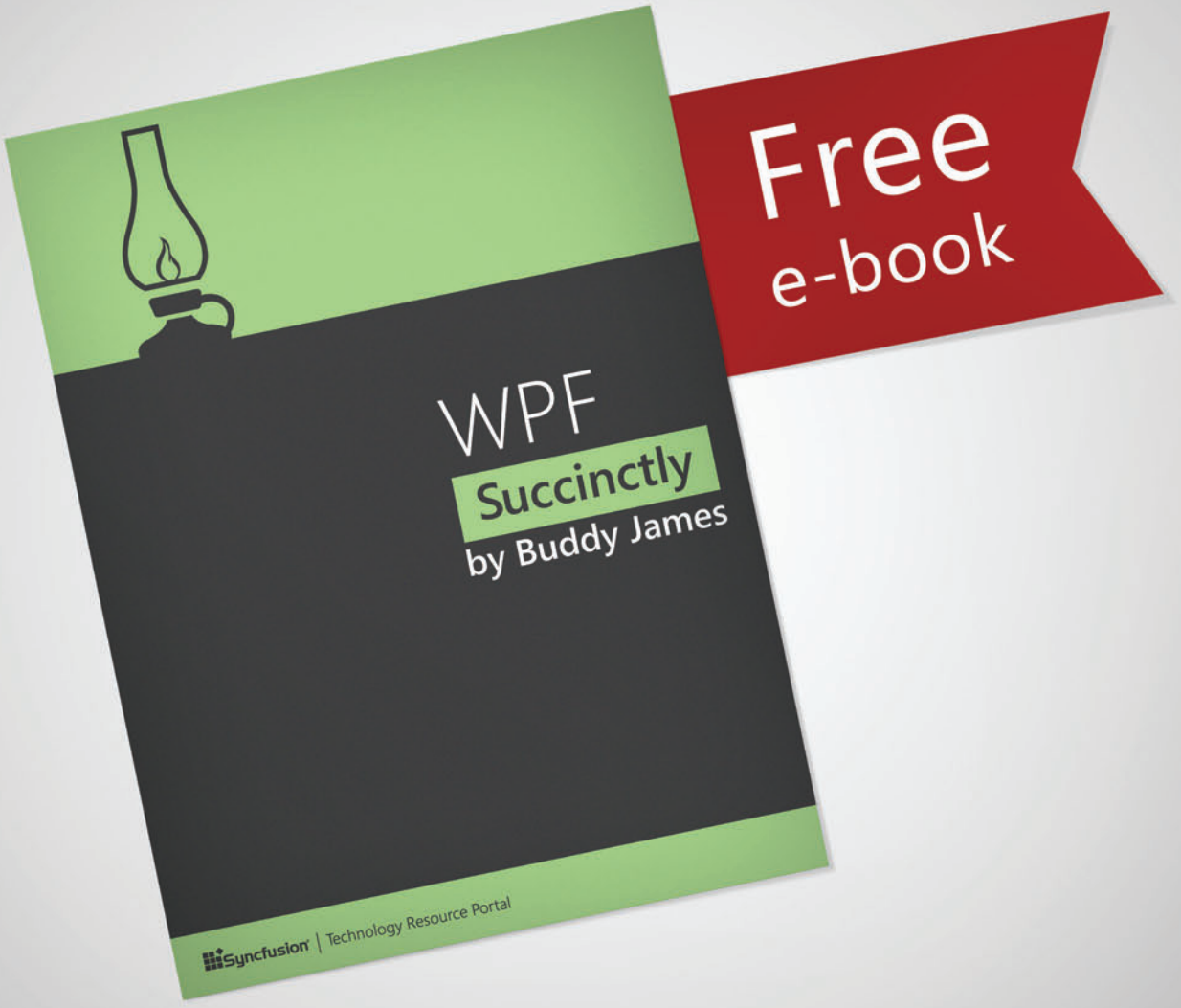
Consumer focus groups take a lot of flak, including some from me. But they offer the very first glance from a bunch of non-geeks, which is hard to get any other way. First experiences are especially important in Web commerce situations, where if you can't catch the user's attention in a second or two, you don't get another chance.

On the other hand, focus groups can't say much about how feelings develop over time. They famously loved Clippy when they first saw him ("Oh, what cute little eyebrows"). But users quickly tired of him in actual use. ("I see you're writing a crude forgery. Can I help? Is this a business forgery or a personal forgery?") As etiquette columnist Judith Martin, aka Miss Manners, once wrote: "The first time they see it, [your users] will think it's cute. The 11th time they see it, they will think about killing you."

You need to understand what each source of user information reveals and what it concomitantly obscures. You need to use each source of information for what it's good at, and fill in with the others for what it's not. Heisenberg aside, I'm certain of that. ■

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
100s of controls for
HTML5 & JavaScript
WPF & Silverlight
Windows Forms
Windows Phone
Windows 8
TOTALS
Media
Appliances
Games
$1M
$800K
$600K
$400K
$200K
$0
Sep 3, 2012
Sep 4, 2012
Sep 5, 2012
Sep 6 2012
Sep 7 2012
Save the Date
Final QA
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 ther respective holders.



Download *WPF Succinctly* today to learn how to:

- Use MVVM and XAML to separate your app's UI, business logic, and data.
- Manage data binding, data flow, and MultiBindings.
- Leverage commands to handle user input.

DOWNLOAD NOW

syncfusion.com/wpfebook

Syncfusion®