THE MICROSOFT JOURNAL FOR DEVELOPERS

JULY 2013 VOL 28 NO 7

msdn magazine

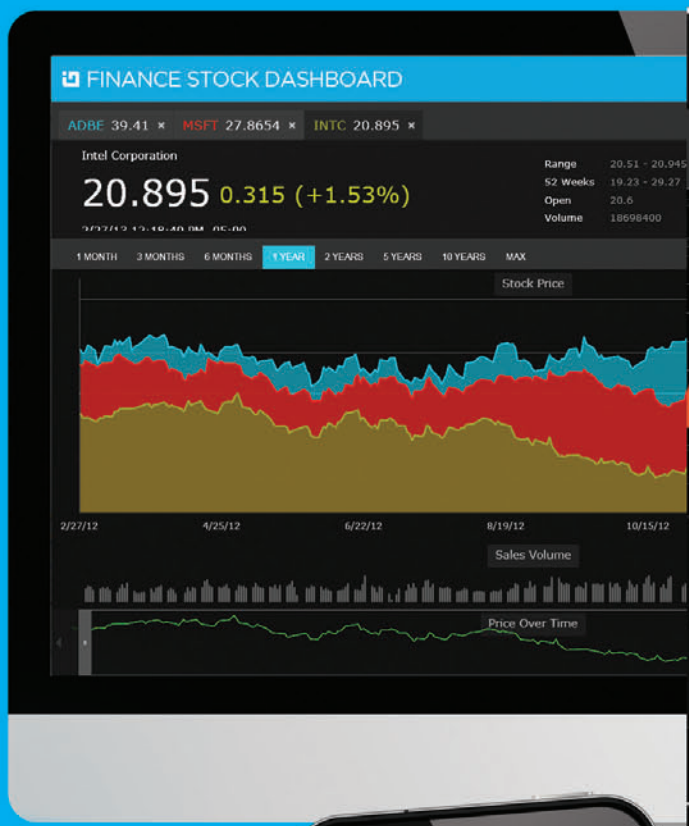
Inside the "Roslyn" Project Compiler.....34

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

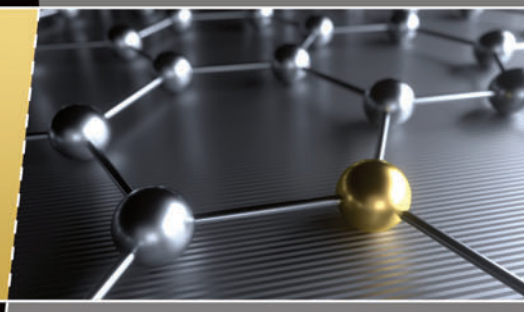
msdn magazine

JULY 2013 VOLUME 28 NUMBER 7

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

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

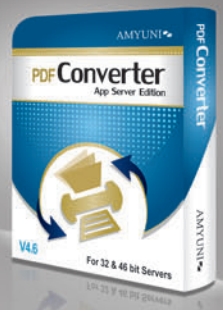
Complete Suite of Accurate PDF Components

- 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

MICHAEL DESMOND

Dev/Test in the Cloud

In case you haven't noticed, Microsoft has been extremely busy of late improving and promoting its Windows Azure cloud platform as a compelling tool for development. Most recently, the Windows Azure group has been preaching the benefits of Windows Azure in dev and test operations, citing the flexibility and rapid scalability provided by Windows Azure virtual machines (VMs).

Craig Kitterman, product manager for the Windows Azure Team at Microsoft, gave a keynote address at the Visual Studio Live! Chicago conference in May. When I asked Kitterman about aspects of Windows Azure that developers find most compelling, he singled out dev and test.

Kitterman stressed that developers often look at the cloud as "an all-or-nothing proposition," when it in fact allows them to start small and go big with things such as dev/test environments.

"We're seeing the most buzz around core scenarios that developers can get started with quickly. The primary one here is the ability to use Windows Azure virtual machines for doing dev and test in the cloud," Kitterman told me. "Many developers think they have to be deploying into the cloud for production to get any benefit, but with on-demand, scriptable VMs, anyone can stand up and tear down a complete dev/test environment in minutes. MSDN subscribers can actually do it for free today by activating their Azure benefit in just a few minutes."

Kitterman stressed that developers often look at the cloud as "an all-or-nothing proposition," when it in fact allows them to start small and go big with things such as dev/test environments.

"Hybrid is a core design point for Azure," Kitterman said, "so we've built it from the ground up to make the on-ramp easy, allowing customers to leverage existing IT investments while taking advantage of some Azure services."

This benefit is certainly visible in the dev/test cycle, where Windows Azure can help dev organizations break time- and budget-sapping bottlenecks. Tasks that once demanded steep investments in physical infrastructure can now be serviced from the cloud.

"Generally speaking, you want your test environment to mirror exactly your production environment. This makes your testing environment realistic. But this also is very expensive because you're essentially doubling the cost of all your hardware," says Bruno Terkaly, coauthor of the *MSDN Magazine* Windows Azure Insider column and a Microsoft technical evangelist. "Being able to quickly deploy to a staging environment, and with the click of a mouse, moving it to production is a seamless process. Companies don't need to invest in these large test environments."

Terkaly adds that Microsoft is investing heavily in automation, enabling the use of Windows PowerShell scripts to automate application testing and deployment, for instance. He also notes that many developers he talks to aren't aware that the cloud, by way of Visual Studio Team Foundation Service, can be used to leverage application lifecycle management (ALM) features such as source control, bug tracking, build automation, test-case management and test labs.

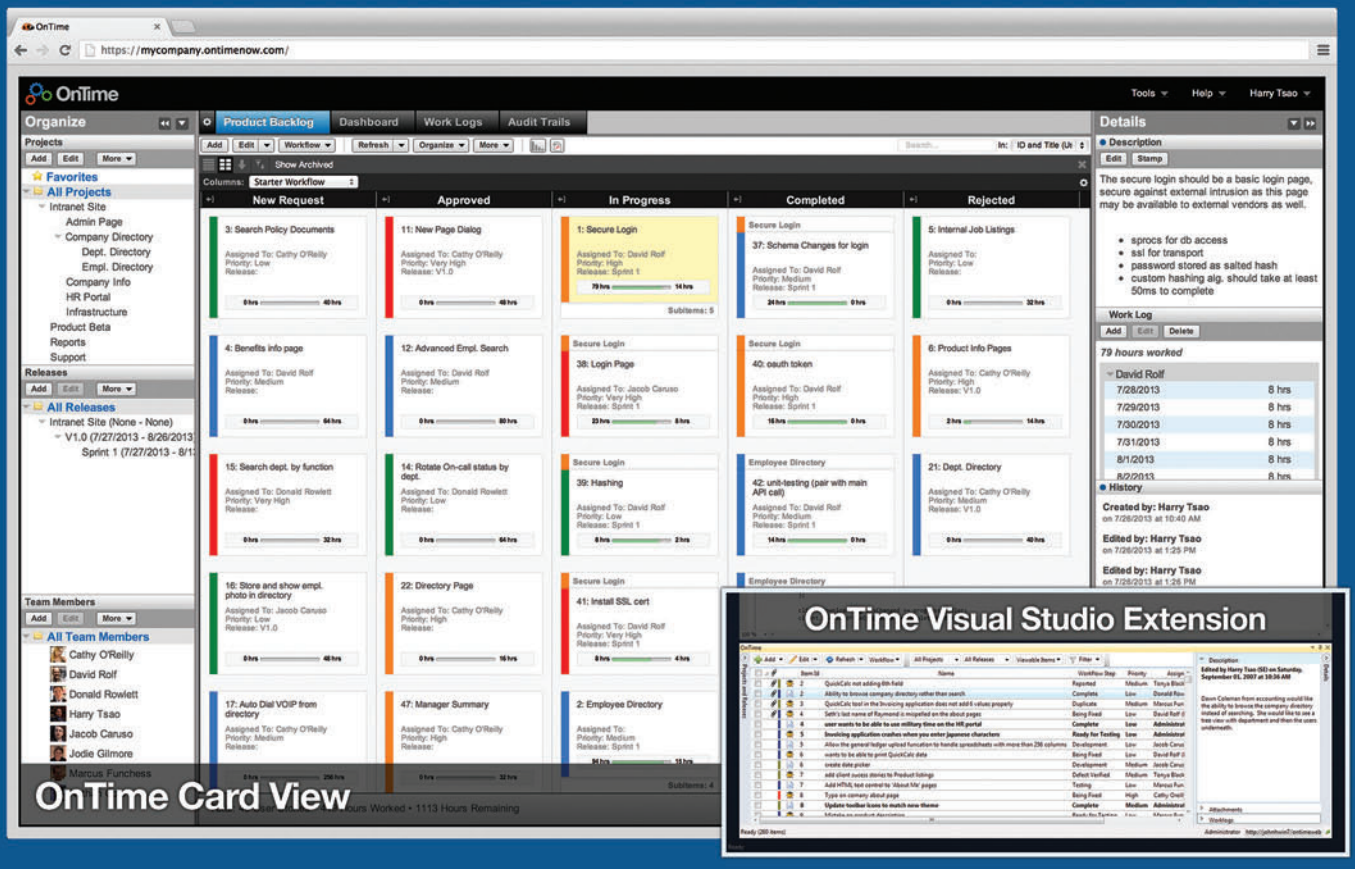
Windows Azure has been a bit slow to earn uptake in the development space, but that trend by all accounts is changing, and offerings like Team Foundation Service are making cloud-based development hard to ignore. Still, Kitterman said, Microsoft has "a ways to go."

"Naturally, it's going to take some time as enterprise application lifecycles are long, and new applications are where developers will see the most benefit," Kitterman said. "Driving awareness of scenarios like dev/test and VM migration that developers can take advantage of immediately will be my focus in the short term."

Is your dev organization looking into Windows Azure? What would you like to see Microsoft do to make Windows Azure more compelling for your needs?

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

the #1 selling scrum software



OnTime Scrum agile project management software

Want to ship your software 24% faster? Think about it: that's **3 free months every year**, and it's the average time saved by users of OnTime Scrum.

How are they doing it? Because OnTime Scrum is fast, extremely customizable, and easy to use. Features like **Card View** make managing your user stories highly visible and effortless. And with our **Visual Studio integration**, your dev team can manage their OnTime items without leaving their development environment.

So is it any wonder that OnTime Scrum is the **#1 selling scrum software**?

just **$7** per user per month

Get 10% off your OnTime subscription with this link:

OnTimeNow.com/MSDN

Plus: learn about our solutions for bug tracking, help desk & customer management, and project wikis

axosoft

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

DINO ESPOSITO

Creating Mobile-Optimized Views in ASP.NET MVC 4

If you scratch the surface of common-sense considerations about programming mobile sites, you find an inherent contradiction. On one hand, you hear people screaming about being (or willing to be) mobile-first in their approach to programming applications and sites. On the other hand, you often hear the same people praise CSS media queries and liquid layouts. The contradiction I see is that the common use being made of CSS media queries and liquid layouts doesn't put the mobile perspective before everything—it's not a mobile-first technique. In this article I'll explain how to use server-side logic to offer up the best display for a given device, incorporating a new feature of ASP.NET MVC 4 called display modes.

The problem isn't CSS media queries as a technology. The problem isn't even responsive Web design (RWD) as the supporting methodology of CSS media queries—if not the technology's inspiring philosophy. So what makes the use of CSS media queries and liquid layouts a "mobile-last" approach? A clue can be found right in the slogan used to push this approach: A single codebase can serve multiple views. In this perspective, CSS—a client-side technology—is used to switch between views, and JavaScript is used to adapt views further when CSS isn't enough.

As I see things, in this approach there's the underlying proposition of serving all devices the same content, just adjusting the page layout to fit the screen size. In doing so, you might not be able to offer the best possible experience to users. I believe you reasonably should aim at having a single codebase—a common ground of Web APIs—but should definitely focus on specific use cases for each class of devices you intend to support. The term "mobile" makes little sense today, as it's being replaced by classes of devices such as smartphones, tablets, laptops and smart TVs—not to mention wearable devices such as eyeglass displays and smart wristwatches.

About a year ago, I presented in this column a server-side approach to take with ASP.NET MVC site development: to build ad hoc views for each class of supported devices ("Mobile Site Development: Markup," msdn.microsoft.com/magazine/jj133814). I did that in the context of ASP.NET MVC 3. Nicely enough, ASP.NET MVC 4 comes with the aforementioned display modes, which easily can be employed to implement server-side logic that intelligently serves the best view and content for a given device. To be really effective, this approach requires that you know as much as possible about the capabilities of the requesting device. However, besides basic information about the screen size and current orientation, there's not much else that can be detected on the client. You then need to resort to a server repository of device information.

Figure 1 **The Standard List of Supported Display Modes**

```
<h2>
  Display Modes currently active
  (@DisplayModeProvider.Instance.Modes.Count mode(s))
</h2>
<ul>
@{
  foreach(var d in DisplayModeProvider.Instance.Modes)
  {
    <li>@(String.IsNullOrEmpty(d.DisplayModeId)
      ?"default" :d.DisplayModeId)</li>
  }
}
</ul>
```

Introducing Display Modes in ASP.NET MVC 4

Before I start delving deep into display modes, let me state up front that this article (as well as display mode technology itself) is mainly concerned with building a new, unique site that dynamically binds the same URL to different views. If you already have a Web site and want to provide a companion site optimized for some (mobile) devices, well, that's a different story. You can still use this column as a guide for building the companion site, but unifying URLs with an existing parent site requires different tools.

In ASP.NET MVC 4, display modes are a system feature that extends the classic behavior of view engines with the ability to pick the view file that's most appropriate for the requesting device. In the aforementioned article for ASP.NET MVC 3, I used a custom view engine for this purpose. In that solution I also was limited to

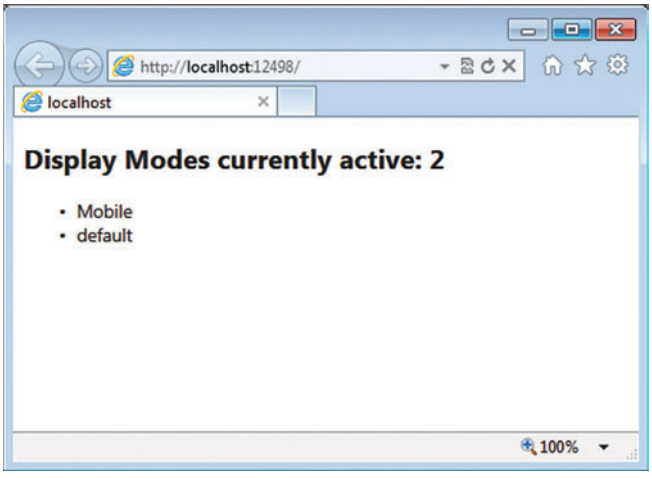
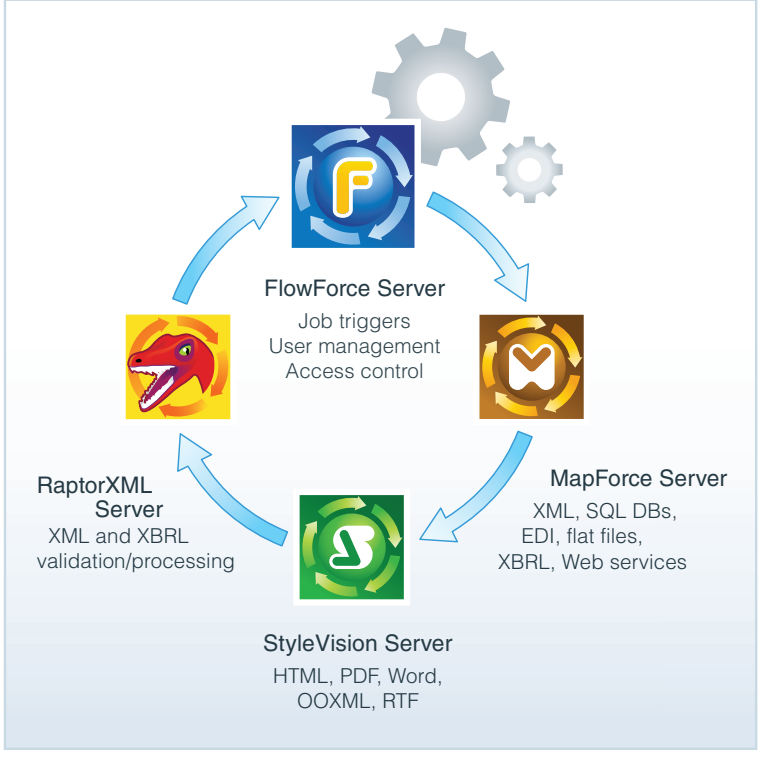


Figure 2 **Default List of Display Modes**

Manage Information Workflows with Altova® FlowForce® Server

ALTOVA® flowforce® SERVER 2013

Introducing FlowForce Server, the powerful new server software for managing today's multi-step, enterprise-level data validation, aggregation, processing, and reporting tasks. This flexible workflow orchestration tool provides easy-to-manage automation of essential business processes on high-performance servers.



FlowForce Server is at the center of Altova's new line of cross-platform server products 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
- **RaptorXML® Server** for hyper-fast validation/processing of XML, XBRL, XSLT, and XQuery

Learn more and download a free trial at www.altova.com/server

Razor views. With display modes, your controller methods will still invoke, say, a view named Index, and the ASP.NET MVC runtime will instead pick up a view file named index.mobile.cshtml if the requesting device is known to be a mobile device.

This is great news, as it means you can still have a single codebase for your site. You just need to add extra CSHTML view files for each class of device you intend to support. To start playing with display modes, let's have a look at the code sample in **Figure 1**.

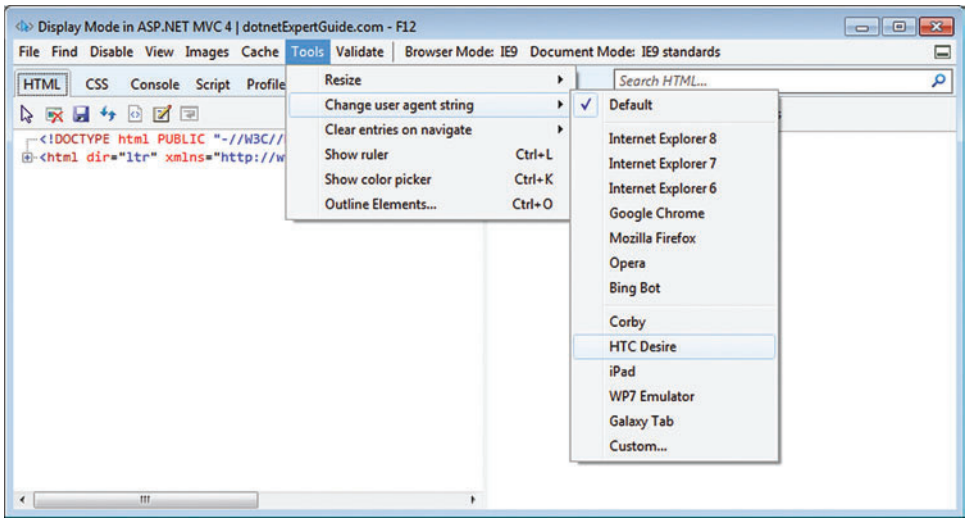


Figure 3 **Forcing a Mobile User Agent into Internet Explorer for Testing Purposes**

The page in the code in **Figure 1** shows the standard list of supported display modes. **Figure 2** shows the output generated by the page.

ASP.NET MVC 4 display modes follow a few conventions. In particular, each display mode is associated with a keyword. The keyword is used to compose the name of the corresponding view file. The default display mode is bound to an empty string. As a result, the following view files are correctly handled by any ASP.NET MVC 4 application without any further intervention from you: index.cshtml and index.mobile.cshtml.

> Each display mode is associated with a keyword.

To see a demo, copy the index.cshtml file to a new file named index.mobile.cshtml and add it to the project. To distinguish between the two files, add the following to the mobile file:

```
<div style="border-bottom: solid 1px #000">Mobile view</div>
```

If you run the application and test it using Internet Explorer or another desktop browser, nothing changes. Try hitting F12 to bring up Internet Explorer Developer Tools and set a mobile user agent (UA) by selecting Tools | Change user agent string, as shown in **Figure 3**.

I've already configured a few mobile and tablet UAs. For example, you can use the following, which identifies the requesting browser as an HTC Desire Android smartphone:

```
Mozilla/5.0 (Linux; U; Android 2.1; xx-xx; HTC Desire Build/ERE27)
AppleWebKit/525.10+ (KHTML, like Gecko) Version/3.0.4 Mobile Safari/523.12.2
```

Figure 4 shows what you get from the ASP.NET MVC 4 site. The page being served from the same pair of controller and action methods is index.mobile.cshtml. More important, all of this took place without any change in the programming style and without learning any new skills.

Going Beyond the Basics

What's been discussed so far is the bare minimum of what you can—and should—do in your mobile site development. Two key points need to be addressed to turn display modes into a solution for a realistic site. One is exploring ways to add multiple display modes. The other is exploring ways to inject some ad hoc logic to detect devices more reliably.

The built-in logic used by ASP.NET MVC to detect mobile devices isn't all that reliable. It probably works with most smartphones, but it fails with older cell phones. For example, consider the following UA:

```
SAMSUNG-GT-S3370/S3370DDJD4 SHP/VPP/R5 Dolfin/1.5 Qtv/5.3
SMM-MMS/1.2.0 profile/MIDP-2.1 configuration/CLDC-1.1 OPN-N
```

This UA refers to a legacy phone (fairly popular a couple years ago) running a proprietary OS and a WebKit-based custom browser. The phone doesn't support Wi-Fi connectivity, but it does have surprisingly good HTML-rendering capabilities. The screen is tinier than most smartphones, but it's touch-enabled. With the basic display mode support from ASP.NET MVC, this phone isn't recognized as a mobile device and is served the full versions of pages. This has two drawbacks. First, users can hardly view content because it will be stretched and wrapped around the screen. Second, a lot of content will be downloaded, and because the phone doesn't support Wi-Fi connectivity, all of this likely happens on a 3G connection—certainly slow and possibly expensive for the user.

When I raise this point, some people reply that their sites just don't support these types of legacy phones. That's perfectly sound,

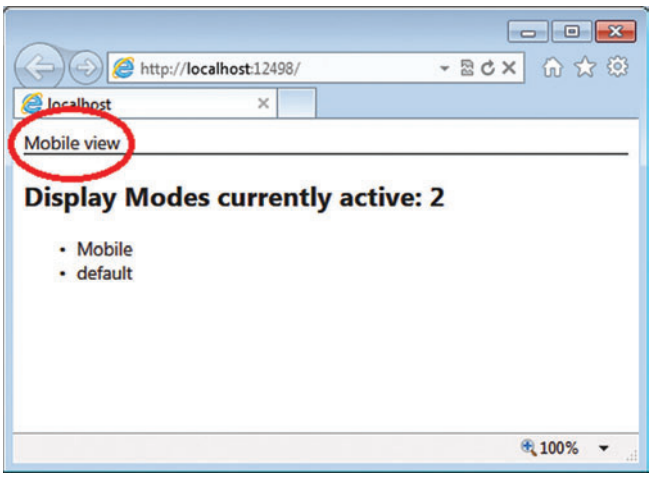


Figure 4 **Switching to a Mobile View**

Figure 5 **The DisplayConfig Class**

```
public class DisplayConfig
{
  public static void RegisterDisplayModes(IList<IDisplayMode> displayModes)
  {
    var modeDesktop = new DefaultDisplayMode("")
    {
      ContextCondition = (c => c.Request.IsDesktop())
    };
    var modeSmartphone = new DefaultDisplayMode("smart")
    {
      ContextCondition = (c => c.Request.IsSmartphone())
    };
    var modeTablet = new DefaultDisplayMode("tablet")
    {
      ContextCondition = (c => c.Request.IsTablet())
    };

    displayModes.Clear();
    displayModes.Add(modeSmartphone);
    displayModes.Add(modeTablet);
    displayModes.Add(modeDesktop);
  }
}
```

but wouldn't it be nice to send a polite message to the user in that case instead of letting things go uncontrolled? To be able to send a message such as, "Sorry, the site can't be viewed on your device," you still need to recognize the device properly and understand that it's different from, say, an iPhone. Furthermore, whether you can safely ignore legacy devices is a business—not implementation—decision. Not serving older generations of devices could affect your business beyond imagination. So, let's see how to add multiple display modes to a site to properly serve multiple classes of devices.

Classes of Devices

A modern Web site offers the best possible experience regardless of the device type. "Best possible experience" means providing ad hoc use cases, selected data and specific functions. The resulting markup has to be device-specific. If you adjust things on the client, instead—as happens when you rely on CSS media queries—you actually have a unified vision of the pages and then just adapt them to tinier screens. This mostly means hiding some blocks, flowing some others vertically and perhaps requesting a lighter set of visuals. The unified vision of the page is often the desktop page. Personally, I wouldn't call this a mobile-first approach.

By saying "device type," I don't mean distinguishing an iPhone device from a Windows Phone device. Rather, I aim to use logic that can serve different markup to smartphones, tablets and laptops. So in ASP.NET MVC 4, I'd have at least three display modes: smartphone, tablet and default (for desktop browsers). I'll add a new DisplayConfig class to be invoked in App_Start (see **Figure 5**).

Figure 6 **Extension Methods to Keep Code Clear**

```
public static class HttpRequestBaseExtensions
{
  public static Boolean IsDesktop(this HttpRequestBase request)
  {
    return true;
  }
  public static Boolean IsSmartphone(this HttpRequestBase request)
  {
    return IsSmartPhoneInternal(request.UserAgent);
  }
  public static Boolean IsTablet(this HttpRequestBase request)
  {
    return IsTabletInternal(request.UserAgent);
  }

  // More code here.
}
```

The class first empties the provided collection of display modes. In this way, it gets rid of default modes. Next, the code fills up the provided system collection with a newly created list of display modes. A new display mode is an instance of the DefaultDisplayMode class. The name of the mode is set through the constructor. The logic that determines whether a given UA is matched is set through the property ContextCondition.

The ContextCondition property is a delegate that accepts an HttpContextBase object and returns a Boolean. The body of the delegate is all about snooping into the HTTP context of the current request in order to determine whether the given display mode is appropriate. In **Figure 5**, I use a few extension methods to keep code highly readable. **Figure 6** lists those extensions methods.

All the code discussed so far is pure infrastructure. You end up with one method to write for each display mode. Each method takes a UA and must return a Boolean answer. Here's a very basic routine to check for tablets:

```
private static Boolean IsTabletInternal(String userAgent)
{
  var ua = userAgent.ToLower();
  return ua.Contains("ipad") || ua.Contains("gt-");
}
```

This routine is only guaranteed to successfully detect iPad and Galaxy Tab devices, but you can see the point about how these context condition routines should be written. You might want to add more code to check for smartphones, at the very least. To detect tablets and smartphones, you can take advantage of any open source or commercial Device Description Repository (DDR) framework. I'll discuss that in more detail in my next column.

Serious Business

A server-side approach to mobile sites may not always be necessary, but it's a serious matter whenever there's some business behind the site. I wouldn't recommend a server-side approach for, say, a conference site or any sort of short-lived site. However, a business site aimed at the largest possible audience needs to emphasize optimization for devices beyond just plain rendering for mobile.

On the client, you're limited to browser window size and orientation, and you can't check the OS or touch capabilities—or check for more advanced things such as whether the device supports wireless, streaming, inline images, SMS and more. Display modes make it particularly easy to implement a multi-view approach in ASP.NET MVC 4.

In my next column I'll cap off my argument by showing how to integrate the DDR used by Facebook—Wireless Universal Resource File (WURFL)—with ASP.NET MVC 4. ■

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

Thanks to the following technical expert for reviewing this article: *Mani Subramanian (Microsoft)*

JULIE LERMAN

Behavior-Driven Design with SpecFlow

By now you're familiar with my penchant to invite developers to give talks on topics I've been curious about at the user group I lead in Vermont. This has resulted in columns on topics such as Knockout.js and Breeze.js. There are still more topics, such as Command Query Responsibility Segregation (CQRS), that I've been chewing on for a while. But recently Dennis Doire, an architect and tester, spoke on SpecFlow and Selenium, two tools for testers doing behavior-driven development (BDD). Once again, my eyes got wide and my mind started looking for excuses to play with the tools. Really, though, it was BDD that got my attention. Even though I'm a data-driven person, my days of designing applications from the database up are far behind me and I've become interested in focusing on the domain.

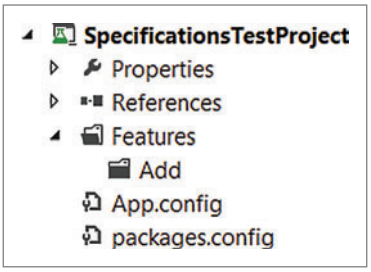


Figure 1 **Test Project with Features and Add Sub-Folders**

BDD is a twist on test-driven development (TDD) that focuses on user stories and building up logic and tests around those stories. Rather than satisfying a single rule, you satisfy sets of activities. It's very holistic, which I love, so this perspective interests me a great deal. The idea is that while a typical unit test might ensure that a single event on a customer object works properly, BDD focuses on the broader story of the behavior that I, the user, expect when I'm using the system you're building for me. BDD is often used to define acceptance criteria during discussions with clients. For example, when I sit in front of the computer and fill out a New Customer form and then hit the Save button, the system should store the customer information and then show me a message that the customer has been successfully stored.

Or perhaps when I activate the Customer Management portion of the software, it should automatically open up the most recent Customer I worked on in my last session.

You can see from these user stories that BDD might be a UI-oriented technique for designing automated tests, but many of the scenarios are written before a UI has been designed. And thanks to tools such as Selenium (docs.seleniumhq.org) and WatiN (watin.org), it's possible to automate tests in the browser. But BDD isn't just about describing user interaction. To get a view of the bigger picture of BDD, check out the panel discussion on InfoQ among some of the authorities on BDD, TDD and Specification by Example at bit.ly/10jp6ve.

Code download available at archive.msdn.microsoft.com/mag201307DataPoints.

I want to step away from worrying about button clicks and such and redefine the user stories a little. I can remove the UI-dependent elements of the story and focus on the part of the process that isn't dependent on the screen. And, of course, the stories I'm interested in are the ones related to data access.

Building up the logic to test that a particular behavior is being satisfied can be tedious. One of the tools Doire demonstrated in his presentation was SpecFlow (specflow.org). This tool integrates with Visual Studio and enables you to define user stories—called scenarios—using its simple rules. It then automates some of the creation and execution of the methods (some with tests and some without). The goal is to validate that the rules of the story are being satisfied.

I'm going to walk you through creating a few behaviors to whet your appetite, and then if you're looking for more, you'll find some resources at the end of the article.

First you need to install SpecFlow into Visual Studio, which you can do from the Visual Studio Extensions and Updates Manager. Because the point of BDD is to begin your project development by describing behaviors, the first project in your solution is a test project in which you'll describe these behaviors. The rest of the solution will flow from that point.

Create a new project using the Unit Test Project template. Your project will need a reference to TechTalk.SpecFlow.dll, which you can install using NuGet. Then create a folder called Features inside this project.

My first feature will be based on a user story about adding a new customer, so inside the Features folder, I create another called Add (see **Figure 1**). Here's where I'll define my scenario and ask SpecFlow to help me out.

SpecFlow follows a specific pattern that relies on keywords that help describe the feature whose behavior you're defining. The keywords come from a language called Gherkin (yes, as in pickle), and this all originates from a tool called Cucumber (cukes.info). Some of these keywords are Given, And, When, and Then, and you can use them to build a scenario. For example, here's a simple scenario, which is encapsulated in a feature—Adding a New Customer:

Given a user has entered information about a customer
When she completes entering more information
Then that customer should be stored in the system

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

Experience for yourself:

Excel®-like grid. Native Microsoft® Excel® Compatibility - same look & feel for your users

Flexible & familiar spreadsheet architecture, advanced charting & powerful formula library

Create financial modeling & risk analysis, budgeting, insurance, scientific applications & more

NEW VERSION 7

Spread Studio for .NET

Download your free trial @

www.componentone.com

ComponentOne®
a division of GrapeCity®

You could be more elaborate, for example:

Given a user has entered information about a customer
And she has provided a first name and a last name as required
When she completes entering more information
Then that customer should be stored in the system

That last statement is where I'll be doing some data persistence. SpecFlow doesn't care about how any of this happens. The goal is to write scenarios to prove the outcome is and remains successful. The scenario will drive the set of tests and the tests will help you flesh out your domain logic:

Given that you have used the proper keywords
When you trigger SpecFlow
Then a set of steps will be generated for you to populate with code
And a class file will be generated that will automate the execution of these steps on your behalf

Let's see how this works.

Right-click the Add folder to add a new item. If you've installed SpecFlow, you can find three SpecFlow-related items by searching on specflow. Select the SpecFlow Feature File item and give it a name. I've called mine AddCustomer.feature.

Figure 2 **The Steps.cs File**

```
[Binding]
public class Steps
{
  [Given(@"a user has entered information about a customer")]
  public void GivenAUserHasEnteredInformationAboutACustomer()
  {
    ScenarioContext.Current.Pending();
  }

  [Given(@"she has provided a first name and a last name as required")]
  public void GivenSheHasProvidedAFirstNameAndALastNameAsRequired
 ()
  {
    ScenarioContext.Current.Pending();
  }

    [When(@"she completes entering more information")]
  public void WhenSheCompletesEnteringMoreInformation()
  {
    ScenarioContext.Current.Pending();
  }

  [Then(@"that customer should be stored in the system")]
  public void ThenThatCustomerShouldBeStoredInTheSystem()
  {
    ScenarioContext.Current.Pending();
  }

  [Given(@"she has not provided both the firstname and lastname")]
  public void GivenSheHasNotProvidedBothTheFirstnameAndLastname()
  {
    ScenarioContext.Current.Pending();
  }

  [Then(@"that user will get a message")]
  public void ThenThatUserWillGetAMessage()
  {
    ScenarioContext.Current.Pending();
  }

  [Then(@"the customer will not be stored into the system")]
  public void ThenTheCustomerWillNotBeStoredIntoTheSystem()
  {
    ScenarioContext.Current.Pending();
  }

}
```

A feature file starts out with a sample—the ubiquitous math feature. Notice that the Feature is described at the top, and on the bottom a Scenario (which represents a key example of the feature) is described using Given, And, When and Then. The SpecFlow add-in ensures that the text is color-coded so you can easily discern the step terms from your own statements.

I'll replace the canned feature and steps with my own:

```
Feature: Add Customer
Allow users to create and store new customers
As long as the new customers have a first and last name

Scenario: HappyPath
Given a user has entered information about a customer
And she has provided a first name and a last name as required
When she completes entering more information
Then that customer should be stored in the system
```

(Thanks to David Starr for the Scenario name! I stole it from his Pluralsight video.)

What if the required data isn't provided? I'll create another scenario in this feature to handle that possibility:

```
Scenario: Missing Required Data
Given a user has entered information about a customer
And she has not provided the first name and last name
When she completes entering more information
Then that user will be notified about the missing data
And the customer will not be stored into the system
```

That will do for now.

From User Story to Some Code

So far you've seen the Feature item and the color-coding that SpecFlow provides. Note that there's a codebehind file attached to the feature file, which has some empty tests that were created based on the features. Each of those tests will execute the steps in your scenario, but you do need to create those steps. There are a few ways to do that. You could run the tests, and SpecFlow will return the code listing for the Steps class in the test output for you to copy and paste. Alternatively, you could use a tool in the feature file's context menu. I'll describe the second approach:

1. Right-click in the text editor window of the feature file. On the context menu, you'll see a section dedicated to SpecFlow tasks.
2. Click Generate Step Definitions. A window will pop up verifying the steps to create.
3. Click the Copy methods to clipboard button and use the defaults.
4. In the AddCustomer folder in your project, create a new class file named Steps.cs.
5. Open the file and, inside the class definition, paste the clipboard contents.
6. Add a namespace reference to the top of the file, using TechTalk.SpecFlow.
7. Add a Binding annotation to the class.

The new class is listed in **Figure 2**.

If you look at the two scenarios I've created, you'll notice that while there's some overlap in what's defined (such as "a user has entered information about a customer"), the generated methods do not create duplicate steps. It's also notable that SpecFlow will leverage the constants in the method attributes. The actual method names are irrelevant.

At this point, you could let SpecFlow run its tests that will call these methods. While SpecFlow supports a number of unit-testing

telerik

deliver more than expected

Telerik DevCraft Q2

RELEASE: JUNE 2013

- Telerik's full stack of .NET controls and tools for the professional developer
- 350+ UI controls and reporting for all Microsoft platforms
- Productivity tools for faster coding, debugging and profiling

See what's new in Q2 2013 Release & download your 30 day free trial at

www.telerik.com

frameworks, I'm using MSTest, so if you're looking at this solution in Visual Studio, you'll see that Feature's codebehind file defines a TestMethod for each scenario. Each TestMethod executes the correct combination of step methods with a TestMethod that's run for the HappyPath scenario.

If I were to run this now, by right-clicking the Feature file and choosing "Run SpecFlow Scenarios," the test would be inconclusive, with the message: "One or more step definitions are not implemented yet." That's because each of the methods in the Steps file are all still calling Scenario.Current.Pending.

> I'm pretty sure that if it weren't for all of the tools that have been built to support BDD, my path into it wouldn't have been so easy.

So, it's time to flesh out the methods. My scenarios tell me I'll need a Customer type with some required data. Thanks to other documentation, I know that the first and last name are currently required, so I'll need those two properties in the Customer type. I also need a mechanism for storing that customer, as well as a place to store it. My tests don't care how or where it's stored, just that it is, so I'll use a repository that will be responsible for getting and storing data.

I'll start by adding _customer and _repository variables to my Steps class:

```
private Customer _customer;
private Repository _repository;
```

And then stub out a Customer class:

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

That's enough to let me add code to my step methods. **Figure 3** shows the logic added to the steps related to HappyPath. I create a new customer in one, then I provide the required first and last names in the next. There's not really anything I need to do to elaborate on the WhenSheCompletesEnteringMoreInformation step.

Figure 3 **Some of the SpecFlow Step Methods**

```
[Given(@"a user has entered information about a customer")]
public void GivenAUserHasEnteredInformationAboutACustomer()
{
  _newCustomer = new Customer();
}

[Given(@"she has provided a first name and a last name as required")]
public void GivenSheHasProvidedTheRequiredData()
{
  _newCustomer.FirstName = "Julie";
  _newCustomer.LastName = "Lerman";
}

[When(@"she completes entering more information")]
public void WhenSheCompletesEnteringMoreInformation()
{
}
```

The last step is the most interesting. This is where I not only store the customer, but prove that it has indeed been stored. I'll need an Add method in my repository to store the customer, a Save to push it into the database and then a way to see if that customer can, in fact, be found by the repository. So I'll add an Add method, a Save and a FindById method to my repository, like so:

```
public class CustomerRepository
{
  public void Add(Customer customer)
    { throw new NotImplementedException();  }

  public int Save()
    { throw new NotImplementedException();  }

  public Customer FindById(int id)
    { throw new NotImplementedException();  }
}
```

Now I can add logic to the final step that will be called by my HappyPath scenario. I'll add the customer to the repository and test to see if it can be found in the repository. This is where I finally use an assertion to determine if my scenario is succeeding. If a customer is found (that is, IsNotNull), the test passes. This is a very common pattern for testing that data has been stored. However, from my experience with Entity Framework, I see a problem that won't get caught by the test. I'll start with the following code so I can show you the problem in a way that may be more memorable than just showing you the correct way to begin with (I'm mean that way):

```
[Then(@"that customer should be stored in the system")]
public void ThenThatCustomerShouldBeStoredInTheSystem()
{
  _repository = new CustomerRepository();
  _repository.Add(_newCustomer);
  _repository.Save();
  Assert.IsNotNull(_repository.FindById(_newCustomer.Id));
}
```

When I run my HappyPath test again, the test fails. You can see in **Figure 4** that the test output shows how my SpecFlow scenario is working out so far. But pay attention to the reason the test failed: It's not because the FindById didn't find the customer, it's because my repository methods are not yet implemented.

So, my next step is to provide logic to my repository. Eventually I'll use this repository to interact with the database and, as I happen to be a fan of Entity Framework, I'll use an Entity Framework DbContext in my repository. I'll start by creating a DbContext class that exposes a Customers DbSet:

```
public class CustomerContext:DbContext
{
  public DbSet<Customer> Customers { get; set; }
}
```

Then I can refactor my CustomerRepository to use the CustomerContext for persistence. For this demo, I'll work directly against the context rather than worrying about abstractions. Here's the updated CustomerRepository:

```
public  class CustomerRepository
{
  private CustomerContext _context = new CustomerContext();

  public void Add(Customer customer
  {    _context.Customers.Add(customer);  }

  public int Save()
  {    return _context.SaveChanges();  }

  public Customer FindById(int id)
  {    return _context.Customers.Find(id);  }
}
```

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

Figure 4 **Output from Failed Test Showing Status of Each Step**

```
Test Name:          HappyPath
Test Outcome:       Failed
Result Message:
Test method UnitTestProject1.UserStories.Add.AddCustomerFeature.HappyPath
threw exception:
System.NotImplementedException: The method or operation is not implemented.

Result StandardOutput:
Given a user has entered information about a customer
-> done: Steps.GivenAUserHasEnteredInformationAboutACustomer() (0.0s)
And she has provided a first name and a last name as required
-> done: Steps. GivenSheHasProvidedAFirstNameAndALastNameAsRequired() (0.0s)
When she completes entering more information
-> done: Steps.WhenSheCompletesEnteringMoreInformation() (0.0s)
Then that customer should be stored in the system
-> error: The method or operation is not implemented.
```

Now when I rerun the HappyPath test, it passes and all of my steps are marked as done. But I'm still not happy.

Make Sure Those Integration Tests Understand EF Behavior

Why am I not happy when my tests pass and I see the pretty green circle? Because I know that the test is not truly proving that the customer was stored.

In the ThenThatCustomerShouldBeStoredInTheSystem method, comment out the call to Save and run the test again. It still passes. And I didn't even save the customer into the database! Now do you smell that funny odor? It's the smell of what's known as a "false positive."

The problem is that the DbSet Find method that I'm using in my repository is a special method in Entity Framework that first checks the in-memory objects being tracked by the context before going to the database. When I called Add, I made the CustomerContext aware of that customer instance. The call to Customers.Find discovered that instance and skipped a wasted trip to the database. In fact, the ID of the customer is still 0 because it hasn't been stored yet.

So, because I'm using Entity Framework (and you should take into account the behavior of any object-relational mapping [ORM] framework you're using), I have a simpler way to test to see if the customer truly got into the database. When the EF SaveChanges instruction inserts the customer into the database, it will pull back the new database-generated ID of the customer and apply it to the instance that it inserted. Therefore, if the new customer's ID is no longer 0, I know my customer really did get into the database. I don't have to re-query the database.

I'll revise the Assert for that method accordingly. Here's the method I know will do a proper test:

```
[Then(@"that customer should be stored in the system")]
  public void ThenThatCustomerShouldBeStoredInTheSystem()
  {
    _repository = new CustomerRepository();
    _repository.Add(_newCustomer);
    _repository.Save();
    Assert.IsNotNull(_newCustomer.Id>0);
  }
```

It passes, and I know it's passing for the right reasons. It's not uncommon to define a failing test, for example, using Assert.IsNull(FindById(customer.Id) to make sure you're not passing for the wrong reason. But in this case, the problem still wouldn't have shown up until I removed the call to Save. If you're not confident about how EF works, it would be wise to also create some specific integration tests, unrelated to the user stories, to ensure that your repositories are behaving the way you expect.

Behavior Test or Integration Test?

As I traversed the learning curve of working out this first SpecFlow scenario, I encountered what I thought was a slippery slope. My scenario states that the customer should be stored in "the system."

The problem is that I wasn't confident about the definition of the system. My background tells me the database or at least some persistence mechanism is a very important part of the system.

The user doesn't care about repositories and databases—just her application. But she won't be very happy if she logs back into her application and can't find that customer again because it never really got stored in the database (because I didn't think that _repository.Save was necessary for fulfilling her scenario).

I consulted with yet another Dennis, Dennis Doomen, the author of Fluent Assertions and a heavy practitioner of BDD, TDD and more in large enterprise systems. He confirmed that, as a developer, I should certainly apply my knowledge to the steps and tests even if this means going beyond the intent of the user who defined the original scenario. Users provide their knowledge and I add mine in a way that doesn't push my technical perspective into the user's face. I continue to speak her language and communicate well with her.

Keep Digging into BDD and SpecFlow

I'm pretty sure that if it weren't for all of the tools that have been built to support BDD, my path into it wouldn't have been so easy. Even though I'm a data geek, I care very much about working with my clients, understanding their businesses and ensuring they have a happy experience using the software I help build for them. This is why Domain Driven Design and Behavior Driven Design speak so loudly to me. I think many developers feel the same way—even if it's deep down in their gut (or heart)—and may also be inspired by these techniques.

Besides the friends who helped me get to this point, here are some of the resources I found useful. The *MSDN Magazine* article, "Behavior-Driven Development with SpecFlow and WatiN," which can be found at msdn.microsoft.com/magazine/gg490346, was quite helpful. I also watched an excellent module from David Starr's Test First Development course on Pluralsight.com. (In fact, I watched that module a number of times.) I found the Wikipedia entry on BDD (bit.ly/LCgkxf) to be interesting in that it presented the bigger picture of BDD's history and where it fits in with other practices. And I'm eagerly awaiting the book, "BDD and Cucumber," which Paul Rayner (who also advised me here) is coauthoring. ■

Julie Lerman *is a Microsoft MVP, .NET mentor and consultant who lives in the hills of Vermont. You can find her presenting on data access and other Microsoft .NET topics at user groups and conferences around the world. She blogs at thedatafarm.com/blog and is the author of "Programming Entity Framework" (2010) as well as a Code First edition (2011) and a DbContext edition (2012), all from O'Reilly Media. Follow her on Twitter at twitter.com/julielerman.*

Thanks to the following technical experts for reviewing this article: *Dennis Doomen (Aviva Solutions) and Paul Rayner (Virtual Genius)*

Aspose.Total

Every Aspose component combined in one powerful suite.

Powerful File Format Components and Controls

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.

Aspose.Diagram
VSD, VSDX, VSS, VST, VSX & other formats.

... and many others!

Try Free Today!

- Aspose.Total for .NET
- Aspose.Total for Java
- Aspose.Total for SharePoint
- Aspose.Total for SSRS
- Aspose.Total for JasperReports
- Aspose.Total for Cloud

BRUNO TERKALY AND
RICARDO VILLALOBOS

Metering and Autoscaling Multi-Tenant Applications in Windows Azure

In our previous column (msdn.microsoft.com/magazine/dn201743), we introduced concepts relating to creating multi-tenant apps, covering two of the four pillars that should be considered when building this type of system: identity and security, and data isolation and segregation. This month, we concentrate on two other important areas that are deeply interwined: metering and autoscaling. Metering allows companies to gather information about the different components that are being shared among all the tenants; autoscaling guarantees that the end-user experience isn't affected during periods of high traffic, and that servers are deprovisioned when resource demand is lower.

Gathering information on resource usage is common when troubleshooting applications, especially during the development and testing processes. By doing this, thresholds and hardware requirements that will guarantee optimal performance of the solution can be set, and minimum hardware requirements can be recommended. In Windows, this task is accomplished by using performance counters that help determine system bottlenecks and error conditions.

Metering becomes particularly important when running multi-tenant solutions in the cloud, and not only during the development stages. Supporting multiple users sharing common resources presents specific challenges, such as how to enforce quotas for tenants, identify any users who might be consuming excessive resources, or decide if the pricing tiers need to be redefined. Keep in mind that metering multi-tenant solutions is not only about determining or validating the usage bill from your cloud provider—in this case Windows Azure—but also about optimizing resources in your deployment that guarantee the level of service tenants expect, typically expressed in a service-level agreement (SLA).

In this article, we'll concentrate on metering and autoscaling the compute portion of a multi-tenant solution, which is the solution type that's most affected by variations in the number of users accessing the application. Now that Windows Azure supports multiple cloud deployment models (Cloud Services, Virtual Machines and Web Sites), it's important to understand the different logging and diagnostics options that each one offers, and how they can be autoscaled based on this information. In case you need to better understand the basic differences, benefits and limitations of these deployment models, you'll find a good guide at bit.ly/Z7YwX0.

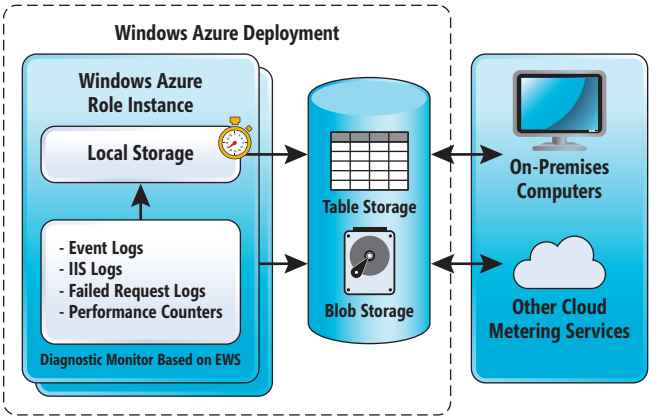


Figure 1 **Windows Azure Diagnostics for Cloud Services**

Collecting Data from Windows Azure Cloud Services

Cloud Services (based on the Platform as a Service concept), collects data via the Windows Azure Diagnostics (WAD) infrastructure, which is built on the Event Tracing for Windows (ETW) framework. Because Cloud Services is based on stateless virtual machines (VMs), WAD allows you to save data locally and, based on a schedule, transfer it to a central repository in Windows Azure storage using blobs and tables. Once the diagnostics data has been collected from the multiple instances in the role, it can be analyzed and used for multiple purposes. **Figure 1** shows how this process works.

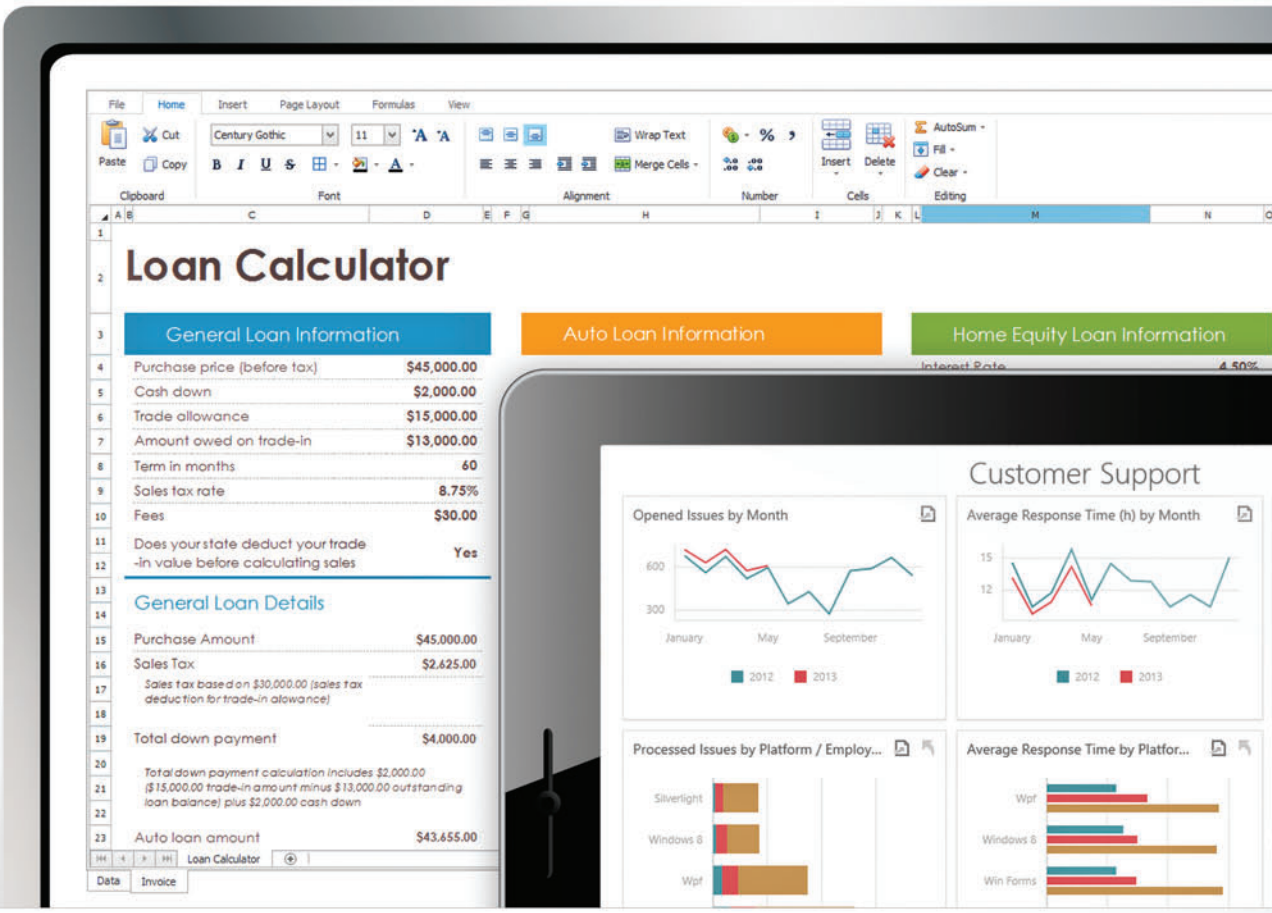
Figure 2 **Basic Service Definition and Diagnostics Configuration Files for Cloud Services**

```
<?xml version="1.0" encoding="utf-8"?>
<ServiceDefinition name="MyHostedService" xmlns=
  "http://schemas.microsoft.com/ServiceHosting/2008/10/ServiceDefinition"
  schemaVersion="2012-10.1.8">
  <WebRole name="WebRole1">
    <!--<Sites> ... </Sites> -->
    <!-- <Endpoints> ... </Endpoints> -->
    <Imports>
      <Import moduleName="Diagnostics" />
    </Imports>
  </WebRole>
</ServiceDefinition>

<?xml version="1.0" encoding="utf-8" ?>
<DiagnosticMonitorConfiguration xmlns=
  "http://schemas.microsoft.com/ServiceHosting/2010/10/DiagnosticsConfiguration"
  configurationChangePollInterval="PT1M"
  overallQuotaInMB="4096">
  <Directories bufferQuotaInMB="0" scheduledTransferPeriod="PT30M">
    <IISLogs container="wad-iis" directoryQuotaInMB="0" />
  </Directories>
</DiagnosticMonitorConfiguration>
```

DevExpress™

When only the best will do.

High-Performance, Elegant and Feature-Complete .NET Controls and Libraries



Download your FREE 30-day trial today. **DevExpress.com/try**

WIN WinForms · ASP ASP.NET · WPF WPF · SL Silverlight · Windows 8 XAML · HTML JS

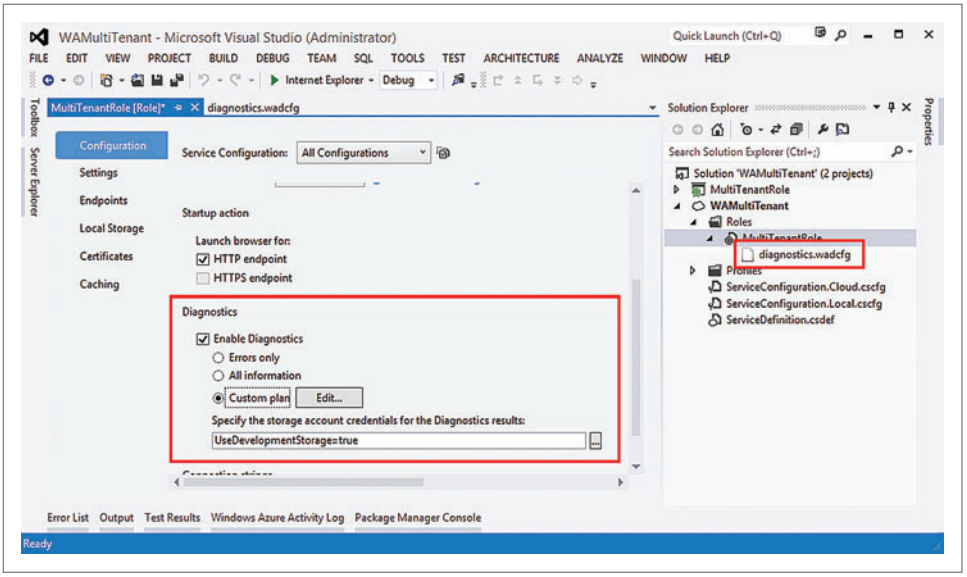


Figure 3 **New Diagnostics Configuration Options in Windows Azure SDK 2.0**

To learn more about other types of data sources via the configuration file—including Windows event logs and performance counters, among others—you can review the Windows Azure documentation at bit.ly/GTXAvo. Also, starting with version 2.0 of the Windows Azure SDK, the process of configuring diagnostics in Visual Studio has been greatly improved. The diagnostics section now offers a Custom plan that can be modified to include one or more data sources to be logged and transferred to the specified Windows Azure storage account (**Figure 3**).

By clicking on the Edit button, you can define specific data to be collected, including Windows performance counters, event logs and log directories (**Figure 4**). In this example, diagnostics information for percentage of processor time being used, available megabytes of memory, and number of requests per second will be collected and transferred to the Windows storage account every 30 minutes (the transfer period setting). This new interface simplifies the process of configuring diagnostics for cloud services and obtaining metering data for later use.

To enable diagnostics for Cloud Services, the corresponding module should be imported into the role deployment (via the ServiceDefinition.csdef file), and then enabled via the WAD configuration file (diagnostics.wadcfg). Another approach is to programmatically configure diagnostics inside the OnStart method for the role, but using the configuration file is preferred because it's loaded first and errors related to startup tasks can be caught and logged. Also, changes to the configuration don't require the code to be rebuilt. **Figure 2** shows the most basic version of the service definition and diagnostics configuration files.

The configurationChangePollInterval attribute defines how often an instance checks for configuration changes, while the scheduledTransferPeriod specifies the interval for the local files to be transferred to Windows Azure Storage (in the example shown in **Figure 2**, to the "wad-iis" blob container). Consider that one minute (PT1M) is the default and minimum value for the scheduled transfer of files parameter, but that it might be overkill for most scenarios. The overallQuotaInMB attribute defines the total amount of file system storage allocated for logging buffers. The bufferQuotaInMB attribute for each data source can either be left at the default of zero—which means it's less than the overallQuotaInMB property—or it can be explicitly set. The OverallQuotaInMB must be less than the sum of all the bufferQuotainMB properties.

Even though different data sources can be used to collect diagnostics information for cloud services, using them to determine which specific tenants are consuming most of the compute resources isn't easy. The closest metric that can be used for this purpose is provided by the IIS World Wide Web Consortium (W3C) logs, assuming traffic from the different users and tenants is tracked via URL parameters or specific virtual directories. In order to activate it, you can add the IISLogs XML node to the diagnostics configuration file (also included in **Figure 2**), but be warned that these IIS logs can get huge quickly. Keep in mind that diagnostics information is stored in a Windows Azure storage account, and configuration changes can be made on deployed and running services.

In addition to the Cloud Services monitoring options provided by the Windows Azure platform, you might want to take a look at the Cloud Ninja Metering Block released by the Windows Azure Incubation team, which encompasses many of these features in an easy-to-use library. It's available at cnmb.codeplex.com.

Collecting Data from Windows Azure Virtual Machines

Virtual Machines are stateful instances running on the Windows Azure platform and can be deployed individually or connected to Cloud Services via virtual networks. Because these instances

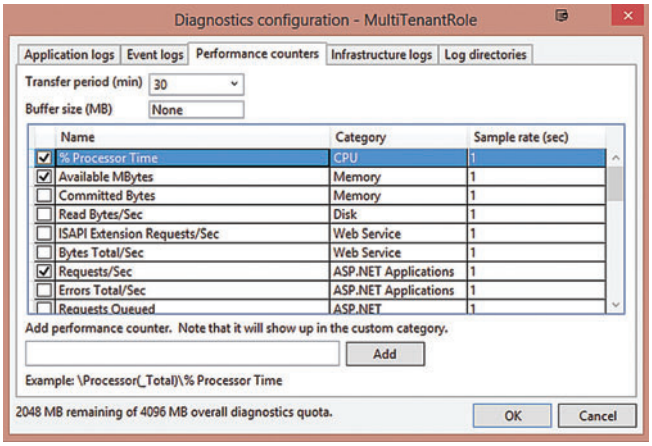


Figure 4 **Configuring Performance Counters in Visual Studio with the Windows Azure SDK 2.0**

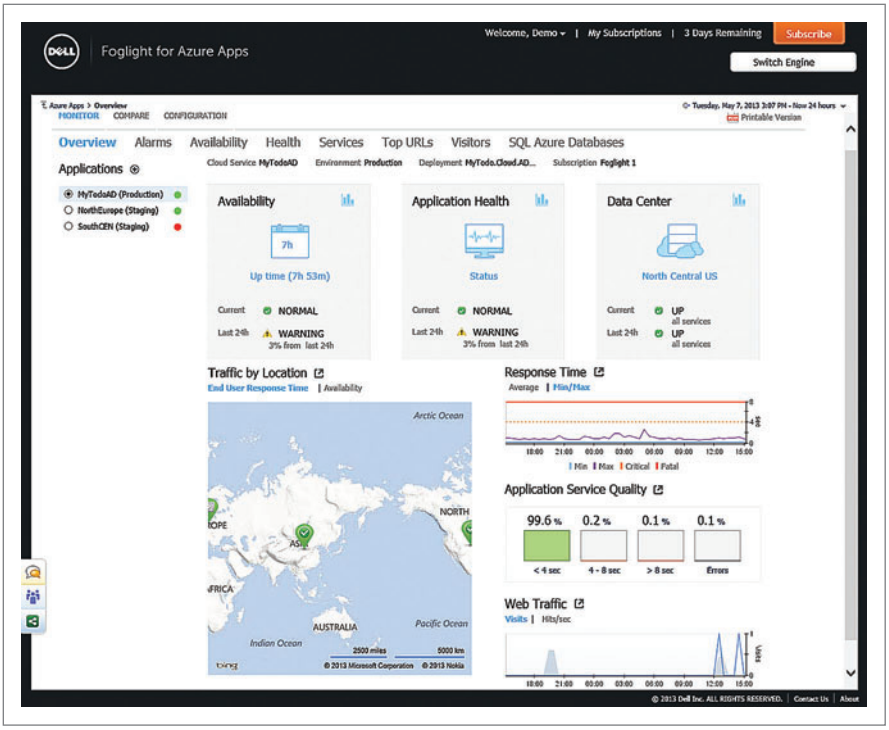


Figure 5 **The Dell Foglight Monitoring Portal**

run full versions of Windows and Linux, gathering diagnostics information is similar to the process for on-premises machines, using performance counters and persisting to local storage. The process of extracting this information varies, but it's usually accomplished by installing local agents that transfer this information to external services.

Collecting Data from Windows Azure Web Sites

Now let's turn our attention to Windows Azure Web Sites. Gathering diagnostics information from Web Sites is a simple process that can be enabled directly in the management portal. For the purpose of monitoring multi-tenant applications, Web Server Logging (the W3C extended log file format) should be activated, and log files downloaded via FTP. Here are the steps to follow:

1. Access manage.windowsazure.com.
2. Select Web Sites, and then the specific site that needs to be configured.
3. Click on Configure and scroll down to the "site diagnostics" section. Turn on Web Server Logging.
4. You can download the logs from /LogFiles/http/RawLogs. Log Parser 2.2, available from the Microsoft Download Center (bit.ly/119mefJ), can be used to parse and query IIS logs.

As with Windows Azure Cloud Services, information from the log files can be used to determine the usage of resources by different tenants, by tracking either URL parameters or individual virtual directories.

Metering as a Service

In addition to the diagnostics options natively provided by Windows Azure, a few companies offer metering services for Windows Azure. For example, Dell Inc. has created a product called Foglight that delivers real-time data on the health of applications and ties it back to the UX. It also includes a notification service that alerts developers of critical problems. Today, Foglight supports Cloud Services and Windows Azure SQL Database, based on the WAD infrastructure, as shown in **Figure 5**.

Autoscaling Options

Once the metering and performance counter data has been collected, it can be used to determine the level of provisioning that's needed to meet the performance requirements of the application. Autoscaling in Windows Azure refers to the act of adding or subtracting instances from a specific deployment (scaling out), with the idea of keeping solutions up and running for the lowest possible cost. Even though it's possible to scale up (increase the resources for a single machine), this usually implies application downtime, which is never desirable. There are basically three ways to autoscale a Windows Azure deployment.

Use an Autoscaling Block One approach to autoscaling a Windows Azure deployment, which specifically applies to Windows Azure Cloud Services, is to add an autoscaling application block to the solution. There are a couple of ready-to-be-used libraries for this purpose. One library is part of the Enterprise Integration Pack for Windows Azure, and it uses a collection of user-defined rules, setting limits for the minimum and maximum number of role instances in the deployment based on counters or measurements collected by WAD. This approach has been extensively documented by the Microsoft patterns & practices team, and can be found at bit.ly/18cr5mD. **Figure 6** shows a basic multi-tenant architecture with an autoscaling block added to the solution.

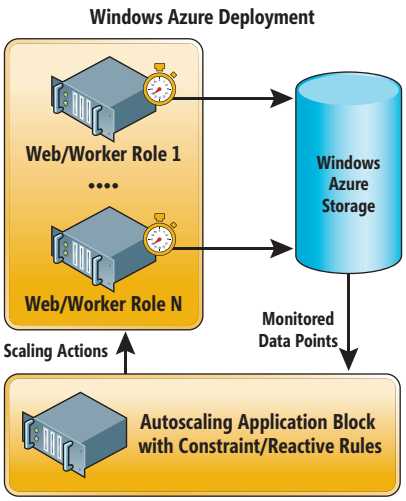


Figure 6 **Using an Autoscaling Application Block Approach for Cloud Services**

Use an External Service There are some scaling-out services available for Windows Azure deployments that act as external autoscaling application blocks. Microsoft recently acquired MetricsHub (metricshub.com), which provides a free monitoring and autoscaling service for Windows Azure subscribers. The logic for scaling out is based on sustained averages, leading indicators, trailing data and specific schedules. You can add the service directly

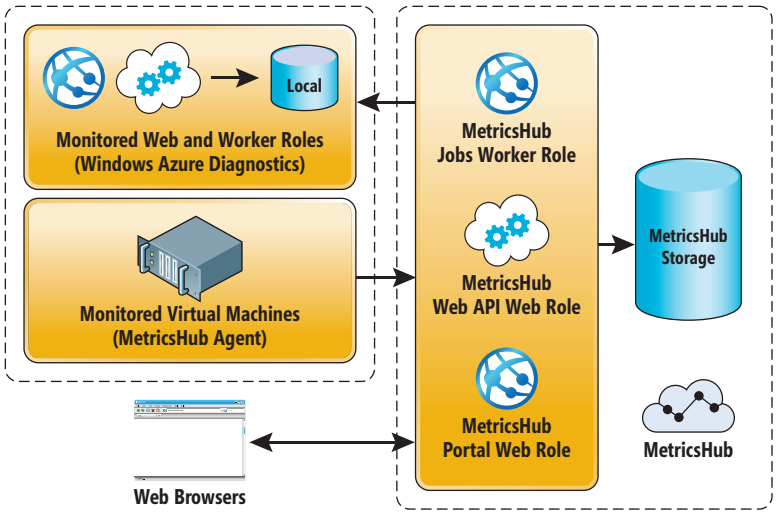


Figure 7 **MetricsHub Architecture**

from the management portal in the Add-Ons section (Windows Azure Store). MetricsHub supports both Windows Azure Cloud Services and Windows Azure Virtual Machines, based on an architecture that extracts information from WAD and receives information from agents installed on single stateful instances (see **Figure 7**).

Once the service has been set up, the MetricsHub portal offers different thresholds for maintaining a healthy cloud environment, based on parameters such as target CPU range and number of messages in a queue. It also provides a cost forecast before and after applying the autoscaling options, truly automating the provisioning process in the smartest way possible, balancing cost with performance (see **Figure 8**).

Use Automated Windows PowerShell Scripts The third method is based on Windows PowerShell scripts that are manually created and directly executed against the Windows Azure Management API. This approach provides a high level of control and flexibility, because these scripts can be used inside custom applications or continuous integration frameworks. Moreover, Windows PowerShell cmdlets for Windows Azure support the three deployment models, including the automation of the provisioning process for Windows Azure Web Sites. For example, changing the number of instances for a specific deployment is as easy as executing the following command:

```
PS C:\> Set-AzureWebsite -Name {WebSiteName} -NumberOfWorkers
{Instances}
```

For instructions on how to set up and install Windows Azure cmdlets, see bit.ly/QqctsU. You can find documentation on how to use each of the cmdlets at bit.ly/U0vOEp.

Wrapping Up

This article concludes our two-part series on building multi-tenant solutions in Windows Azure. In addition to identity and data isolation in the first article, we introduced you to the process of configuring and extracting performance information from each of the Windows Azure deployment models—Cloud Services, Virtual Machines and Web Sites. At the same time, we analyzed three different ways of autoscaling deployments via internal and external components and services. By taking advantage of the cloud economic model—which is based on usage cost and pools of shared resources—more companies are releasing solutions that can be efficiently adapted to their needs. ■

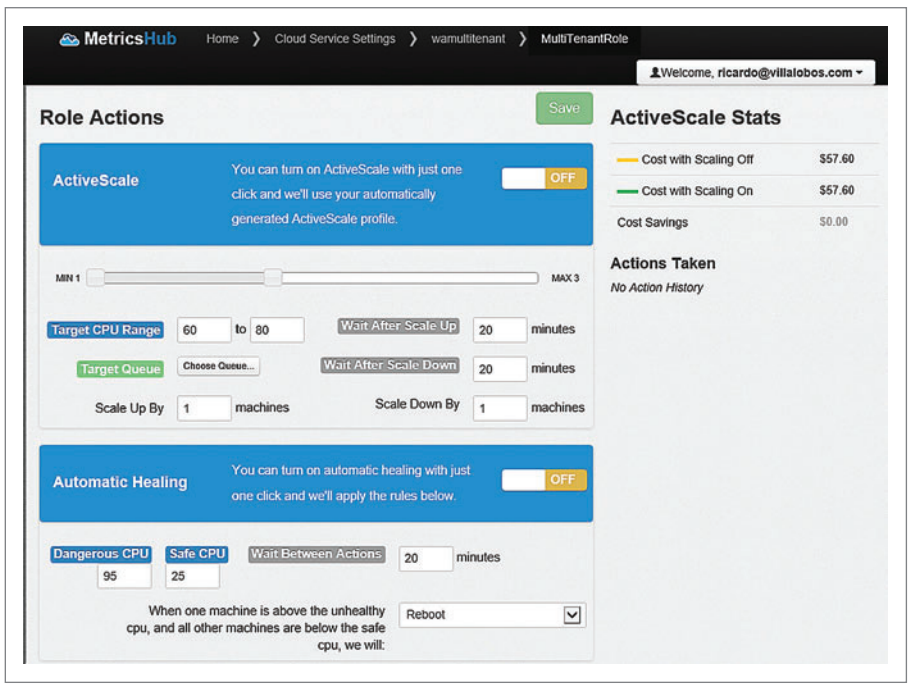


Figure 8 **The MetricsHub Architecture Autoscaling Portal**

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

Ricardo Villalobos *is a seasoned software architect with more than 15 years of experience designing and creating applications for companies in the supply chain management industry. Holding different technical certifications, as well as a master's degree in business administration from the University of Dallas, he works as a cloud architect in the Windows Azure CSV incubation group for Microsoft. You can read his blog at blog.ricardovillalobos.com.*

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

Thanks to the following technical expert for reviewing this article:
Trent Swanson (Full Scale 180)

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

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 SPONSORS esri NETPath

SUPPORTED BY Visual Studio msdn magazine Visual Studio MAGAZINE

PRODUCED BY 1105 MEDIA

REDMOND, WA | AUGUST 19-23, 2013

MICROSOFT CAMPUS

REGISTER BEFORE JULY 17 & SAVE $300

USE PROMO CODE REDJUL4

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 REDJUL4

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** - What's New in Windows 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 LOB Apps in Windows 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 SharePoint Hosted Apps as Single Page Apps - *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, Custom Forms, Tasks, Events and Workflow CSOM - *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** - WebMatrix 3: The Code Editor for the Cloud – *Justin Beckwith*
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*

How Microsoft's Next-Gen Compiler Project Can Improve Your Code

Jason Bock

I believe every developer wants to write good code. Nobody wants to create bug-ridden, unmaintainable systems that require endless hours to add features or fix problems. I've been on projects that felt like they were in a constant state of chaos, and they're not fun. Long hours are lost in a code base that's barely comprehensible due to inconsistent approaches. I like being on projects where layers are well-defined, unit tests are in abundance and build servers are constantly running to ensure everything works. Projects like that usually have a set of guidelines and standards in place that the developers follow.

I've seen teams put such guidelines in place. Maybe the developers are supposed to avoid calling certain methods in their code because they've been deemed problematic. Or maybe they want to make sure the code follows the same patterns in certain situations. For example, developers on projects may agree on standards like these:

- No one should use local DateTime values. All DateTime values should be in Universal Time Coordinate (UTC).
- The Parse method found on value types (such as int.Parse) should be avoided; int.TryParse should be used instead.
- All of the entity classes created should support equality—that is, they should override Equals and GetHashCode and implement the == and != operators, and the IEquatable<T> interface.

I'm sure you've seen similar rules in a standards document. Consistency is a good thing, and if everyone follows the same practices, it becomes easier to maintain the code. The trick is to quickly express that knowledge to all the developers on the team in a reusable, effective way.

Code reviews are one way to find potential issues. It's common for people with a fresh perspective on a given implementation to see issues that the original author is not aware of. Having another set of eyes review what you did can be beneficial, especially when the reviewer isn't familiar with the work. However, it's still easy to miss issues during development. Furthermore, code reviews are time-consuming—developers have to spend hours reviewing code and meeting with other developers to communicate the problems they find. I want a process that's quicker. I want to know as soon as possible that I've done something wrong. Failing as fast as possible saves time and money in the long run.

There are tools in Visual Studio, such as Code Analysis, that can analyze your code and inform you of potential problems. Code

Microsoft "Roslyn" is still in CTP mode; all information is subject to change.

This article discusses:

- Rules and tools for developing good code
- The Microsoft "Roslyn" project
- Syntax trees
- Finding and solving code issues

Technologies discussed:

Microsoft "Roslyn," Visual Studio 2012

Code download available at:

archive.msdn.microsoft.com/mag201307Roslyn

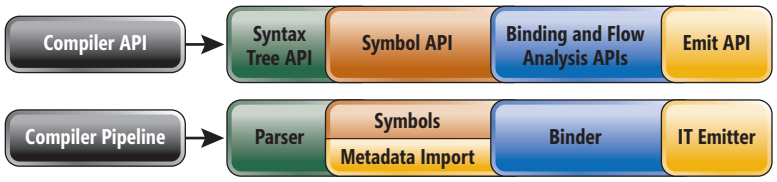


Figure 1 **The Roslyn Compiler Pipeline**

Analysis has a number of predefined rules that can uncover cases where you haven't disposed your object, or when you have unused method arguments. Unfortunately, Code Analysis doesn't run its rules until compilation is complete, and that's not soon enough! I want to know as soon as I'm typing that my new code has a mistake in it according to my standards. Failing as fast as I can is a good thing. Time (and therefore money) is saved, and I avoid committing code that can potentially lead to numerous problems in the future. To do that, I need to be able to codify my rules such that they're executed as I type, and that's where the Microsoft "Roslyn" CTP comes into play.

What's Microsoft "Roslyn"?

One of the best tools .NET developers can use to analyze their code is the compiler. It knows how to parse code into tokens, and turn those tokens into something that's meaningful based on their placement in the code. The compiler does this by emitting an assembly to disk as its output. There's a lot of hard-won knowledge that's gleaned in the compilation pipeline that you'd love to be able to use, but, alas, that's not possible in the .NET world because the C# and Visual Basic compilers don't provide an API for you to access. This changes with Roslyn. Roslyn is a set of compiler APIs that provides you with full access to every stage the compiler moves through. **Figure 1** is a diagram of the different stages in the compiler process that are now available in Roslyn.

Even though Roslyn is still in CTP mode (I used the September 2012 version for this article), it's worth taking the time to investigate the functionality available in its assemblies, and to learn what you can do with Roslyn. A good place to start is to look at its scripting facility. With Roslyn, C# and Visual Basic code are now scriptable. That is, there's a scripting engine available in Roslyn into which you can input snippets of code. This is handled via the ScriptEngine class. Here's a sample that illustrates how this engine can return the current DateTime value:

```
class Program
{
  static void Main(string[] args)
  {
    var engine = new ScriptEngine();
    engine.ImportNamespace("System");
    var session = engine.CreateSession();
    Console.Out.WriteLine(session.Execute<string>(
      "DateTime.Now.ToString();"));
  }
}
```

In this code, the engine is creating and importing the System namespace so Roslyn will be able to resolve what DateTime means. Once a session is created, all it takes is to call Execute, and then Roslyn will parse the given code. If it can parse it correctly, it will run it and return the result.

Making C# into a scripting language is a powerful concept. Even though Roslyn is still in CTP mode, people are creating amazing projects and frameworks using its bits, such as scriptcs (scriptcs.net). However, where I think Roslyn really shines is in letting you create Visual Studio extensions to warn you of issues while you write code. In the previous snippet, I used DateTime.Now. If I were on a project that enforced the first bullet point I made at the beginning of the article, I'd be in violation of that standard. I'll explore how that rule can be enforced using Roslyn. But before I do that, I'll cover the first stage of compilation: parsing code to get tokens.

Syntax Trees

When Roslyn parses a piece of code, it returns an immutable syntax tree. This tree contains everything about the given code, including trivia such as spaces and tabs. Even if the code has errors, it'll still try as best it can to give you as much information as possible.

This is all well and good, but how do you figure out where in the tree the pertinent information is? Currently, the documentation on Roslyn is fairly light, which is understandable given that it's still a CTP. You can use the Roslyn forums to post questions (bit.ly/16qNf7w), or use the #RoslynCTP tag in a Twitter post. There's also a sample called SyntaxVisualizerExtension when you install the bits, which is an extension for Visual Studio. As you type code in the IDE, the visualizer automatically updates with the current version of the tree.

This tool is indispensable in figuring out what you're looking for and how to navigate the tree. In the case of using .Now on the DateTime class, I figured out that I needed to find MemberAccessExpression (or, to be precise, a MemberAccessExpressionSyntax-based object), where the last IdentifierName value equals Now. Of course, that's for the simple case where you'd type "var now = DateTime.Now;"—you could put "System." in front of DateTime, or use "using DT = System.DateTime;"; furthermore, there may be a property in the system in a different class called Now. All of the cases must be processed correctly.

Figure 2 **Finding DateTime.Now Usages**

```
public IEnumerable<CodeIssue> GetIssues(
  IDocument document, CommonSyntaxNode node, CancellationToken cancellationToken)
{
  var memberNode = node as MemberAccessExpressionSyntax;

  if (memberNode.OperatorToken.Kind == SyntaxKind.DotToken &&
    memberNode.Name.Identifier.ValueText == "Now")
  {
    var symbol = document.GetSemanticModel().GetSymbolInfo(memberNode.Name).Symbol;

    if (symbol != null &&
      symbol.ContainingType.ToDisplayString() ==
        Values.ExpectedContainingDateTimeTypeDisplayString &&
      symbol.ContainingAssembly.ToDisplayString().Contains(
        Values.ExpectedContainingAssemblyDisplayString))
    {
      return new [] { new CodeIssue(CodeIssueKind.Error,
        memberNode.Name.Span,
        "Do not use DateTime.Now",
        new ChangeNowToUtcNowCodeAction(document, memberNode))};
    }
  }

  return null;
}
```

Finding and Solving Code Issues

Now that I know what to find, I need to create a Roslyn-based Visual Studio extension to hunt down DateTime.Now property usage. To do this, you simply select the Code Issue template under the Roslyn option in Visual Studio.

Once you do this, you'll get a project that contains one class called CodeIssueProvider. This class implements the ICodeIssueProvider interface, though you don't have to implement each of its four members. In this case, only the members that work with SyntaxNode types are used; the others can throw NotImplementedException. You implement the SyntaxNodeTypes property by specifying which syntax node types you want to handle with the corresponding GetIssues method. As was mentioned in the previous example, MemberAccessExpressionSyntax types are the ones that matter. The following code snippet shows how you implement SyntaxNodeTypes:

```
public IEnumerable<Type> SyntaxNodeTypes
{
  get
  {
    return new[] {
typeof(MemberAccessExpressionSyntax) };
  }
}
```

This is an optimization for Roslyn. By having you specify which types you care to examine in more detail, Roslyn doesn't have to call the GetIssues method for each syntax type. If Roslyn didn't have this filtering mechanism in place and called your code provider for every node in the tree, the performance would be appalling.

Now all that's left is to implement GetIssues such that it will only report use of the Now property. As I mentioned in the previous section, you only want to find cases where Now has been used on DateTime. When you're using tokens, you don't have a lot of information besides the text. However, Roslyn provides what's called a semantic model, which can provide a lot more information about the code under examination. The code in **Figure 2** demonstrates how you can find DateTime.Now usages.

You'll notice the cancellationToken argument isn't used, nor is it used anywhere in the sample project that accompanies this article. This is a deliberate choice, because putting code into the sample that constantly checks the token to see if the processing should stop can be distracting. But if you're going to create Roslyn-based extensions that are production-ready, you should make sure you check the token often and stop if the token is in the canceled state.

Once you've determined that your member access expression is trying to get a property called Now, you can get symbol information for that token. You do this by getting the semantic model for the tree, and then you get a reference to an ISymbol-based object via the Symbol property. Then, all you have to do is get the containing type and see if its name is System.DateTime and if its containing assembly name includes mscorlib. If that's the case, that's the issue you're looking for, and you can flag it as an error by returning a CodeIssue object.

This is good progress so far, because you'll see a red squiggly error line underneath the Now text in the IDE. But it doesn't go far enough. It's nice when the compiler tells you your code is missing a semicolon or a curly brace. Getting error information is better than nothing at all, and with simple errors it's usually pretty easy to fix them based on the error message. However, wouldn't it be nice if tools could just figure out errors all by themselves? I like being told when I'm wrong—and I'm much happier when the error message gives me detailed information explaining how I can fix the issue. And if that information could be automated such that a tool could resolve the issues for me, that's less time I have to spend on the problem. The more time saved, the better.

That's why you see in the previous code snippet a reference to a class called ChangeNowToUtcNowCodeAction. This class implements the ICodeAction interface, and its job is to change Now to UtcNow. The main method you have to implement is called GetEdit. In this case, the Name token in the MemberAccessExpressionSyntax object needs to be changed to a new token. As the following code shows, it's pretty easy to make this replacement:

```
public CodeActionEdit
GetEdit(CancellationToken cancellationToken)
{
  var nameNode = this.nowNode.Name;
  var utcNowNode =
    Syntax.IdentifierName("UtcNow");
  var rootNode = this.document.
    GetSyntaxRoot(cancellationToken);
  var newRootNode =
    rootNode.ReplaceNode(nameNode, utcNowNode);
  return new CodeActionEdit(
    document.UpdateSyntaxRoot(newRootNode));
}
```

Figure 3 **Finding Bad DateTime Constructor Calls**

```
public IEnumerable<CodeIssue> GetIssues(
  IDocument document, CommonSyntaxNode node, CancellationToken cancellationToken)
{
  var creationNode = node as ObjectCreationExpressionSyntax;
  var creationNameNode = creationNode.Type as IdentifierNameSyntax;

  if (creationNameNode != null && creationNameNode.Identifier.ValueText == "DateTime")
  {
    var model = document.GetSemanticModel();
    var creationSymbol = model.GetSymbolInfo(creationNode).Symbol;

    if (creationSymbol != null &&
      creationSymbol.ContainingType.ToDisplayString() ==
        Values.ExpectedContainingDateTimeTypeDisplayString &&
      creationSymbol.ContainingAssembly.ToDisplayString().Contains(
        Values.ExpectedContainingAssemblyDisplayString))
    {
      var argument = FindingNewDateTimeCodeIssueProvider.GetInvalidArgument(
        creationNode, model);

      if (argument != null)
      {
        if (argument.Item2.Name == "Local" ||
          argument.Item2.Name == "Unspecified")
        {
          return new [] { new CodeIssue(CodeIssueKind.Error,
            argument.Item1.Span,
            "Do not use DateTimeKind.Local or DateTimeKind.Unspecified",
            new ChangeDateTimeKindToUtcCodeAction(document, argument.Item1)) };
        }
      }
      else
      {
        return new [] { new CodeIssue(CodeIssueKind.Error,
          creationNode.Span,
          "You must use a DateTime constuctor that takes a DateTimeKind") };
      }
    }
  }

  return null;
}
```

All you need to do is create a new identifier with the UtcNow text, and replace the Now token with this new identifier via ReplaceNode. Remember that syntax trees are immutable, so you don't change the current document tree. You create a new tree, and return that tree from the method call.

With all of this code in place, you can test it out in Visual Studio by simply pressing F5. This launches a new instance of Visual Studio with the extension automatically installed.

Figure 4 **The GetInvalidArgument Method**

```
private static Tuple<ArgumentSyntax, ISymbol> GetInvalidArgument(
  ObjectCreationExpressionSyntax creationToken, ISemanticModel model)
{
  foreach (var argument in creationToken.ArgumentList.Arguments)
  {
    if (argument.Expression is MemberAccessExpressionSyntax)
    {
      var argumentSymbolNode = model.GetSymbolInfo(argument.Expression).Symbol;

      if (argumentSymbolNode.ContainingType.ToDisplayString() ==
        Values.ExpectedContainingDateTimeKindTypeDisplayString)
      {
        return new Tuple<ArgumentSyntax,ISymbol>(argument, argumentSymbolNode);
      }
    }
  }

  return null;
}
```

Analyzing DateTime Constructors

This is a good start, but there are more cases that have to be handled. The DateTime class has a number of constructors defined that can cause issues. There are two cases in particular to be aware of:

1. The constructor may not take a DateTimeKind enumeration type as one of its parameters, which means the resulting DateTime will be in the Unspecified state.
2. The constructor may take a DateTimeKind value with one of its parameters, which means you may specify an enumeration value other than Utc.

You can write code to find both conditions. However, I'll only create a code action for the second one.

Figure 3 lists the code for the GetIssues method in the ICodeIssue-based class that will find bad DateTime constructor calls.

It's very similar to the other issue. Once you know the constructor comes from a DateTime, you need to evaluate the arguments. (I'll explain what GetInvalidArgument does in a moment.) If you find an argument of the DateTimeKind type and it doesn't specify Utc, you have a problem. Otherwise, you know you're using a constructor that won't have the DateTime in Utc, so that's another issue to report. **Figure 4** shows what GetInvalidArgument looks like.

This search is very similar to the others. If the argument type is DateTimeKind, you know you have a potentially invalid argument value. To fix the argument, the code is virtually identical to the first code action you saw, so I won't repeat it here. Now, if other developers try to get around the DateTime.Now restriction, you can catch them in the act, and correct the constructor calls, too!

In the Future

It's wonderful to think about all the tools that will be created with Roslyn, but work still needs to be done. One of the biggest frustrations I think you'll have with Roslyn right now is the lack of documentation. There are good samples online and in the installation bits, but Roslyn is a large API set and it can be confusing finding out exactly where to start and what to use to accomplish a particular task. It's not uncommon to have to dig around for a while to figure out the right calls to use. The encouraging aspect is that I'm usually able to do something in Roslyn that seems fairly complex at first but ends up being fewer than 100 or 200 lines of code.

I believe that as Roslyn gets closer to release, everything surrounding it will improve. And I'm also convinced that Roslyn has the potential to underpin many frameworks and tools in the .NET ecosystem. I don't see every .NET developer using the Roslyn APIs on a day-to-day basis directly, but you'll probably end up using bits that use Roslyn at some level. This is why I'm encouraging you to dive into Roslyn and see how things work. Being able to codify idioms into reusable rules that every developer on a team can take advantage of helps everyone quickly produce better code. ■

Jason Bock *is a practice lead at Magenic (magenic.com) and recently coauthored "Metaprogramming in .NET" (Manning Publications, 2013). Reach him at jasonb@magenic.com.*

THANKS to the following technical expert for reviewing this article: *Kevin Pilch-Bisson (Microsoft)*

WORKFLOW APPLICATIONS | HELP DESK | BUG TRACKING

MADE EASY!

Alexsys Team® offers ***flexible task management*** software for Windows, Web, and Mobile Devices. Track whatever you need to get the job done - anywhere, anytime on practically any device!

Thousands are using Alexsys Team® to create workflow solutions and web apps - without coding!
Fortune 500 companies, state, local, DOD, and other federal agencies use Team to manage their tasks. Easily tailor Team to meet your exact requirements - even if they change daily, weekly, or even every minute.

Alexsys Team® Features Include:

- Form and Database customization
- Custom workflows
- Role-based security
- Adaptive Rules Engine
- DOD CAC card support
- Time recording
- Automated Escalations, Notifications, and SOAP Messaging
- Supports MS-SQL, MySQL, and Orcale databases

Our renowned tech support team is here to make you and your team a success. Don't have enough resources? Our professional services staff has helped companies large and small use Alexsys Team® at a fraction of the cost of those big consulting firms.

Find out yourself:
Free Trial and Single User FreePack™ available at Alexcorp.com

ALEXSYS TEAM®

LIVE EVENT

AMPLIFY YOUR KNOWLEDGE

Live! 360 is back to turn your conference experience up to 11. Spanning 5 days at the Royal Pacific Resort in sunny Orlando, Live! 360 gives you the ultimate IT and Developer line-up, offering hundreds of sessions on the most relevant topics affecting your business today.

SUPPORTED BY

Microsoft | SharePoint | Microsoft SQL Server 2012 | Visual Studio | msdn magazine | Visual Studio MAGAZINE

PRODUCED BY

1105 MEDIA

ORLANDO | **NOVEMBER 18-22, 2013**

Royal Pacific Resort at Universal Orlando

- 5 Days.
- 4 Events.
- 22 Tracks.
- 175+ Sessions.
- The Ultimate IT and Developer Line-up.

SUMMER SPECIAL!
SAVE $500!
REGISTER BEFORE AUGUST 7
Use Promo Code LIVE360

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

LIVE! 360

IT EVENTS WITH PERSPECTIVE

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

Calling all .NET Developers! We've got your ticket to Code for the final stop on the Visual Studio Live! 2013 Tour. **vslive.com/orlando**

SharePoint LIVE!

TRAINING FOR COLLABORATION

Collaborate and Listen. SharePoint Live! returns to rock its newest release, SharePoint 2013. **splive360.com**

SQL Server LIVE!

TRAINING FOR DBAs AND IT PROS

Bringing SQL Server to Your World. Fine tune your skills to solve your biggest data challenges at SQL Server Live!. **sqllive360.com**

Modern Apps LIVE!

MODERN APPS FROM START TO FINISH

The Future of Software Development is Back, Modern Apps Live! will break down the latest techniques in low cost, high value application development. **modernappslive.com**

CONNECT WITH LIVE!360

- twitter.com/@live360events
- facebook.com – Search "Live 360"
- linkedin.com – Join the "Live 360" group!

live360events.com

Leveraging Windows 8 Features with MVVM

Brent Edwards

Windows 8 brings many new features that developers can leverage to create compelling applications and a rich UX. Unfortunately, these capabilities aren't always very unit-test friendly. Features such as sharing and secondary tiles can make your app more interactive and enjoyable, but also less testable.

In this article, I'll look at different ways to let an application use features such as sharing, settings, secondary tiles, application settings and application storage. Using the Model-View-ViewModel (MVVM) pattern, dependency injection and some abstraction, I'll show you how to leverage these features while keeping the presentation layer unit-test friendly.

About the Sample App

To illustrate the concepts I'll be talking about in this article, I've used MVVM to write a sample Windows Store app that lets a user view blog posts from the RSS feed of his favorite blogs. The app illustrates how to:

- Share information about a blog post with other apps via the Share charm
- Change which blogs the user wants to read with the Settings charm
- Pin a favorite blog post to the Start screen for reading later with secondary tiles
- Save favorite blogs to view across all devices with roaming settings

In addition to the sample app, I've taken the specific Windows 8 functionality I'll be talking about in this article and abstracted it into an open source library called Charmed. Charmed can be used as a helper library or just as a reference. The goal of Charmed is to be a cross-platform MVVM support library for Windows 8 and Windows Phone 8. I'll be talking more about the Windows Phone 8 side of the library in a future article. Check out the progress of the Charmed library at bit.ly/17AzFxW.

My goal with this article and the sample code is to demonstrate my approach to testable applications with MVVM, using some of the new features that Windows 8 offers.

This article discusses:

- The basics of MVVM
- The Sharing and Settings contracts
- The message bus
- Roaming settings
- Secondary tiles

Technologies discussed:

Windows 8, Visual Studio

Code download available at:

archive.msdn.microsoft.com/mag201307MVVM

MVVM Overview

Before diving into the code and specific Windows 8 features, I'll take a quick look at MVVM. MVVM is a design pattern that has gained enormous popularity in recent years for XAML-based technologies, such as Windows Presentation Foundation (WPF), Silverlight, Windows Phone 7, Windows Phone 8 and Windows 8 (the Windows Runtime, or WinRT). MVVM breaks an application's architecture into three logical layers: Model, View Model and View, as shown in **Figure 1**.

The Model layer is where the business logic of the application lives—business objects, data validation, data access and so forth. In reality, the Model layer is typically broken up into more layers and possibly even multiple tiers. As **Figure 1** shows, the Model layer is the logical bottom, or foundation, of the application.

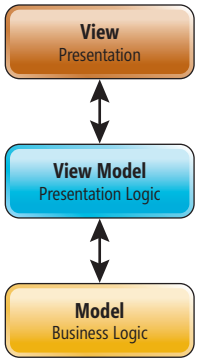


Figure 1 **The Three Logical Layers of Model-View-ViewModel**

The View Model layer holds the presentation logic of the application, which includes data to be displayed, properties to help enable UI elements or make them visible, and methods that will interact with both the Model and the View layers. Basically, the View Model layer is a view-agnostic representation of the current state of the UI. I say "view-agnostic" because it merely provides data and methods for the view to interact with, but it doesn't dictate how the view will represent that data or allow the user to interact with those methods. As **Figure 1** shows, the View Model layer logically sits between the Model layer and the View layer and can interact with both. The View Model layer contains code that would previously be in the codebehind of the View layer.

The View layer contains the actual presentation of the application. For XAML-based applications, such as those for the Windows Runtime, the View layer consists mostly, if not entirely, of XAML. The View layer leverages the powerful XAML data-binding engine to bind to properties on the view model, applying a look-and-feel to data that would otherwise have no visual representation. As **Figure 1** shows, the View layer is the logical top of the application. The View layer interacts directly with the View Model layer, but has no knowledge of the Model layer.

The main purpose of the MVVM pattern is to separate an application's presentation from its functionality. Doing so makes the application more conducive to unit tests because the functionality now lives in Plain Old CLR Objects (POCOs), rather than in views that have their own lifecycles.

Contracts

Windows 8 introduces the concept of contracts, which are agreements among two or more apps on a user's system. These contracts provide consistency across all apps, allowing developers to leverage functionality from any app that supports them. An app can declare what contracts it supports in the Package.appxmanifest file, as shown in **Figure 2**.

While it's optional to support contracts, it's generally a good idea. There are three contracts in particular that an application should support—Share, Settings and Search—because they're always available via the charms menu, shown in **Figure 3**.

I'll focus on two contract types: Sharing and Settings.

Sharing

The Share contract enables an app to share context-specific data with other apps on the user's system. There are two sides to the Share contract: the source and the target. The source is the app that's doing the sharing. It provides some data to be shared, in whatever format is necessary. The target is the app that receives the shared data. Because the Share charm is always available to the user via the charms menu, I want the sample app to be a share source, at the very least. Not every app needs to be a share target because not every app has a need to accept input from other sources. However, there's a pretty good chance that any given app will have at least one thing that's worth sharing with other apps. So, a majority of apps will likely find it useful to be a share source.

When the user presses the Share charm, an object called the Share Broker begins the process of taking the data an app shares (if any) and sending it to the share target as specified by the user. There's an object called the DataTransferManager that I can use to share data during that process. The DataTransferManager has an event called DataRequested, which is raised when the user presses the Share charm. The following code shows how to get a reference to the DataTransferManager and subscribe to the DataRequested event:

```
public void Initialize()
{
  this.DataTransferManager = DataTransferManager.GetForCurrentView();
  this.DataTransferManager.DataRequested += this.DataTransferManager_DataRequested;
}

private void DataTransferManager_DataRequested(
  DataTransferManager sender, DataRequestedEventArgs args)
{
  // Do stuff ...
}
```

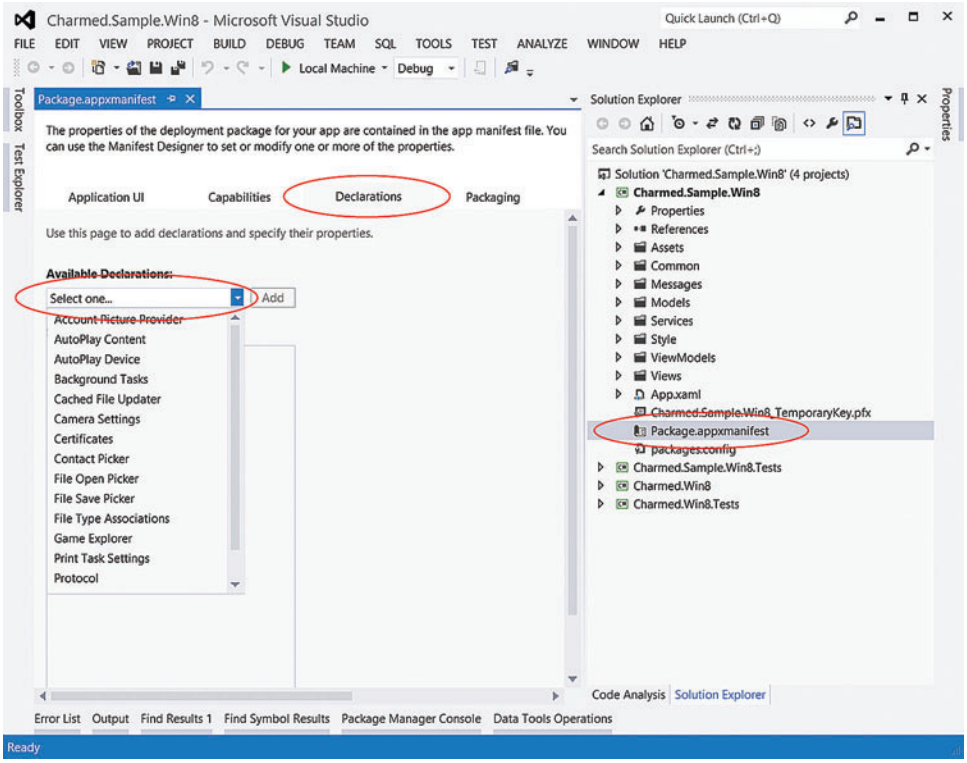


Figure 2 **Contracts in the Package.appxmanifest File**

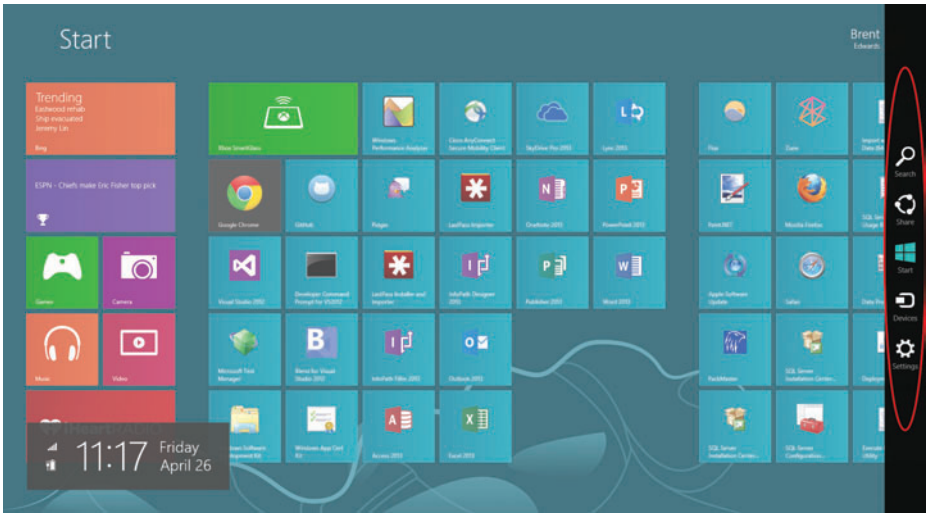


Figure 3 **The Charms Menu**

Calling DataTransferManager.GetForCurrentView returns a reference to the active DataTransferManager for the current view. While it's possible to put this code in a view model, it creates a hard dependency on the DataTransferManager, a sealed class that can't be mocked in unit tests. Because I really want my app to stay as testable as possible, this isn't ideal. A better solution is to abstract the DataTransferManager interaction out into a helper class and define an interface for that helper class to implement.

Before abstracting this interaction, I must decide what parts really matter. There are three parts of the interaction with the DataTransferManager I care about:

1. Subscribing to the DataRequested event when my view is activated.
2. Unsubscribing from the DataRequested event when my view is deactivated.
3. Being able to add shared data to the DataPackage.

With those three points in mind, my interface materializes:

```
public interface IShareManager
{
  void Initialize();
  void Cleanup();
  Action<DataPackage> OnShareRequested { get; set; }
}
```

Initialize should get a reference to the DataTransferManager and subscribe to the DataRequested event. Cleanup should unsubscribe from the DataRequested event. OnShareRequested is where I can define what method gets called when the DataRequested event has been raised. Now I can implement IShareManager, as shown in **Figure 4**.

When the DataRequested event is raised, the event args that come through contain a DataPackage. That DataPackage is where the actual shared data needs to be placed, which is why the Action for OnShareRequested takes a DataPackage as a parameter. With my IShareManager interface defined and ShareManager implementing it, I'm now ready to include sharing in my view model, without sacrificing the unit testability for which I'm aiming.

Once I've used my Inversion of Control (IoC) container of choice to inject an instance of IShareManager into my view model, I can put it to use, as shown in **Figure 5**.

LoadState is called when the page and view model are activated, and SaveState is called when the page and view model are deactivated. Now that the ShareManager is all set up and ready to handle sharing, I need to implement the ShareRequested method that will be called when the user initiates sharing. I want to share some info about a particular blog post (FeedItem), as shown in **Figure 6**.

I chose to share several different data types. This is generally a good idea because you have no control over what apps a user has on his system or what data types those apps support. It's important to remember that sharing is essentially a fire-and-forget scenario. You have no idea what app the user will choose to share to and what that app will do with the shared data. To share with the broadest possible audience, I provide a title, a URI, a text-only version and an HTML version.

Settings

The Settings contract allows the user to change context-specific settings in an app. These can be settings that affect the app as a whole, or just specific items that relate to the current context. Users of Windows 8 will become conditioned to using the Settings charm to make changes to the app, and I want the sample app to support it because it's always available to the user via the charms menu. In fact, if an app declares Internet capability via the Package.appxmanifest file, it *must* implement the Settings contract by providing a link to a Web-based privacy policy somewhere in

Figure 4 **Implementing IShareManager**

```
public sealed class ShareManager : IShareManager
{
  private DataTransferManager DataTransferManager { get; set; }

  public void Initialize()
  {
    this.DataTransferManager = DataTransferManager.GetForCurrentView();
    this.DataTransferManager.DataRequested +=
      this.DataTransferManager_DataRequested;
  }

  public void Cleanup()
  {
    this.DataTransferManager.DataRequested -=
      this.DataTransferManager_DataRequested;
  }

  private void DataTransferManager_DataRequested(
    DataTransferManager sender, DataRequestedEventArgs args)
  {
    if (this.OnShareRequested != null)
    {
      this.OnShareRequested(args.Request.Data);
    }
  }

  public Action<DataPackage> OnShareRequested { get; set; }
}
```

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 5 **Wiring up IShareManager**

```
public FeedItemViewModel(IShareManager shareManager)
{
  this.shareManager = shareManager;
}

public override void LoadState(
  FeedItem navigationParameter, Dictionary<string, object> pageState)
{
  this.shareManager.Initialize();
  this.shareManager.OnShareRequested = ShareRequested;
}

public override void SaveState(Dictionary<string, object> pageState)
{
  this.shareManager.Cleanup();
}
```

Figure 6 **Populating the DataPackage on ShareRequested**

```
private void ShareRequested(DataPackage dataPackage)
{
  // Set as many data types as possible.
  dataPackage.Properties.Title = this.FeedItem.Title;

  // Add a Uri.
  dataPackage.SetUri(this.FeedItem.Link);

  // Add a text-only version.
  var text = string.Format(
    "Check this out! {0} ({1})", this.FeedItem.Title, this.FeedItem.Link);
  dataPackage.SetText(text);

  // Add an HTML version.
  var htmlBuilder = new StringBuilder();
  htmlBuilder.AppendFormat("<p>Check this out!</p>", this.FeedItem.Author);
  htmlBuilder.AppendFormat(
    "<p><a href='{0}'>{1}</a></p>", this.FeedItem.Link, this.FeedItem.Title);
  var html = HtmlFormatHelper.CreateHtmlFormat(htmlBuilder.ToString());
  dataPackage.SetHtmlFormat(html);
}
```

the Settings menu. Because apps using Visual Studio 2012 templates automatically declare Internet capability right out of the box, this is something that should not be overlooked.

When a user presses the Settings charm, the OS begins dynamically building the menu that will be displayed. The menu and the associated flyout are controlled by the OS. I can't control what the menu and flyout look like, but I can add options to the menu. An object called SettingsPane will notify me when the user selects the Settings charm via the CommandsRequested event. Getting a reference to the SettingsPane and subscribing to the CommandsRequested event is quite straightforward:

```
public void Initialize()
{
  this.SettingsPane = SettingsPane.GetForCurrentView();
  this.SettingsPane.CommandsRequested += SettingsPane_CommandsRequested;
}

private void SettingsPane_CommandsRequested(
  SettingsPane sender, SettingsPaneCommandsRequestedEventArgs args)
{
  // Do stuff ...
}
```

The catch with this is another hard dependency. This time the dependency is SettingsPane, which is another class that can't be mocked. Because I want to be able to unit test the view model that uses SettingsPane, I need to abstract out references to it, just as I did for references to DataTransferManager. As it turns out, my interactions with SettingsPane are very similar to my interactions with DataTransferManager:

1. Subscribing to the CommandsRequested event for the current view.
2. Unsubscribing from the CommandsRequested event for the current view.
3. Adding my own SettingsCommand objects when the event is raised.

So, the interface I need to abstract looks a lot like the IShareManager interface:

```
public interface ISettingsManager
{
  void Initialize();
  void Cleanup();
  Action<IList<SettingsCommand>> OnSettingsRequested { get; set; }
}
```

Initialize should get a reference to the SettingsPane and subscribe to the CommandsRequested event. Cleanup should unsubscribe from the CommandsRequested event. OnSettingsRequested is where I can define what method gets called when the CommandsRequested event has been raised. Now I can implement ISettingsManager, as shown in **Figure 7**.

When the CommandsRequested event is raised, the event args ultimately give me access to the list of SettingsCommand objects that represent the Settings menu options. To add my own Settings menu options, I just need to add a SettingsCommand instance to

Figure 7 **Implementing ISettingsManager**

```
public sealed class SettingsManager : ISettingsManager
{
  private SettingsPane SettingsPane { get; set; }

  public void Initialize()
  {
    this.SettingsPane = SettingsPane.GetForCurrentView();
    this.SettingsPane.CommandsRequested += SettingsPane_CommandsRequested;
  }

  public void Cleanup()
  {
    this.SettingsPane.CommandsRequested -= SettingsPane_CommandsRequested;
  }

  private void SettingsPane_CommandsRequested(
    SettingsPane sender, SettingsPaneCommandsRequestedEventArgs args)
  {
    if (this.OnSettingsRequested != null)
    {
      this.OnSettingsRequested(args.Request.ApplicationCommands);
    }
  }

  public Action<IList<SettingsCommand>> OnSettingsRequested { get; set; }
}
```

Figure 8 **Wiring up ISettingsManager**

```
public ShellViewModel(ISettingsManager settingsManager)
{
  this.settingsManager = settingsManager;
}

public void Initialize()
{
  this.settingsManager.Initialize();
  this.settingsManager.OnSettingsRequested = OnSettingsRequested;
}

public void Cleanup()
{
  this.settingsManager.Cleanup();
}
```

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

Figure 9 **Showing the SettingsView on SettingsRequested with Callisto**

```
private void OnSettingsRequested(IList<SettingsCommand> commands)
{
  SettingsCommand settingsCommand =
    new SettingsCommand("FeedsSetting", "Feeds", (x) =>
  {
    SettingsFlyout settings = new Callisto.Controls.SettingsFlyout();
    settings.FlyoutWidth =
      Callisto.Controls.SettingsFlyout.SettingsFlyoutWidth.Wide;
    settings.HeaderText = "Feeds";

    var view = new SettingsView();
    settings.Content = view;
    settings.HorizontalContentAlignment = HorizontalAlignment.Stretch;
    settings.VerticalContentAlignment = VerticalAlignment.Stretch;
    settings.IsOpen = true;
  });
  commands.Add(settingsCommand);
}
```

that list. A SettingsCommand object doesn't ask for much, just a unique identifier, label text and code to execute when the user selects the option.

I use my IoC container to inject an instance of ISettingsManager to my view model, then set it up to initialize and clean up, as shown in **Figure 8**.

I'll be using the Settings to allow users to change which RSS feeds they can view with the sample app. This is something I want the user to be able to change from anywhere in the app, so I've included the ShellViewModel, which is instantiated when the app starts up. If I wanted the RSS feeds to be changed only from one of the other views, I'd include the settings code in the associated view model.

Built-in functionality for creating and maintaining a flyout for settings is lacking in the Windows Runtime. There's a lot more manual coding required than there should be to get functionality that's supposed to be consistent across all apps. Luckily, I'm not the only one who feels this way. Tim Heuer, a program manager on the Microsoft XAML team, has created an excellent framework called Callisto, which helps with this pain point. Callisto is available on GitHub (bit.ly/Kijr1S) and on NuGet (bit.ly/112ehch). I use it in the sample app and I recommend checking it out.

Figure 10 **Implementing ISettings**

```
public sealed class Settings : ISettings
{
  public void AddOrUpdate(string key, object value)
  {
    ApplicationData.Current.RoamingSettings.Values[key] = value;
  }

  public bool TryGetValue<T>(string key, out T value)
  {
    var result = false;
    if (ApplicationData.Current.RoamingSettings.Values.ContainsKey(key))
    {
      value = (T)ApplicationData.Current.RoamingSettings.Values[key];
      result = true;
    }
    else
    {
      value = default(T);
    }

    return result;
  }

  public bool Remove(string key)
  {
    return ApplicationData.Current.RoamingSettings.Values.Remove(key);
  }

  public bool ContainsKey(string key)
  {
    return ApplicationData.Current.RoamingSettings.Values.ContainsKey(key);
  }
}
```

Because I have the SettingsManager all wired up in my view model, I just need to provide the code to execute when the settings are requested, as shown in **Figure 9**.

I create a new SettingsCommand, giving it the id "FeedsSetting" and the label text "Feeds." The lambda I use for the callback, which gets called when the user selects the "Feeds" menu item, leverages Callisto's SettingsFlyout control. The SettingsFlyout control does the heavy lifting of where to put the flyout, how wide to make it, and when to open and close it. All I have to do is tell it whether I want the wide or narrow version, give it some header text and the content, then set IsOpen to true to open it up. I also recommend setting the HorizontalContentAlignment and the VerticalContentAlignment to Stretch. Otherwise, your content won't match the size of the SettingsFlyout.

Message Bus

One important point when dealing with the Settings contract is that any changes to settings are expected to be applied to and reflected in the app immediately. There are a number of ways you can accomplish broadcasting the settings changes made by the

Figure 11 **Loading and Saving the User's Feeds**

```
public SettingsViewModel(
  ISettings settings,
  IMessageBus messageBus)
{
  this.settings = settings;
  this.messageBus = messageBus;

  this.Feeds = new ObservableCollection<string>();

  string[] feedData;
  if (this.settings.TryGetValue<string[]>(Constants.FeedsKey, out feedData))
  {
    foreach (var feed in feedData)
    {
      this.Feeds.Add(feed);
    }
  }
}

public void AddFeed()
{
  this.Feeds.Add(this.NewFeed);
  this.NewFeed = string.Empty;

  SaveFeeds();
}

public void RemoveFeed(string feed)
{
  this.Feeds.Remove(feed);

  SaveFeeds();
}

private void SaveFeeds()
{
  this.settings.AddOrUpdate(Constants.FeedsKey, this.Feeds.ToArray());

  this.messageBus.Publish<FeedsChangedMessage>(new FeedsChangedMessage());
}
```

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

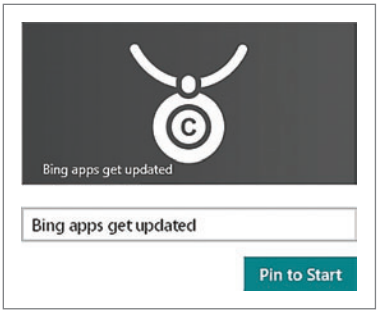


Figure 12 **SecondaryTile Requesting Permission to Pin a Tile to the Start Screen**

user. The method I prefer to use is a message bus (also known as an event aggregator). A message bus is an app-wide message publication system. The concept of the message bus is not built in to the Windows Runtime, which means I have to either create one or use one from another framework. I've included a message bus implementation that I've used on several projects with the Charmed framework. You can find the source at bit.ly/12EBHrb. There are several other good implementations out there as well. Caliburn.Micro has the EventAggregator and MVVM Light has the Messenger. All implementations typically follow the same pattern, providing a way to subscribe to, unsubscribe from and publish messages.

Using the Charmed message bus in the settings scenario, I configure my MainViewModel (the one that displays the feeds) to subscribe to a FeedsChangedMessage:

```
this.messageBus.Subscribe<FeedsChangedMessage>((message) =>
  {
    LoadFeedData();
  });
```

Once MainViewModel is set up to listen for changes to the feeds, I configure SettingsViewModel to publish the FeedsChangedMessage when the user adds or removes an RSS feed:

```
this.messageBus.Publish<FeedsChangedMessage>(new FeedsChangedMessage());
```

Whenever a message bus is involved, it's important that every part of the app uses the same message bus instance. So, I made sure to configure my IoC container to give a singleton instance for every request to resolve an IMessageBus.

Now the sample app is set up to let the user make changes to the RSS feeds displayed via the Settings charm and update the main view to reflect these changes.

Roaming Settings

Another cool thing Windows 8 introduced is the concept of roaming settings. Roaming settings allow app developers to transition small amounts of data among all of a user's devices. This data must be less than 100KB and should be limited to bits of information needed to create a persistent, customized UX across all devices. In the case of the sample app, I want to be able to persist the RSS feeds the user wants to read across all his devices.

The Settings contract I talked about earlier typically goes hand-in-hand with roaming settings. It only makes sense that the customizations I allow the user to make using the Settings contract be persisted across devices with roaming settings.

Gaining access to roaming settings, like the other issues I've looked at so far, is pretty straightforward. The ApplicationData class gives access to both LocalSettings and RoamingSettings. Putting something into RoamingSettings is as simple as providing a key and an object:

```
ApplicationData.Current.RoamingSettings.Values[key] = value;
```

While ApplicationData is easy to work with, it's another sealed class that can't be mocked in unit tests. So, in the interest of keeping my view models as testable as I can, I need to abstract the interaction with ApplicationData. Before defining an interface to abstract the roaming settings functionality behind, I need to decide what I want to do with it:

1. See if a key exists.
2. Add or update a setting.
3. Remove a setting.
4. Get a setting.

Now I have what I need to create an interface I'll call ISettings:

```
public interface ISettings
{
  void AddOrUpdate(string key, object value);
  bool TryGetValue<T>(string key, out T value);
  bool Remove(string key);
  bool ContainsKey(string key);
}
```

With my interface defined, I need to implement it, as **Figure 10** shows.

TryGetValue will first check whether a given key exists and assign the value to the out parameter if it does. Rather than throw an exception if the key isn't found, it returns a bool indicating whether the key was found. The rest of the methods are fairly self-explanatory.

Figure 13 **The TileInfo Helper Class**

```
public sealed class TileInfo
{
  public TileInfo(
    string tileId,
    string shortName,
    string displayName,
    TileOptions tileOptions,
    Uri logoUri,
    string arguments = null)
  {
    this.TileId = tileId;
    this.ShortName = shortName;
    this.DisplayName = displayName;
    this.Arguments = arguments;
    this.TileOptions = tileOptions;
    this.LogoUri = logoUri;
    this.Arguments = arguments;
  }

  public TileInfo(
    string tileId,
    string shortName,
    string displayName,
    TileOptions tileOptions,
    Uri logoUri,
    Uri wideLogoUri,
    string arguments = null)
  {
    this.TileId = tileId;
    this.ShortName = shortName;
    this.DisplayName = displayName;
    this.Arguments = arguments;
    this.TileOptions = tileOptions;
    this.LogoUri = logoUri;
    this.WideLogoUri = wideLogoUri;
    this.Arguments = arguments;
  }

  public string TileId { get; set; }
  public string ShortName { get; set; }
  public string DisplayName { get; set; }
  public string Arguments { get; set; }
  public TileOptions TileOptions { get; set; }
  public Uri LogoUri { get; set; }
  public Uri WideLogoUri { get; set; }
}
```

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.

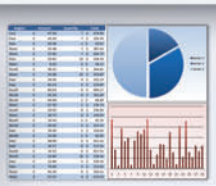
Excel Reporting for ASP.NET, WinForms, WPF and Silverlight

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 14 **Implementing ISecondaryPinner**

```
public sealed class SecondaryPinner : ISecondaryPinner
{
  public async Task<bool> Pin(
    FrameworkElement anchorElement,
    Placement requestPlacement,
    TileInfo tileInfo)
  {
    if (anchorElement == null)
    {
      throw new ArgumentNullException("anchorElement");
    }
    if (tileInfo == null)
    {
      throw new ArgumentNullException("tileInfo");
    }

    var isPinned = false;

    if (!SecondaryTile.Exists(tileInfo.TileId))
    {
      var secondaryTile = new SecondaryTile(
        tileInfo.TileId,
        tileInfo.ShortName,
        tileInfo.DisplayName,
        tileInfo.Arguments,
        tileInfo.TileOptions,
        tileInfo.LogoUri);

      if (tileInfo.WideLogoUri != null)
      {
        secondaryTile.WideLogo = tileInfo.WideLogoUri;
      }

      isPinned = await secondaryTile.RequestCreateForSelectionAsync(
        GetElementRect(anchorElement), requestPlacement);
    }

    return isPinned;
  }

  public async Task<bool> Unpin(
    FrameworkElement anchorElement,
    Placement requestPlacement,
    string tileId)
  {
    var wasUnpinned = false;

    if (SecondaryTile.Exists(tileId))
    {
      var secondaryTile = new SecondaryTile(tileId);
      wasUnpinned = await secondaryTile.RequestDeleteForSelectionAsync(
        GetElementRect(anchorElement), requestPlacement);
    }

    return wasUnpinned;
  }

  public bool IsPinned(string tileId)
  {
    return SecondaryTile.Exists(tileId);
  }

  private static Rect GetElementRect(FrameworkElement element)
  {
    GeneralTransform buttonTransform =
      element.TransformToVisual(null);
    Point point = buttonTransform.TransformPoint(new Point());
    return new Rect(point, new Size(
      element.ActualWidth, element.ActualHeight));
  }
}
```

Now I can let my IoC container resolve ISettings and give it to my SettingsViewModel. Once I do, the view model will use the settings to load the user's feeds to be edited, as shown in **Figure 11**.

One thing to note about the code in **Figure 11** is that the data I actually save in the settings is a string array. Because roaming settings are limited to 100KB, I need to keep things simple and stick to primitive types.

Secondary Tiles

Developing apps that engage users can be enough of a challenge. But how do you keep users coming back once they install your app? One thing that can help with this challenge is secondary tiles. A secondary tile provides the ability to deep link into an application, letting the user bypass the rest of the app and go straight to what he cares about most. A secondary tile gets pinned to the user's home screen, with an icon of your choosing. Once tapped, the secondary tile launches your app with arguments that tell the app exactly where to go and what to load. Providing secondary tile functionality to your users is a good way to let them customize their experience, making them want to come back for more.

Secondary tiles are more complicated than the other topics I cover in this article, because there are several things that have to be implemented before the full experience of using secondary tiles will work correctly.

Pinning a secondary tile involves instantiating the SecondaryTile class. The SecondaryTile constructor takes several parameters that help it determine what the tile will look like, including a display name, a URI to the logo image file to use for the tile, and string arguments that will be given to the app when the tile is pressed. Once the SecondaryTile has been instantiated, I must call a method that will ultimately show a little pop-up window asking the user for permission to pin the tile, as shown in **Figure 12**.

Once the user has pressed Pin to Start, the first half of the work is done. The second half is configuring the app to actually support the deep linking by using the arguments the tile provides when it's

Figure 15 **Pinning and Unpinning with ISecondaryPinner**

```
public FeedItemViewModel(
  IShareManager shareManager,
  ISecondaryPinner secondaryPinner)
{
  this.shareManager = shareManager;
  this.secondaryPinner = secondaryPinner;
}

public async Task Pin(FrameworkElement anchorElement)
{
  var tileInfo = new TileInfo(
    FormatSecondaryTileId(),
    this.FeedItem.Title,
    this.FeedItem.Title,
    TileOptions.ShowNameOnLogo | TileOptions.ShowNameOnWideLogo,
    new Uri("ms-appx:///Assets/Logo.png"),
    new Uri("ms-appx:///Assets/WideLogo.png"),
    this.FeedItem.Id.ToString());

    this.IsFeedItemPinned = await this.secondaryPinner.Pin(
    anchorElement,
    Windows.UI.Popups.Placement.Above,
    tileInfo);
}

public async Task Unpin(FrameworkElement anchorElement)
{
  this.IsFeedItemPinned = !await this.secondaryPinner.Unpin(
    anchorElement,
    Windows.UI.Popups.Placement.Above,
    this.FormatSecondaryTileId());
}
```

Work With Relational Data, Not Complex APIs or Services

Whether you are a developer using ADO.NET, JDBC, OData, or MySQL, or a systems integrator working with SQL Server or Biztalk, or even an information worker familiar with ODBC or Excel – our products give you bi-directional access to live data through easy-to-use technologies that you are already familiar with. If you can connect to a database, then you will already know how to connect to Salesforce, SAP, SharePoint, Dynamics CRM, Google Apps, QuickBooks, and much more!

Give RSSBus a try today and see what mean:

visit us online at www.rssbus.com to learn more or download a free trial.

rssbus

INTEGRATION YOUR WAY

pressed. Before I get into the second half, let me talk about how I'll implement the first half in a testable way.

Because SecondaryTile uses methods that interact directly with the OS—which, in turn, shows UI components—I can't use it directly from my view models without sacrificing testability. So, I'll abstract out another interface, which I'll call ISecondaryPinner (it should allow me to pin and unpin a tile, and check if a tile has already been pinned):

```
public interface ISecondaryPinner
{
  Task<bool> Pin(FrameworkElement anchorElement,
    Placement requestPlacement, TileInfo tileInfo);
  Task<bool> Unpin(FrameworkElement anchorElement,
    Placement requestPlacement, string tileId);
  bool IsPinned(string tileId);
}
```

Notice that both Pin and Unpin return Task<bool>. That's because the SecondaryTile uses async tasks to prompt the user to pin or unpin a tile. It also means that my ISecondaryPinner Pin and Unpin methods can be awaited.

Also notice that both Pin and Unpin take a FrameworkElement and a Placement enumeration value as parameters. The reason is that the SecondaryTile needs a rectangle and a Placement to tell it where to put the pin request pop-up. I plan to have my implementation of SecondaryPinner calculate that rectangle based on the FrameworkElement that's passed in.

Finally, I create a helper class, TileInfo, to pass around the required and optional parameters used by SecondaryTile, as shown in **Figure 13**.

TileInfo has two constructors that can be used, depending on the data. Now, I implement ISecondaryPinner, as shown in **Figure 14**.

Pin will first make sure the requested tile doesn't already exist, then it will prompt the user to pin it. Unpin will first make sure the requested tile does exist, then it will prompt the user to unpin it. Both will return a bool indicating whether the pin or unpin was successful.

Now I can inject an instance of ISecondaryPinner into my view model and put it to use, as shown in **Figure 15**.

In Pin, I create a TileInfo helper instance, giving it a uniquely formatted id, the feed title, URIs to the logo and wide logo, and the feed id as the launch argument. Pin takes the button that was clicked as the anchor element to base the location of the pin request pop-up. I use the result of the SecondaryPinner.Pin method to determine whether the feed item has been pinned.

In Unpin, I give the uniquely formatted id of the tile, using the inverse of the result to determine whether the feed item is still pinned. Again, the button that was clicked is passed to Unpin as the anchor element for the unpin request pop-up.

After I have this in place and I use it to pin a blog post (FeedItem) to the Start screen, I can tap the newly created tile to launch the app. However, it will launch the app in the same way as before, taking me to the main page, displaying all blog posts. I want it to take me to the specific blog post that I pinned. That's where the second half of the functionality comes into play.

The second half of the functionality goes into app.xaml.cs, from which the app is launched, as shown in **Figure 16**.

I add some code to the end of the overridden OnLaunched method to check whether arguments have been passed in during the launch. If arguments have been passed, I parse the arguments into an int to be used as the feed id. I get the feed with that id from my saved feeds and pass it to FeedItemViewModel to be displayed. One thing to note is that I make sure the app already has the main page displayed, and I navigate to it first if it hasn't been displayed. That way the user can press the back button and land on the main page whether or not he was already running the app.

Figure 16 **Launching the App**

```
protected override async void OnLaunched(LaunchActivatedEventArgs args)
{
  Frame rootFrame = Window.Current.Content as Frame;
  if (rootFrame.Content == null)
  {
    Ioc.Container.Resolve<INavigator>().
      NavigateToViewModel<MainViewModel>();
  }

  if (!string.IsNullOrWhiteSpace(args.Arguments))
  {
    var storage = Ioc.Container.Resolve<IStorage>();
    List<FeedItem> pinnedFeedItems =
      await storage.LoadAsync<List<FeedItem>>(Constants.PinnedFeedItemsKey);
    if (pinnedFeedItems != null)
    {
      int id;
      if (int.TryParse(args.Arguments, out id))
      {
        var pinnedFeedItem = pinnedFeedItems.FirstOrDefault(fi => fi.Id == id);
        if (pinnedFeedItem != null)
        {

          Ioc.Container.Resolve<INavigator>().
            NavigateToViewModel<FeedItemViewModel>(
            pinnedFeedItem);
        }
      }
    }
  }

  Window.Current.Activate();
}
```

Wrapping Up

In this article, I talked about my approach to implementing a testable Windows Store app using the MVVM pattern, while still leveraging some of the cool new features that Windows 8 brings to the table. Specifically, I looked at abstracting sharing, settings, roaming settings and secondary tiles into helper classes that implement mockable interfaces. Using this technique, I'm able to unit test as much of my view model functionality as possible.

In future articles, I'll dive into more of the specifics of how I can actually write unit tests for these view models now that I've set them up to be more testable. I'll also explore how these same techniques can be applied to make my view models cross-platform with Windows Phone 8, while keeping them testable.

With a little planning, you can create a compelling application with an innovative UX that leverages new key features of Windows 8—and do so without sacrificing best practices or unit tests. ■

Brent Edwards *is an associate principal consultant for Magenic, a custom application development firm that focuses on the Microsoft stack and mobile application development. He's also a cofounder of the Twin Cities Windows 8 User Group in Minneapolis, Minn. Reach him at brente@magenic.com.*

THANKS to the following technical expert for reviewing this article: *Rocky Lhotka (Magenic)*

Creating a Windows Phone 8 Company Hub App

Tony Champion

Prior to the release of Windows Phone 8, companies had only a couple of options for deploying enterprise apps to their employees' devices. They could release the apps to the Windows Phone Store and require user authentication, allowing the apps to be successfully deployed while securing their usage. Of course, the app would then be available in the store and available to the general public for download. The other approach was to use a mobile device management (MDM) solution to directly manage the devices. Microsoft currently offers two MDMs: Windows Intune and System Center 2012 Configuration Manager. While MDMs are still the preferred approach for many large-scale enterprises, they might not align with a company's deployment goals.

Windows Phone 8 brings a new, unmanaged option for deploying enterprise apps that allows devices to install apps directly from an e-mail attachment or by downloading the app from a URL. This direct approach to deploying enterprise apps opens enterprise app development and deployment to more companies.

A drawback to this unmanaged approach is that installation and app management for the user is not altogether intuitive. When it comes to mobile devices, users expect to be able to install apps with the push of a button. Having to manually install apps from an e-mail or a Web site goes against the normal app experience and can cause uncertainty in users and training issues for the IT staff. Luckily, there's a solution—a company hub app.

Windows Phone 8 includes some additions to the SDK that support this new concept. A company hub app gives the enterprise a mechanism for presenting and delivering apps to the user for installation. And it gives users a familiar experience to discover and install company apps.

GET HELP BUILDING YOUR WINDOWS PHONE APP

Receive the tools, help and support you need to develop your Windows Phone 8 app.

bit.ly/12hQbO0

This article discusses:

- Options for deploying enterprise apps to employee devices
- Requirements for company hub apps
- Getting necessary information from the manifest file
- Classes in the Windows Phone SDK 8.0 for managing company apps
- The company hub app demo

Technologies discussed:

Windows Phone 8, Visual Studio

Code download available at:

archive.msdn.microsoft.com/mag201307Hub

Requirements for Company Hub Apps

Before you can publish a company hub app to your employees, there are a few administrative steps that need to be taken. The first step is to register for a company account on the Windows Phone Dev Center. The registration process is similar to a regular development account, but a company account goes through some additional account verification.

After creating an account, you must purchase an Enterprise Mobile Code Signing Certificate from Symantec Corp. The certificate-purchasing process requires a valid Symantec ID from the Dev Center, so this can only be completed after the company account has been established and verified. Once the certificate is purchased and installed on a machine, you'll need to export the certificate in a PFX format that includes the private key. This certificate will be used for generating an application enrollment token (AET) and for signing any apps developed by the company. When installing the certificate on your development machine, it's important to follow the steps outlined at bit.ly/1287H8j. Otherwise, you'll end up with an incomplete PFX file that won't validate correctly during deployment. Skipping these steps is a common headache for many developers.

An AET must be installed on a device before that device can install any apps developed by a company. By installing the AET, the device is enrolled in the previously established company account. The Windows Phone SDK 8.0 includes a tool, AETGenerator, that can be used to create the AET from the exported certificate. The generator creates three different forms of the AET: a raw version containing the AET in XML format (.xml), a Base64-encoded version used with an MDM such as Windows Intune or System Center 2012 Configuration Manager (.aet), and an XML format that can be installed directly on the device through e-mail or Internet Explorer (.aetx).

An AET must be installed on a device before that device can install any apps developed by a company.

When deciding which method to use to distribute the AET and your company apps, there's an important point to consider. When a device installs an AET, it's valid until its expiration date, which by default is one year. Once the AET expires, the device won't be able to run any apps signed and distributed by the company—including hub apps—until a new, valid AET is installed. This creates two items you need to add to your deployment strategy planning.

The first is how to handle the expiration of the AET. If you're using an MDM to manage your devices, an updated AET can be published to the devices directly from the MDM. This will help to minimize the impact of the expiration of the original AET on the devices. If the AET is installed via an unmanaged process—e-mail or Internet Explorer—the new AET will need to be installed manually by the user. Because the user won't be able to run any apps from the company, including the hub app, once the AET expires, it's a good practice to create and distribute a new AET before the original expires. This will prevent the user from losing access to company apps.

The second item to consider is the removal of the AET as well as any company apps installed on the device. In the current Bring Your Own Device (BYOD) world, this can be a major consideration. Using an MDM gives the company complete control over its apps installed on a device. Apps and the AET can be remotely removed directly from the MDM. However, in an unmanaged deployment, this isn't possible. Once an AET is added to the device, it's valid and can't be removed prior to its expiration. Similarly, the SDK doesn't provide a way to remove an app from a device through code. The best practice for addressing this issue is requiring users to authenticate to all company apps. This enables you to prevent a user's account from launching an app. While this isn't the same as being able to remove the app, it does give you a way to manage the security of your app once it's deployed to a device.

Figure 1 **The Company Hub App**

After you've completed the initial steps, you're ready to begin creating apps that can be deployed to your company's employees without needing to go through the store. You'll find a more complete look at creating a company account, obtaining an Enterprise Mobile Code Signing Certificate and generating an AET at bit.ly/SBN6Tf.

Preparing for Development

A company hub app requires a bit more setup than most Windows Phone apps. For one thing, it's quite handy to have at least a few apps available to install and use as test cases within the app. While the apps themselves can be blank Windows Phone apps, they all have to have one thing in common: a Publisher ID.

If you've already created a Windows Phone app, you're probably familiar with the Windows Phone app manifest file, which by default is the WMAppManifest.xml file located in the Properties folder of the solution. This file contains information Windows Phone and the store need to know about your app. Opening the WMAppManifest.xml file from within Visual Studio launches a designer to make it easier to maintain the file. The designer contains four tabs, one of which is the Packaging tab.

The Packaging tab provides information about the developer of the app, versioning info and supported languages. For the purposes of this article, the Version, Product ID and Publisher ID are the most important items on this tab. By default, the Windows Phone project templates generate a new GUID for the Product ID and Publisher ID when the project is created. When working with an app such as a company hub, the app will have visibility only into other apps that have the same Publisher ID it does. This Publisher ID is commonly set to the Publisher GUID that's assigned to your developer account in the Dev Center. The Publisher GUID can be found on your Account Summary page in the Dev Center.

The downloadable file for this article contains seven solutions. The first six are named CDSAPP[1-6]. Each of these apps was created from

the Windows Phone App project template and only has the app title on the main page and the Publisher ID modified. If you're going to use these apps for your testing, it's important to use the same Publisher ID for your company hub app or to change the app IDs to yours.

The Version and Product ID are two important pieces of information to know when creating a company hub solution. The Version allows you to determine when apps need to be upgraded on a device, and the Product ID is used for identifying the app.

The next thing to consider is your method of testing. Generally, you can do most of your testing on the Windows Phone Emulator. It does a great job and allows you to test most of the things you need to test during the development process. The difficulty with testing a company hub app is that your app needs to have additional apps installed on the emulator to test with. However, each time you launch a new instance of the emulator, it starts from a clean version of the OS. This means that anything you've previously installed for testing is no longer there.

There are two ways to approach this problem. The first is to leave the emulator running, install some test apps, and then do development of your company hub app while the emulator is still running. Of course, if the emulator gets restarted for any reason, you have to reinstall those apps to begin testing again.

The preferred approach is to do your development testing against a real device. This provides a more stable platform for your testing. If you're going to do your testing on a live device, you'll need to make sure the Publisher ID of that device corresponds to the Publisher ID of your account.

Figure 2 **The CompanyPackage View Model**

```
public class CompanyPackageViewModel : INotifyPropertyChanged
{
  private IEnumerable<CompanyPackage> _packages;

  public CompanyPackageViewModel()
  {
    LoadData();
  }

  private void LoadData()
  {
    // Get list of packages and populate properties
    _packages = CompanyPackage.GenerateData();

    UpdatePackageStatus();

  }

  private IEnumerable<CompanyPackage> _newPackages;
  public IEnumerable<CompanyPackage> NewPackages
  {
    get
    {
      return _newPackages;
    }
  }

  private IEnumerable<CompanyPackage> _updatePackages;
  public IEnumerable<CompanyPackage> UpdatePackages
  {
    get
    {
      return _updatePackages;
    }
  }

  private IEnumerable<CompanyPackage> _installedPackages;
  public IEnumerable<CompanyPackage> InstalledPackages
  {
    get
    {
      return _installedPackages;
    }
  }

  public event PropertyChangedEventHandler PropertyChanged;
  private void NotifyPropertyChanged(String propertyName)
  {
    PropertyChangedEventHandler handler = PropertyChanged;
    if (null != handler)
    {
      handler(this, new PropertyChangedEventArgs(propertyName));
    }
  }
}
```

The Company Hub SDK

The Windows Phone SDK 8.0 includes two classes that are primarily responsible for managing and interacting with the company apps that are installed on the machine. While there are some supporting objects that you'll be introduced to along the way, understanding these two classes is critical to your hub app.

The Package Class Each app installed on the device is represented by the Package class, which is located in the Windows.ApplicationModel namespace. The Package class contains a couple of important members that I'll discuss here. The first is the Id property, which returns a PackageId class. The class contains most of the manifest information that was entered into the Packaging tab in the manifest designer, including the Product Id that was assigned to the app, the app name, publisher information and the current version.

The other important member of the Package class is the Launch method, which enables you to launch the app that the Package represents from the current app. Not only is this a great tool for constructing a company hub app, it can also be useful for other line-of-business (LOB) apps.

InstallationManager Class InstallationManager, in the Windows.Phone.Management.Deployment namespace, is the class responsible for installing packages on the device. This is accomplished through the AddPackageAsync method. A list of all apps from the same PublisherId as the current app, including the Package representing the current app, is returned by the FindPackagesFromCurrentPublisher method. In addition, the class can also be used to get the installation progress of any apps with the same PublisherId.

Building a Company Hub App

I'm going to introduce you to the core principles of developing a company hub app, using the demo shown in **Figure 1**. The demo contains a three-page panorama app that lists available company apps in three categories: apps that are currently not installed, apps with an available update and apps that have the latest version installed. In addition, the app will have a detail page that presents the information about the app and provides the commands to install the latest version and launch the app from within the hub. The working solution can be found as a downloadable resource from archive.msdn.microsoft.com/mag201307Hub.

What's Missing For the sake of completeness, it's important to point out a few items that aren't included in the demo app but would have to be created for any real-world solution. The first is the source of the available company apps. If you examine the

Figure 3 **Method for Determining the Status of Each CompanyPackage**

```
public void UpdatePackageStatus()
{
  var devicePkgs = InstallationManager.FindPackagesForCurrentPublisher();

  foreach (var pkg in _packages)
  {
    var qry = devicePkgs.Where(p => p.Id.ProductId == pkg.Id);
    if (qry.Count() > 0)
    {
      var devicePkg = qry.First();
      var devicePkgVersion =
        PackageVersionHelper.VersionToString(devicePkg.Id.Version);

      pkg.Status = PackageVersionHelper.IsNewer(
        pkg.Version, devicePkgVersion) ?
        CompanyPackageStatus.Update : CompanyPackageStatus.Installed;
    }
    else
    {
      pkg.Status = CompanyPackageStatus.New;
    }

  }

  _newPackages = _packages.Where(
     p => p.Status == CompanyPackageStatus.New).ToList();
  _updatePackages = _packages.Where(
     p => p.Status == CompanyPackageStatus.Update).ToList();
  _installedPackages = _packages.Where(
     p => p.Status == CompanyPackageStatus.Installed).ToList();

  // Fire notifications for all properties
  NotifyPropertyChanged("NewPackages");
  NotifyPropertyChanged("UpdatePackages");
  NotifyPropertyChanged("InstalledPackages");

}
```

CompanyPackage class, you'll see it contains a GenerateData method. This method is used to fake the available company apps by generating a list of the installed company apps on the device, modifying some of the data and creating some fictional data, as well.

The second missing piece is a Web site to host the .xap files to be downloaded and installed on the device. For the company hub to be functional, it must be able to download the apps from some location. This Web solution would have to be created as well.

The CompanyPackage Class In order to represent the list of available apps to install on the device, the first thing to define is a CompanyPackage class that's modeled after the Package class. The CompanyPackage class contains the display information and the location of the installation package:

```
public class CompanyPackage
{
  public string Id { get; set; }
  public string Name { get; set; }
  public string Description { get; set; }
  public string Thumbnail { get; set; }
  public string Version { get; set; }
  public Uri SourceUri { get; set; }
  public CompanyPackageStatus Status { get; set; }
}
```

The Status property is used to determine if the app is currently installed on the device and if there's a newer version. CompanyPackageStatus is an enum that I'll explore later in the article:

```
public enum CompanyPackageStatus
{
  Unknown,
  New,
  Update,
  Installed
};
```

Creating a View Model For this company hub app, you have to create a single view model, CompanyPackageViewModel, shown in **Figure 2**. The view model should have three properties that return an IEnumerable collection of CompanyPackage objects: NewPackage, UpdatePackages and InstalledPackages. The NewPackage property contains any apps that are available that aren't currently installed on the machine. UpdatePackages represents any apps that are currently installed on the machine, but have a newer version available. Finally, InstalledPackages contains all of the apps that are currently installed and up-to-date on the device.

Defining the Three Statuses Before you can populate the three different properties of the view model, you need to generate the list of available apps represented as a collection of CompanyPackages. In the view model shown in **Figure 2**, the data is loaded and populated in the LoadData method. The LoadData method pulls a list of test data and stores it in the private _packages variable. It then calls the UpdatePackageStatus method shown in **Figure 3**.

The UpdatePackageStatus method has two responsibilities: determine the current status of each available CompanyPackage class and then populate the three collection properties of the view model based on that status.

The Status property is determined by comparing each CompanyPackage against the apps that are currently installed on the device. A list of installed apps is obtained from the InstallationManager.FindPackagesForCurrentPublisher static method. If a Package object with the same Id as the CompanyPackage doesn't exist, it's marked with a "New" status.

If a Package with the same Id does exist, then the Package Version property is compared to the Version of the CompanyPackage. The PackageId Version property returns a PackageVersion struct. Unlike the System.Version class, this struct is missing a couple of features. The first feature is the ability to convert the struct to a string representation. If you call the ToString method, it returns the type name and not the actual version number. The second missing feature is the ability to compare two instances of PackageVersion to determine which one is newer.

Figure 4 shows a helper class that implements both of these missing features. The VersionToString method returns a proper string

Figure 4 **A Helper Class for PackageVersion**

```
public static class PackageVersionHelper
{
  public static string VersionToString(PackageVersion version)
  {
    return String.Format("{0}.{1}.{2}.{3}",
                         version.Major,
                         version.Minor,
                         version.Build,
                         version.Revision);
  }

  public static bool IsNewer(string newVersion, string oldVersion)
  {
    var newVer = Version.Parse(newVersion);
    var oldVer = Version.Parse(oldVersion);

    return newVer.CompareTo(oldVer) > 0;
  }
}
```

Figure 5 **Panorama to Display the Available Apps**

```
<phone:Panorama Title="my company hub">
  <phone:Panorama.Resources>
    <DataTemplate x:Key="listItemTemplate">
      <StackPanel Margin="0,-6,0,12" Orientation="Horizontal">
        <Image  Source="{Binding Thumbnail,
          Converter={StaticResource debugConv}}"/>
        <TextBlock Text="{Binding Name}" TextWrapping="Wrap"
          VerticalAlignment="Center"
          Style="{StaticResource PhoneTextExtraLargeStyle}"
          FontSize="{StaticResource PhoneFontSizeExtraLarge}"/>
      </StackPanel>
    </DataTemplate>
  </phone:Panorama.Resources>

  <!--Panorama New Apps-->
  <phone:PanoramaItem Header="New Apps">
    <!--Single line list with text wrapping-->
    <phone:LongListSelector Margin="0,0,-22,0"
      ItemsSource="{Binding NewPackages}"
      SelectionChanged="ItemSelected"
      ItemTemplate="{StaticResource listItemTemplate}"/>
  </phone:PanoramaItem>

  <!--Panorama Update Apps-->
  <phone:PanoramaItem Header="Update Apps">
    <!--Single line list with text wrapping-->
    <phone:LongListSelector Margin="0,0,-22,0"
      ItemsSource="{Binding UpdatePackages}"
      SelectionChanged="ItemSelected"
      temTemplate="{StaticResource listItemTemplate}"/>
  </phone:PanoramaItem>

  <!--Panorama Installed Apps-->
  <phone:PanoramaItem Header="Installed Apps">
    <!--Single line list with text wrapping-->
    <phone:LongListSelector Margin="0,0,-22,0"
      ItemsSource="{Binding InstalledPackages}"
      SelectionChanged="ItemSelected"
      ItemTemplate="{StaticResource listItemTemplate}"/>
  </phone:PanoramaItem>
</phone:Panorama>
```

representation of a PackageVersion. The IsNewer method takes two string representations of a version number and determines if the newVersion argument is newer than the oldVersion argument. It accomplishes this by converting the strings to System.Version objects and using the available CompareTo method.

Once the UpdatePackageStatus method has calculated the Status property for each CompanyPackage object, it populates the three collection properties using LINQ queries. Finally, the view model raises the PropertyChanged event for each of the properties.

Displaying the List of Apps

The three lists of available apps are displayed within a Panorama control with three PanoramaItems, each containing a LongListSelector bound to one of the lists. Each uses the same DataTemplate for displaying a CompanyPackage, and the complete Panorama XAML can be found in **Figure 5**. In the downloadable project, you'll see that the DataContext of the Panorama control inherits the DataContext of the parent PhoneApplicationPage, which is set to an instance of the CompanyPackageViewModel. You saw the result of this in **Figure 1**.

The CompanyPackage Detail View

Each LongListSelector shares the same event handler for the SelectionChanged event, ItemSelected. The event handler uses the NavigationService to navigate to a detail PhoneApplicationPage, PackagePage, and passes in the Id of the CompanyPackage. Because the current page will be cached in the navigation, the SelectedItem of the LongListSelector is reset to null to insure proper event firing each time:

```
private void ItemSelected(object sender, SelectionChangedEventArgs e)
{
  if (e.AddedItems.Count > 0 && e.AddedItems[0] != null)
  {
    var pkg = e.AddedItems[0] as CompanyPackage;
    NavigationService.Navigate(
      new Uri("/PackagePage.xaml?id=" + pkg.Id, UriKind.Relative));

    (sender as LongListSelector).SelectedItem = null;

  }
}
```

Because the PackagePage class only receives the Id of the CompanyPackage to be displayed, it has to use that Id to find the appropriate object. This is done by adding a FindPackage method to the CompanyPackageViewModel:

```
public CompanyPackage FindPackage(string id)
{
  return _packages.Where(p => p.Id == id).FirstOrDefault();
}
```

The Windows Phone Panorama App project exposes a global view model by adding a ViewModel property to the App class. This project uses that property to expose its view model to the main and detail pages.

The PackagePage class overrides the OnNavigatedTo method to set its DataContext to the CompanyPackage matching the provided Id. It then calls an UpdateUI method that toggles the Visibility and Content of two buttons that are added to the screen based on the Status of the CompanyPackage. The results of each Status type can be seen in **Figure 6**.

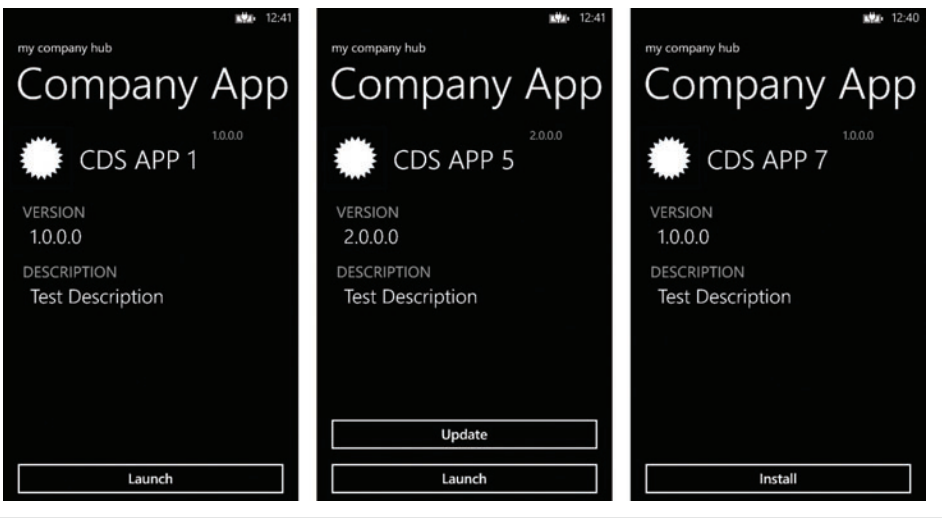


Figure 6 **The Different Detail Pages**

Figure 7 **Install and Update Button Event Handler**

```
private async void btnInstall_Click(object sender, RoutedEventArgs e)
{
  var pkg = DataContext as CompanyPackage;
  if (MessageBox.Show("Install " + pkg.Name + "?", "Install app",
    MessageBoxButton.OKCancel) == MessageBoxResult.OK)
  {
    try
    {
      var result =
        await InstallationManager.AddPackageAsync(pkg.Name, pkg.SourceUri);
      if (result.InstallState ==
        Windows.Management.Deployment.PackageInstallState.Installed)
      {
        MessageBox.Show(pkg.Name + " was installed.");
        App.ViewModel.UpdatePackageStatus();
        UpdateUI();
      }
      else
      {
        MessageBox.Show("An error occurred during installation.");
      }
    }
    catch (Exception)
    {
      MessageBox.Show("An error occurred during installation.");
    }
  }
}
```

The two buttons expose the two actions available within the company hub classes. The first is the ability to launch the app. If the app is currently installed on the device, the Launch button is visible. The event handler of the button finds the correct Package object that matches the current CompanyPackage and calls the Launch method:

```
private void btnLaunch_Click(
  object sender, RoutedEventArgs e)
{
  var pkg = DataContext as CompanyPackage;

  var devicePkgs = InstallationManager.
    FindPackagesForCurrentPublisher();

  var devicePkg = devicePkgs.Where(p =>
    p.Id.ProductId == pkg.Id).FirstOrDefault();

  if (devicePkg != null)
  {
    devicePkg.Launch("");
  }
}
```

If the CompanyPackage Status is "New" or "Update," an Install button is visible on the page. The event handler for this button attempts to install the latest version of the app from the Uri provided in the SourceUri property of the CompanyPackage. This is accomplished with the InstallationManager.AddPackageAsync method. The same method is called whether the app is being updated or it's a new install. This method can be very temperamental and you need to make sure to take care of any errors that are generated. **Figure 7** shows the event handler and the installation process. If the app is installed successfully, the UpdatePackageStatus method of the CompanyPackageViewModel is called to update the status collections being displayed in the main page and the UpdateUI method of the page to update the page.

Next Steps

In this article I took a quick look at developing a company hub app while exposing the details necessary to create a more robust solution. Some of the details that were omitted for the sake of brevity can be found in the included download. However, it's important to remember that this is just the beginning.

There are a lot of features that can be added to a company hub to provide great benefits to your users, such as taking advantage of live tiles to inform the user when new apps are available. You can create solutions that only expose certain apps to certain users based on their roles within the company. An app can provide additional functionality, such as company news and notifications. While the possibilities might not be limitless, there are more than enough to keep you busy for quite some time. ■

TONY CHAMPION *is president of Champion DS, is a Microsoft MVP, and active in the community as a speaker, blogger, and author. He maintains a blog at tonychampion.net and can be reached via e-mail at tony@tonychampion.net.*

THANKS to the following technical expert for reviewing this article: *Cliff Strom (Microsoft)*

ORLANDO • NOVEMBER • 18-22, 2013

LIVE EVENT

SharePoint LIVE!

TRAINING FOR COLLABORATION

LIVE!

COLLABORATE AND LISTEN

No-hype, Practical, Independent SharePoint Training

Whether you've implemented SharePoint 2013, or you're still using and supporting older versions, SharePoint Live! is THE place to gather and learn how to customize, deploy and maintain SharePoint Server and SharePoint foundation to maximize business value.

SUPPORTED BY

Microsoft SharePoint Visual Studio msdn magazine Visual Studio MAGAZINE

PRODUCED BY

1105 MEDIA

WHETHER YOU ARE AN:

- IT Manager
- Administrator
- IT Pro
- Developer

SharePoint Live! brings together the best the industry has to offer for 5 days of workshops, keynotes, and sessions to help you work through your most pressing SharePoint projects.

SUMMER SPECIAL! SAVE $500!

REGISTER BEFORE AUGUST 7

Use Promo Code SPJUL2

CONNECT WITH SHAREPOINT LIVE!

twitter.com/SPlive360 – @SPLIVE360

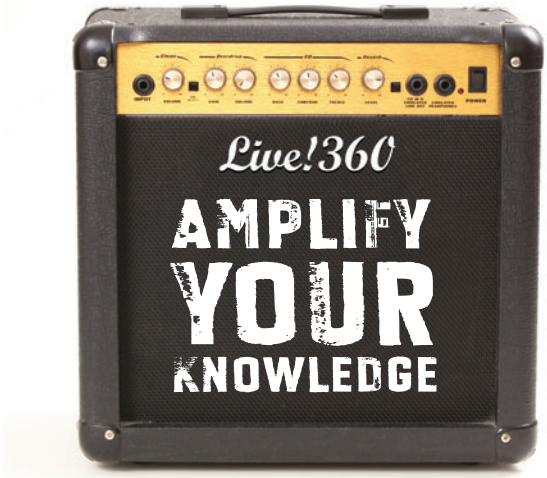
facebook.com – Search "SharePoint Live"

linkedin.com – Join the "SharePoint Live" group!

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

ORLANDO | NOVEMBER 18-22, 2013

Royal Pacific Resort at Universal Orlando



LIVE! 360

IT EVENTS WITH PERSPECTIVE

SharePoint Live! is part of Live! 360, the ultimate IT and Developer line-up. This means you'll have access to four (4) events, 22 tracks, and hundreds of sessions to choose from – mix and match sessions to create your own, custom event line-up – it's like no other conference available today!

SharePoint LIVE! TRAINING FOR COLLABORATION

SQL Server LIVE! TRAINING FOR DBAs AND IT PROS

Visual Studio LIVE! EXPERT SOLUTIONS FOR .NET DEVELOPERS

Modern Apps LIVE! MODERN APPS FROM START TO FINISH

splive360.com | live360events.com

Building Apps for Windows 8 and Windows Phone 8

Joel Reyes

Windows 8 and Windows Phone 8 are on a path toward convergence. In the meantime, developers interested in building for both platforms must understand the major similarities and differences between the two. Learning where the Windows 8 and Windows Phone Runtime APIs currently intersect gives you the best opportunity to deliver applications for both, leveraging much of the same knowledge, tools, code and assets. In this article I'll explore these differences and commonalties to help you understand what is and is not possible before you start building a solution that targets both platforms.

The consistency in the UX—the use of tiles, the rich touch interface, the app bar and navigation—simplifies application design and implementation for both platforms. And the adoption of a common API surface area facilitates code sharing for a lot of scenarios.

This article discusses:

- UX considerations
- Features for which code reuse shouldn't be attempted
- Platform-specific features

Technologies discussed:

Windows 8, Windows Phone 8

You can choose the right technologies for your apps: C#, Visual Basic or C++, or a hybrid for both platforms. The resources found at `aka.ms/sharecode` contain great info about creating applications that run on Windows Phone 8 and Windows 8, and present code-sharing techniques to maximize code reuse when building for both.

Specific Areas of Comparison

In order to effectively write for both platforms, you need to understand the major feature differences, as well as the features that at a high level appear similar, but have different APIs and implementations. For those features, code reuse should not be attempted. I'll examine three key areas of concern:

- UX
- Data model and supporting code
- Platform-specific features

UX

Form factors are a major consideration when designing the UI for a cross-platform app. You need to consider attributes such as screen resolutions, screen sizes and default device orientation. Careful UI decisions must be made to ensure that the experience is optimized for the user given the constraints of the particular device.

You'll quickly realize that it doesn't make sense to include all features of a Windows Store app in a Windows Phone 8 app and that you must design a native UX for each platform. **Figure 1** shows examples of the kind of UX differences you'll have to consider to build the best experience possible.

XAML Namespaces Even though both Windows 8 and Windows Phone 8 use XAML for UI design, don't expect to reuse XAML across your apps. There are major differences in the platforms—page layout and orientation, XAML namespaces and XAML controls—that make this challenging.

Although many controls exist for both platforms, they reside in different namespaces. For instance, Windows 8 controls equivalents for Windows Phone can be found in Windows.UIXaml.Controls, as well as in the Microsoft.Phone.Controls and Microsoft.Phone.Shell, while System.Windows.Controls contains Windows 8 controls and some shared controls. Luckily, the XAML editor will alert you when you attempt to add an unsupported control. However, XAML doesn't support conditional compilation, so there's no easy way to include namespaces for one platform or the other at run time.

XAML Controls Each library of controls is optimized for its corresponding platform. Consequently, it's highly recommended that you design your own user controls separately for each platform. This will help maintain a consistent UX on each platform while reducing the likelihood of bugs from porting the XAML. **Figure 2** shows some basic controls.

The Windows 8 Visual Studio templates create a LayoutAwarePage, which is a subclass of the Page in the raw API, providing the following additional features and ensuring consistency of design and UX:

- Application view state to visual state mapping
- GoBack, GoForward and GoHome event handlers
- Mouse and keyboard shortcuts for navigation
- State management for navigation and process lifetime management
- A default view model

Figure 1 **A Comparison of Windows Phone 8 and Windows 8 UX Features**

Windows Phone 8	Windows 8
One-handed touch most common	One or two-handed touch, mouse
Guaranteed hardware, such as camera and accelerometer	No guarantee of any specific hardware, must check at run time
Avoid multiple columns of content	Rows and columns of content can work well
Scroll vertically for more content; limited room on the app bar	Scroll horizontally for more content; significant room on the app bar
Hardware back button	On-screen back button
No semantic zoom	Semantic zoom

Figure 2 **Basic Windows Phone 8 and Windows 8 Controls**

Windows Phone 8	Windows 8
PhoneApplicationPage control is the root page element	Page control is the root page element
Use the LongListSelector control to show vertically scrolling content	Use the GridView control to show vertically scrolling content
Use the Pivot control for paging content horizontally	Use the SemanticZoom control for grouping many items into grouped segments (it's also possible to use the Grouped Items Page control)
Use the ApplicationBar control	Use the AppBar control

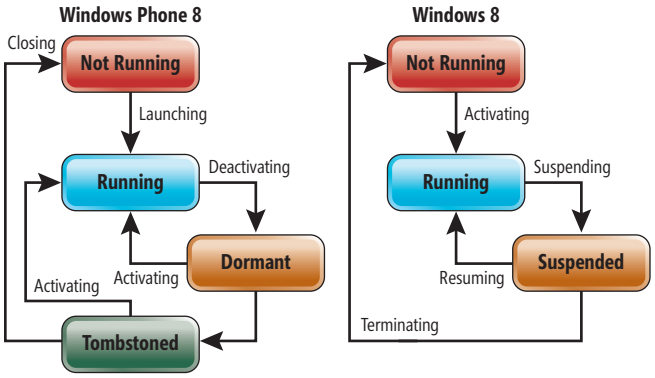


Figure 3 **Windows Phone 8 and Windows 8 Lifecycles**

Data Model and Supporting Code

All applications have to manage data, and data models provide the best way for doing so. They help ensure higher maintainability, modularity, project organization, and portability, and the APIs that support them are good candidates for code reuse. Still, you shouldn't sacrifice application architecture or maintainability for code reuse. Shared libraries, code-sharing techniques and patterns such as Model-View-ViewModel (MVVM) will likely serve you better. For more information on these, see Doug Holland's June issue article, "Code-Sharing Strategies for Windows Store and Windows Phone Apps" (msdn.microsoft.com/magazine/dn201744).

Platform-Specific Features

There are many features that behave similarly in both Windows 8 and Windows Phone 8 but are implemented differently. Because of this, you should avoid reusing code across platforms for certain processes and features, including:

- Application Lifecycle
- System Tasks and Contracts
- Tiles
- Toast Notifications
- Local Storage
- Networking
- Background Processing
- Camera
- App Bar

Application Lifecycle Although there are strong parallels between the application lifecycle on Windows Phone 8 and on Windows 8, the two platforms have different APIs. As you can see in **Figure 3**, the high-level program states and flow between states are similar on both platforms, but resources on Windows Phone 8 are far more constrained, which is the reason for most of the differences in the lifecycle. On both platforms, only the foreground app can consume CPU resources (except for background tasks, which are managed by the OS).

Figure 4 shows differences in lifecycle behavior between the two platforms.

Figure 5 shows the namespaces where application lifecycle events are found.

There are several app entry points on each platform, as **Figure 6** shows.

Figure 4 **Differences in Windows Phone 8 and Windows 8 Lifecycles**

Windows Phone 8	Windows 8
Windows Phone 8 apps are always launched or relaunched.	Windows Store apps are resumed without page navigation.
Windows Phone 8 may tombstone apps.	Windows 8 either suspends or completely terminates apps. Windows Store apps can support hosted-view activation, in which the app performs a single task inside a system-provided UI.
Windows Phone 8 supports Fast Application Resume.	Windows 8 has no need for the Fast Application Resume feature because apps are kept in a suspended state for as long as possible; otherwise, they're terminated.

Figure 5 **Application Lifecycle Namespaces**

Windows Phone 8	Windows 8
Derive from: System.Windows.Application	Derive from: Windows.UI.Xaml.Application
Application_Launching	
Application_Activated	OnLaunched
Application_Deactivated	OnSuspending
Application_Closing	

Figure 6 **App Entry Points**

Windows Phone 8	Windows 8	Both
Speech launch	Search	App tile
App Connect	Share	Secondary tiles
		Toasts
		Protocol and file association

System Tasks and Contracts Both Windows 8 and Windows Phone 8 have built-in support for common operations that involve coordinating with the OS. Windows 8 handles these using contracts, while Windows Phone 8 uses tasks (such as Launchers and Choosers). Contracts and tasks provide similar capabilities to the user, but development is different. The functionality they each expose doesn't necessarily map one-to-one, so you need to carefully consider tasks and contracts when porting an app.

Windows Phone 8 task APIs are found in the Microsoft.Phone.Tasks namespace: PhotoChooserTask, EmailAddressChooseTask, SaveContractTask and so on. In contrast, Windows 8 contracts have individual APIs related to the panes opened by charms (search pane, share pane and the like); support for contracts is declared in the application manifest. Contracts are essentially agreements and implementing them involves overriding methods in the Application class or handling events from classes in the Windows API.

Tiles Tiles are entry points for Windows Store and Windows Phone 8 apps. Primary tiles simply launch apps, while secondary tiles can be pinned to the Start screen; they're effectively a "deep link" that can take the user to a specific page in the app. Both platforms support live tiles, in which content is periodically updated, but the APIs are completely different. **Figure 7** describes Windows Phone 8 and Windows 8 tiles.

Figure 8A and **Figure 8B** show a code comparison of secondary tile creation. On Windows Phone 8, the ShellTile class provides a static interface for creating (and removing) secondary tiles. You can provide this using one of several classes that encapsulate a tile's data: StandardTile, CycleTileData, FlipTileData or IconicTileData. Windows 8 has only one data structure to define a secondary tile (SecondaryTile). Tile creation is done with an asynchronous request, which asks the user if he would like to pin the tile to the Start screen.

Toast Notifications Windows 8 toast notifications are similar to Windows Phone 8 reminders and alarms. They let an app notify the user of an event at a given time. Clicking a toast notification launches the app if it's closed and resumes the app if it's suspended.

Windows Phone 8 supports two types of notifications: alarm and reminder. It can also display toasts (even from the background) using ShellToast. Alarms and reminders are found in Microsoft.Phone.Scheduler, while ShellToast is in Microsoft.Phone.Shell.

Windows 8 uses toast notifications via ToastNotificationManager and ScheduledToastNotification, which are found in the Windows.UI.Notifications namespace and are enabled via the application manifest.

Local Storage The Windows.Storage namespace provides a new set of shared APIs that let applications handle local file management. On Windows Phone 8, System.IO.IsolatedStorage.IsolatedStorageFile provides backward compatibility with earlier Windows Phone versions, but you should move toward using the new APIs. You can use serialization and Windows.Storage.ApplicationData.Current.LocalFolder to persist app data to local app storage.

Both platforms support storage of key/value pairs (settings), and files and folders. However, Windows Phone 8 APIs are a subset of the full APIs and don't provide support for the roaming data store, the temporary data store, local settings or roaming settings. **Figure 9** is a summary of storage options for each platform.

Networking Most applications rely heavily on the Internet capabilities of the devices they reside on to function properly. For both Windows 8 and Windows Phone 8, this capability is declared in the manifest. **Figure 10** shows the relevant networking APIs.

Windows 8 introduces HttpClient, which acts as session to send requests to a server (HTTP, GET, POST and PUT). Each instance of HttpClient has its own connection pool to avoid interference

Figure 7 **Comparison of Windows Phone 8 and Windows 8 Tiles**

Windows Phone 8 Tiles	Windows 8 Tiles
Three sizes: small, medium, wide	Two sizes: smaller, larger
Standard tiles, flip tiles, cyclic tiles and iconic tiles (Microsoft.Phone.Shell)	SecondaryTile can be customized using XML templates (Windows.UI.StartScreen)
Provide tile images in the manifest	Provide tile images in the manifest
Create secondary tiles using the static method ShellTile.Create	Create secondary tiles using the SecondaryTile.RequestCreateAsync method
Update app tile using ShellTile.Update (Microsoft.Phone.Shell); push notification using ShellTileSchedule (ShellTileToast also updates tiles)	Update app tile using TileUpdateManager and TileNotification (Windows.UI.Notifications)
User can unpin the app tile	User can turn off the live tile

Figure 8A **Tile Creation in Windows Phone 8**

```
CycleTileData tileData = new CycleTileData()
{
  Title = item.Title,
  SmallBackgroundImage = new Uri(item.GetImageUri(),
    UriKind.RelativeOrAbsolute),
  CycleImages = list
};

ShellTile.Create(new Uri(navDataSource,
  UriKind.Relative), tileData, true);
```

Figure 8B **Tile Creation in Windows 8**

```
var tile = new SecondaryTile(
  item.UniqueId,      // Tile ID
  item.ShortTitle,    // Tile short name
  item.Title,         // Tile display name
  item.UniqueId,      // Activation argument
  TileOptions.ShowNameOnLogo, // Tile options
  uri                 // Tile logo URI
  );
await tile.RequestCreateAsync();
```

from others. Data is returned as a string, HttpRequestMessage, stream or byte array.

WebClient, on the other hand, is not available in Windows 8 (except for desktop apps). It's used in Windows Phone 8 apps and its implementation is slightly different from the WebClient in earlier versions of Windows. In Windows Phone 8, you can also use HttpWebRequest/HttpWebResponse with an async wrapper. A release preview of a portable HttpClient is now available for both platforms. You can get this preview from the NuGet Gallery.

Background Processing In some situations you might want to run code in the background: Streaming music, pinging a server for updates, downloading a file and so on. Both platforms can give the OS a task to be run when some trigger occurs. These tasks are independent from the app and the OS has complete control over them.

There can only be one background agent per application, but that agent can run one or both of these tasks:

- **Periodic tasks:** short tasks that the OS might run every 30 minutes.
- **Resource-intensive tasks:** longer tasks that run when the phone is plugged in or has ample battery life.

Figure 11 shows what you need in order to implement background tasks on both platforms. Note that Windows Store apps that use background tasks must declare this in the app manifest.

Now I'll demonstrate how to implement a background task, first in Windows 8 and then in Windows Phone 8.

Figure 9 **Storage Options in Windows Phone 8 and Windows 8**

Feature and Namespace	Purpose	Windows Phone 8	Windows 8
Windows.Storage	Local app file storage	Yes	Yes
System.IO.IsolatedStorage.IsolatedStorageFile	Local app file storage	Yes	No
ApplicationData settings (LocalFolder)	Key/value storage	No	Yes
System.IO.IsolatedStorage.IsolatedStorageSettings	Key/value storage	Yes	No
SQL CE	Database	Yes	No

Here are the high-level steps for Windows 8:

1. Implement IBackgroundTask interface and Run method
2. Declare background task use in the manifest
3. Check that the task is not already registered
4. Use BackgroundTaskBuilder to create and register a task instance
5. Handle the BackgroundTaskCompleted event to get data from task:

```
using Windows.ApplicationModel.Background;
namespace MyTaskExample
{
  public class SimpleBackgroundTask : IBackgroundTask
  {
    public void Run(IBackGroundTaskInstance task)
    {
      // Do stuff ...
    }
  }
}
```

6. Declare that the app uses a background task:

```
<Extensions>
  <Extension Category="windows.backgroundTasks"
    EntryPoint="Tasks.MyTask">
    <BackgroundTasks>
      <Task Type="systemEvent" />
    </BackgroundTasks>
  </Extension>
</Extensions>
```

7. And check the registration:

```
foreach (var task in Background.BackgroundTaskRegistration.AllTasks)
{
  // Use if (task.Value.Name == taskName) to see if
  // the task has already been registered
}
// If not ...
var taskBuilder = new BackgroundTaskBuilder();
taskBuilder.Name = taskName;
taskBuilder.TaskEntryPoint = "Tasks.MyTask";
taskBuilder.SetTrigger(
  new SystemTrigger(SystemTriggerType.TimeZoneChange, false));
BackgroundTaskRegistration myTaskRegistration =
  taskBuilder.Register();
```

Here are the high-level steps for Windows Phone 8:

1. Create ScheduledTaskAgent (Windows Phone Scheduled Task Agent template):

```
public class ScheduledAgent : ScheduledTaskAgent
{
  protected override void OnInvoke(ScheduledTask task)
  {
    // Write code here to perform task ...
    NotifyComplete();
  }
}
```

2. Check that task has already been scheduled to run, and if not, create a new one; otherwise, remove the existing task (the task scheduled to run periodically for 10 days):

```
PeriodicTask task =
  ScheduledActionService.Find(taskName) as PeriodicTask;
bool found = (task != null);
if (!found)
  task = new PeriodicTask(taskName);
else
  ScheduledActionService.Remove(taskName);
task.Description = description;
task.ExpirationTime = DateTime.Now.AddDays(10);
ScheduledActionService.Add(task);
```

Unlike with Windows 8, there's no need to declare support for background tasks in the manifest.

Camera Both Windows Phone 8 and Windows 8 support capturing images and videos but, as with other features, the APIs are different.

Capturing photos on Windows Phone 8 is a task handled by the OS. You can register a callback to handle the result when the user exits the camera UI. You may or may not have a photo as a result, so check before using the result. Video recording is not built-in to the camera-capture task, so it requires more work.

Unique to Windows Phone 8 is the ability to create a camera app called lens. You can also create rich media lenses, which provide a unique experience for viewing or editing photos. Moreover, the camera can be used as a gateway to other experiences. For instance, barcode-reader apps use the camera to scan a barcode and then display its associated data. All these lens apps are possible using lens extensibility, which allows launching directly from the lens picker into a viewfinder experience. You can also use special APIs to programmatically access the camera sensor. For more info, see "Lens extensibility for Windows Phone 8" (aka.ms/Vdgk8e) and "Capturing photos for Windows Phone" (aka.ms/Owcwpl) in the Windows Phone Dev Center.

Windows 8 uses the CameraCaptureUI (Windows.Media.Capture) for photos and videos. This requires enabling Webcam and Microphone in the manifest:

```
private async void OnCapturePhoto(object sender, TappedRoutedEventArgs e)
{
  var camera = new CameraCaptureUI();
  var file = await camera.CaptureFileAsync(CameraCaptureUIMode.Photo);

  if (file != null)
  {
      // Do something with the result ...
  }
}
```

Windows Phone 8 uses CameraCaptureTask (Microsoft.Phone.Tasks) to take photos. This requires enabling ID_CAP_ISV_CAMERA and ID_CAP_MICROPHONE in the manifest, as shown in **Figure 12**.

App Bar Windows 8 app bars are much more powerful than Windows Phone 8 app bars. Commonly used actions—such as adding, editing and deleting items—can be placed on an app bar. An app can be notified when the user opens or closes an app bar by handling the Opened and Closed events.

There are significant differences in how the app bar works for each platform, as shown in **Figure 13**.

The app bar declaration in Windows 8 is shown in **Figure 14**. Notice that two stack panels are used to group the buttons, with one group on the left and the other on the right.

Figure 15 shows the app bar declaration in Windows Phone 8 (note that only four ApplicationBarIconButton elements are allowed on the phone ApplicationBar, and no containers are placed within the ApplicationBar element).

Wrapping Up

Increasingly, consumers expect continuity of experience between their smartphones and other devices. The Windows 8 and Windows Phone 8 common core allows for a reasonable level of reuse when building solutions for both platforms. However, as you've seen in this article, it's important to recognize the major differences in the

Figure 10 **Networking API Differences in Windows Phone 8 and Windows 8**

Windows Phone 8	Windows 8	Both
ConnectionManager • System.Net	NetworkInterface • System.Net.NetworkInformation	HttpWebRequest, HttpWebResponse • System.Net
DatagramSocket, StreamSocket • Windows.Networking.Sockets	Socket (TCP or UDP), StreamSocket • System.Net.Sockets	
HttpClient • System.Net.Http	WebClient • System.Net	

Figure 11 **Using Background Tasks in Windows Phone 8 vs. Windows 8**

Windows Phone 8	Windows 8
Derive from BackgroundAgent or one of its subclasses: • Microsoft.Phone and Microsoft.Phone.Scheduler namespaces	Windows.ApplicationModel.Background
Override the OnInvoke method and call NotifyComplete when finished	Implement the IBackgroundTask interface and its Run method
Check that the task is not already registered	Declare background tasks in the manifest; check that the task isn't already registered
Use ScheduledActionService to add and remove background agents	Use BackgroundTaskBuilder to create and register an instance of your task
	Handle the BackgroundTaskCompleted event to get data back from the task

Figure 12 **Enabling ID_CAP_ISV_CAMERA and ID_CAP_MICROPHONE in the Windows Phone 8 Manifest**

```
private readonly CameraCaptureTask cameraTask;
public Init() {
  cameraTask = new CameraCaptureTask();
  cameraTask.Completed += PhotoCaptured;
}
public void TakePhoto() {
  cameraTask.Show();
}
private async void PhotoCaptured (object sender, PhotoResult result) {
  await Task.Run(() => {
    // Do something with the result...
  });
}
```

Figure 13 **Differences in How the App Bar Works in Windows Phone 8 and Windows 8**

Windows Phone 8	Windows 8
One app bar at the bottom of the page	Two app bars: one at the bottom and one at the top
Only four items allowed • Put additional items on the menu • No grouping	Behaves like any container • No menu • Can group items in nested containers
ApplicationBar control inside PhoneApplicationPage.ApplicationBar	AppBar control inside Page.BottomAppBar or Page.TopAppBar
Set Mode to Default to show the app bar when the page loads	Set IsOpen to true to show the app bar when the page loads
Set IsMenuEnabled to enable the menu	Set IsSticky to true to force an app bar to always remain open

Figure 14 **The App Bar Declaration in Windows 8**

```
<Page.BottomAppBar IsOpen="True">
  <AppBar x:Name="bottomAppBar" Opened="AppBar_Opened" Padding="10,0,10,0">
    <Grid>
      <StackPanel Orientation="Horizontal" HorizontalAlignment="Left">
        <Button Style="{StaticResource TakePictureAppBarButtonStyle}"
          Click="TakePicture_Click"/>
        <Button Style="{StaticResource ShareTaskAppBarButtonStyle}"
          Click="ShareImage_Click"/>
      </StackPanel>
      <StackPanel Orientation="Horizontal" HorizontalAlignment="Right">
        <Button Style="{StaticResource StartCoookingAppBarButtonStyle}"
          Click="StartCooking_Click"/>
        <Button Style="{StaticResource PinToStartAppBarButtonStyle}"
          Click="PinToStart_Click"/>
      </StackPanel>
    </Grid>
  </AppBar>
</Page.BottomAppBar>
```

Figure 15 **The App Bar Declaration in Windows Phone 8**

```
<phone:PhoneApplicationPage.ApplicationBar x:name="ddd">
  <shell:ApplicationBar x:Name="bottomAppBar" IsVisible="True"
    IsMenuEnabled="True"  Mode="Default" Opacity="1.0">
    <shell:ApplicationBarIconButton x:Name="btnTakePicture"
      IconUri="/Assets/Icons/camera.png" Click="btnTakePicture_Click"
      Text="Take Picture"/>
    <shell:ApplicationBarIconButton x:Name="btnShareTask"
      IconUri="/Assets/Icons/share.png" Click="btnShareShareTask_Click"
      Text="Share Image"/>
    <shell:ApplicationBarIconButton x:Name="btnStartCooking"
      IconUri="/Assets/Icons/alarm.png" Click="btnStartCooking_Click"
      Text="Start Cooking"/>
    <shell:ApplicationBarIconButton x:Name="btnPinToStart"
      IconUri="/Assets/Icons/like.png" Click="btnPinToStart_Click"
      Text="Pin To Start"/>
  </shell:ApplicationBar>
</phone:PhoneApplicationPage.ApplicationBar>
```

physical devices and the way in which users interact with them, as well as the differences in the APIs for each. These differences mean application designers must account for building a separate UI and UX, which in turn means little or no reuse of XAML across the platforms. Although some APIs are available for both platforms, as with Windows.Store.StorageFile, many features are externally similar but implemented differently. That's the case for lifecycle events, tiles, media capture, tasks and contracts, settings, background tasks, and the app bar.

Once you grasp these differences, you can confidently exploit cross-platform code-reuse techniques such as Portable Class Libraries, runtime components and so forth. To learn more about these techniques, please refer to Doug Holland's code-sharing article I mentioned earlier, as well as to the Windows Phone Dev Center article at aka.ms/sharecode. And you'll want to download the sample code showing sharing in action from aka.ms/gxcvq3. ■

Joel Reyes *is a technology evangelist in DPE Public Sector focused on Windows Phone 8 and Windows 8. You can reach him at joel.reyes@microsoft.com.*

Thanks to the following technical experts for reviewing this article: *Andrew Byrne (Microsoft) and Matthias Shapiro (Microsoft)*

Build Modern Business Productivity Apps with Visual Studio LightSwitch

Jan Van der Haegen and Beth Massi

In this article we'll take a look at some of the new features in Visual Studio LightSwitch that help you build modern, mobile, business productivity apps.

Visual Studio LightSwitch (just LightSwitch from here on for brevity) is designed to simplify and shorten the development of businesses apps. LightSwitch apps can consume a variety of data sources, provide business and authorization logic, and create clients that can run on a variety of devices—all without the need to write plumbing code.

LightSwitch can aggregate multiple data sources, and it exposes a set of open data services (the Open Data protocol, or OData) automatically to support custom clients and self-service business intelligence (BI) scenarios. LightSwitch also lets you write code to customize the apps as necessary, whether that's the UI controls, business logic, data services or other components.

With the release of Visual Studio 2012 Update 2 in April, LightSwitch also has added the ability to quickly build touch-centric HTML5 clients that run well on modern mobile devices. LightSwitch HTML clients are built on standards-compliant HTML5 and JavaScript and provide modern, touch-first experiences on Windows RT, Windows Phone 8, iPhones and iPads with iOS 5 and 6, and Android 4.x devices.

Additionally, with the new SharePoint 2013 app model, LightSwitch is also bringing the simplicity and ease of building custom business apps into SharePoint and Office 365. Many enterprises today use SharePoint as a hub to enable better collaboration between people, content and processes. Although you can still choose to host your apps on your own or in Windows Azure, enabling SharePoint in your LightSwitch apps allows you to take advantage of the app life-cycle management, identity and access control capabilities within SharePoint—not to mention the business data and processes already running in SharePoint in your enterprise.

This article discusses:

- The LightSwitch HTML client
- Screen templates and navigation
- App customization
- Integrating with SharePoint 2013 and Office 365
- Working with the SharePoint Client-Side Object Model
- SharePoint deployment considerations

Technologies discussed:

Visual Studio LightSwitch, SharePoint 2013, Office 365

Code download available at:

archive.msdn.microsoft.com/mag201307LightSwitch

The LightSwitch HTML Client

In LightSwitch you always start with the data model, whether you're modeling new data or connecting to existing data sources. Because in this article we're focusing on new features, we've already created

a simple data model and filled the database with some initial data. This way, you're all set up and ready to start creating a client that you can use to make sure you sit in on the most interesting sessions when attending an exciting conference, such as Visual Studio Live! In case this is your first introduction to LightSwitch, make sure you don't miss out on the wealth of data modeling, self-service BI, concurrency handling and multithreading goodness it has to offer by reading *MSDN Magazine*'s previous articles—such as "Shape Up Your Data with Visual Studio LightSwitch 2012" in the September 2012 issue (msdn.microsoft.com/magazine/jj618303)—or by checking out the LightSwitch Developer Center (msdn.com/lightswitch).

The HTML client provides an approach to building touch-first single-page apps that run on a broad range of mobile devices, often referred to as "companion apps" to signify that these apps play just one specific role in a larger architecture. Creating such a companion app to browse data on location—as we do in this article—is a great exercise to get started with the HTML client, but is in no way representative of the broad range of problems these apps can solve. These companion apps can help your business save time and money by allowing data not only to be consulted, but also modified and stored back (collecting a customer's signature when a package is delivered, for example), and sometimes even making use of some device-specific capabilities such as geolocation (capturing the location where a soil sample is collected, for example).

Screen Templates and Navigation No matter what type of client you create, LightSwitch takes the same approach to defining screens using a set of predefined templates from which you can choose. To add a screen, simply right-click on the project in Solution Explorer and select the Add Screen option. This opens a dialog where you can select a screen template and some additional options such as the name of the screen and the data it will use. This screen then will be generated based on the chosen template and opened in the screen designer.

For the somewhat-experienced LightSwitch developer, this already feels familiar, because the design-time experience is almost exactly the same as when using LightSwitch to create screens in an out-of-browser Silverlight app, simplifying the already easy learning process. LightSwitch has stayed faithful to the app's underlying Model-View-ViewModel (MVVM) architecture and the way the screen designer shows view models on the left-hand side and a representation of the corresponding views on the right.

> With the new SharePoint 2013 app model, LightSwitch is also bringing the simplicity and ease of building custom business apps into SharePoint and Office 365.

One notable change is that setting up navigation from one screen to another (or binding any other action to events caused by the user tapping on elements in the screen) has really become a first-class citizen in the screen designer. This is an obvious improvement, not only because apps are so touch-focused, but also because the LightSwitch HTML client produces a single-page application (SPA). The Silverlight client had a Multiple Document Interface (MDI) shell that allowed many screens to be open at the same time; hence, in LightSwitch HTML apps, navigating between screens becomes incrementally more important.

For example, if you use the Browse Data template on the session entity to create the first screen in this app, a screen is generated that contains a list of sessions. (This list is selected in the screen designer, and then the Edit Item Tap Action link is clicked in the Properties window.) This opens up a dialog that's highly adaptive in its suggestions: Because the user is browsing the sessions and has tapped on a single item, the dialog suggests the action of opening a screen where the user can view the details of the selected session. Because such a screen hasn't been created yet, the dialog is suggesting that you go ahead and create that screen as well. This is shown in **Figure 1**.

After accepting, pressing F5 builds the app and starts a new debugging session in the default browser. The home screen opens and displays a simple list of sessions.

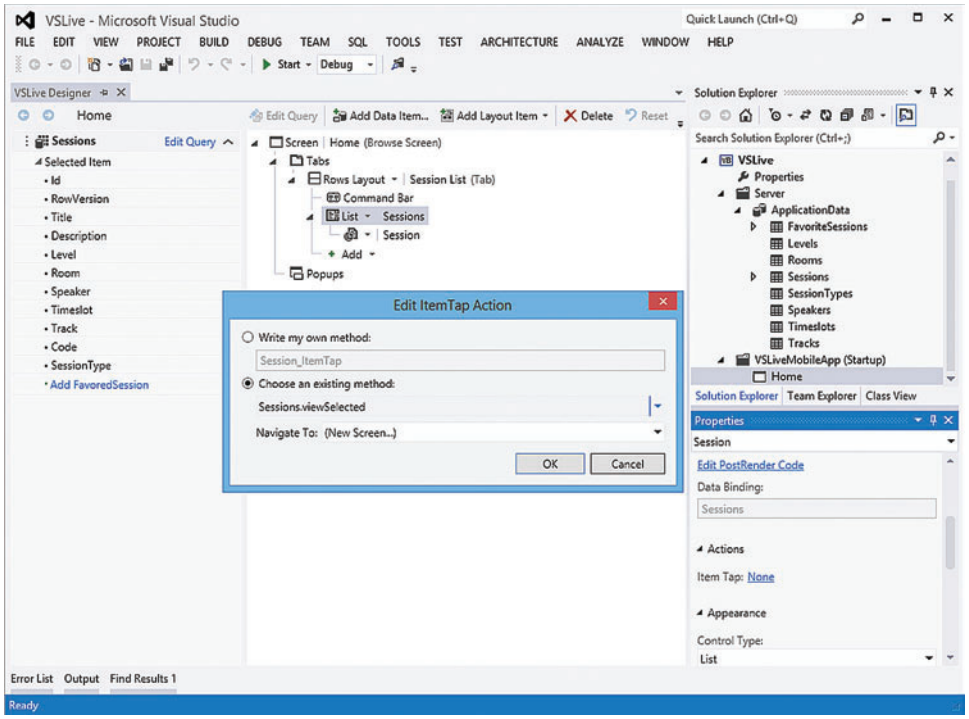


Figure 1 **Setting Up Actions for Touch Events Is Now a First-Class Citizen in the Screen Designer**

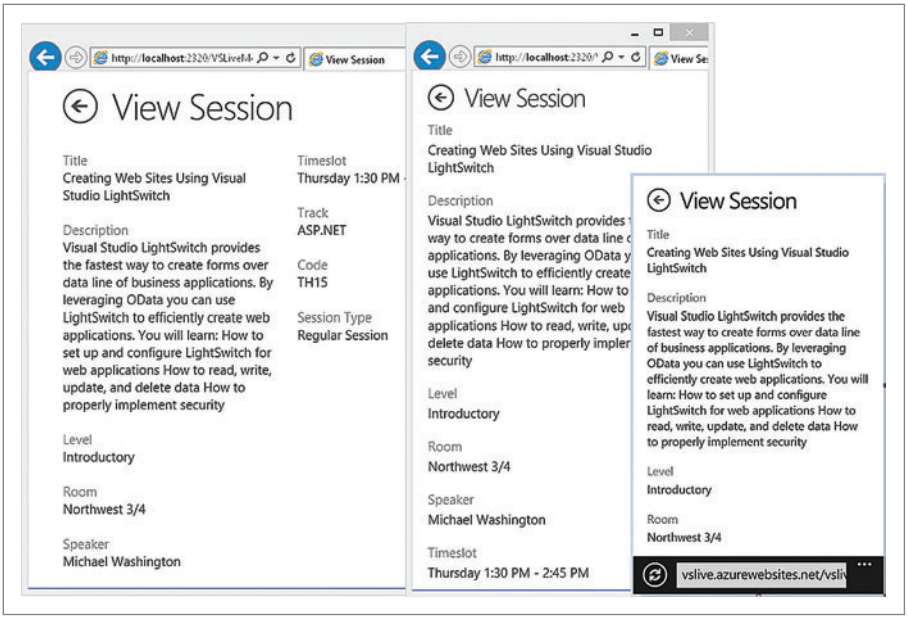


Figure 2 **Adaptive Design Helps to Write a Single App for Multiple Form Factors**

When you tap on a session, the app smoothly navigates to the newly generated detail screen that displays the details of the particular session in a simple two-column layout, shown in **Figure 2**.

Although this screen looks simple, there's a lot going on behind the scenes. This "simple list of sessions" is actually backed by a completely virtualized collection that loads more elements as the user scrolls down, to avoid unnecessarily transferring large amounts of data over a mobile connection. The two-column layout dynamically adapts to the available size, changing to a single-column layout on smaller devices. This is a vital element in the strategy to write a single HTML app and run it on a variety of tablets and phones, each with its own form factor. The Visual Studio LightSwitch Team Blog post, "Designing for Multiple Form Factors" (bit.ly/18F320N) by Program Manager Heinrich Wendel, provides more detail about the vision behind this adaptive design, which normally requires a great deal of work from the developer, but which LightSwitch simply offers with little effort.

Other notable elements in the screen designer include the promoted usage of pop-ups and tabs, which each have their own command bar. One example of how these can make life easier for the end user is to add a filter on the sessions. To do this, you first select the Sessions query element in the view model (the left-hand side of the screen designer) and click Edit Query, which opens the query editor where you can quickly modify the query that's used by adding some optional arguments. Next, hit the Add Data Item button to add a local Speaker property called FilterBySpeaker, set up data binding of this Speaker's Id to the corresponding query argument, and repeat this process for the other entities: Room, Time Slot and Track.

To complete the screen, simply drag these four new view model properties onto a newly created pop-up in the view (the right-hand side of the screen designer). By default, LightSwitch will suggest to visualize each property as a Modal Picker control, which is just fine for this app. Finally, add a new button to the command bar and use the same dialog that appears when setting up the Edit Item Tap Action to open the newly created pop-up.

Hitting Save and refreshing the browser is all it takes to view the newly added filter in action (JavaScript doesn't need recompilation, as do traditional Microsoft .NET Framework apps). The home screen opens, showing all sessions in a virtualized collection—and this time also showing a blue command bar with a single Filter button. When this button is clicked, a pop-up is displayed showing the four filter options (shown in **Figure 3**) on a couple different devices. As soon as you actually make a selection, the optional query arguments are automatically updated because of the data binding. This in turn automatically triggers a new HTTP GET operation to be sent to the back-end OData service, which filters the sessions by the correct arguments and returns the result.

In summary, the screen designer does an excellent job of making the correct abstractions so you can focus on setting up a professional HTML app with minimal effort. For example, this app includes a virtualized list that can be filtered, all of which has been set up without having to write a single line of code.

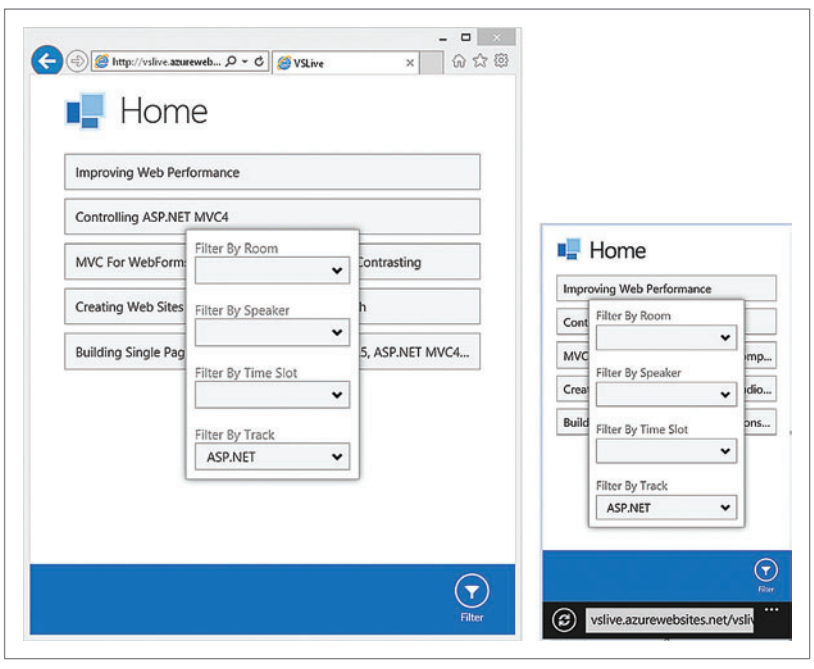


Figure 3 **The Added Filter in Action**

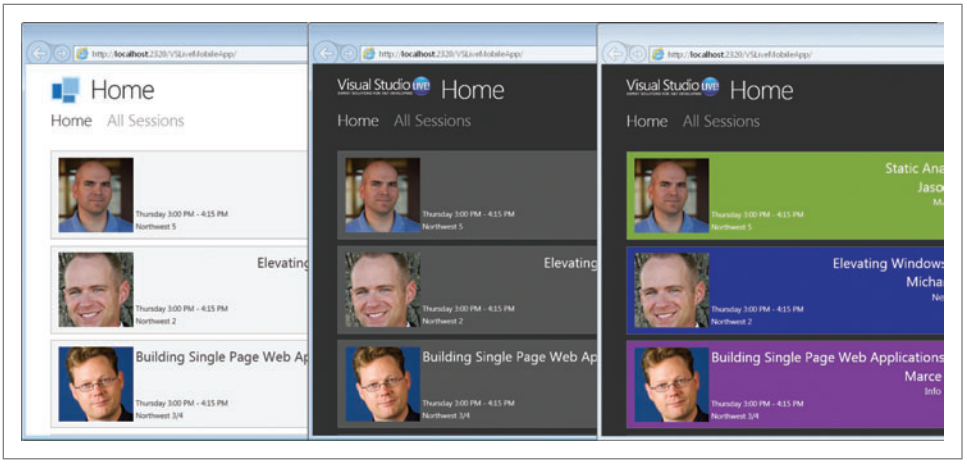


Figure 4 **Three Different Designs for the Home Screen**

Note, however, that as a developer you shouldn't get too hung up on that no-code philosophy.

Yes, it's true that the screen designer focuses on abstracting the actual technology to increase developer velocity, and that like all rapid-application development (RAD) environments, this implies sacrificing some customizability. However, LightSwitch offers many scenarios in the screen designer, as well as a large number of extension points, so you can roll up your sleeves and override, alter or append to the HTML, JavaScript and CSS mix that forms an app.

> LightSwitch does an excellent job of making the correct abstractions so you can focus on setting up a professional HTML app with minimal effort.

As an example, you can add a new query to the app that returns only those sessions in the next Time Slot as a Data Item on the screen, and drag that query onto a new tab in the home screen. Again, you can keep the List control to show the sessions, but instead of having a one-line summary per session (the title), the screen designer is used to nest some row and column layouts and display some additional information about the session, which results in the first (on the left-hand side) layout in **Figure 4**.

Branding, Theming and Further Customization When it comes time to start mixing in custom code, it's quick and easy to apply some global theming and branding. The logo, in fact, is nothing more than a .png file that you can replace from Windows Explorer, and the theme is nothing more than a standard JQuery Mobile theme. LightSwitch ships with a light (default) and a dark theme, both of which you can further modify using the online JQuery Mobile ThemeRoller. Changing both the logo and the theme results in the center layout in **Figure 4**.

Finally, you can further define the look of your app by using Render or PostRender methods. By selecting any LightSwitch control in the screen designer, a link in the Properties window labeled Edit PostRender Code becomes available. When you click this link, a JavaScript method stub is generated that will be executed at run time, directly after the LightSwitch JavaScript library has rendered the required HTML elements for the particular control. A possible use of this extension point is to alter the background of each row in the list, depending on the track that the session is a part of, by writing the following code in that method:

```
myapp.Home.NextSessionsTemplate_postRender =
  function (element, contentItem) {
  $(element).parent("li").css("background", contentItem.value.Track.Color);
};
```

This code grabs the parent of the HTML element that LightSwitch added and sets the CSS background property to the color of the track of the current session (contentItem.value), resulting in the right-most layout in **Figure 4**.

Figure 4 reveals an obvious fault in design: If the track has a bright color associated with it, then because of the contrast between the content and the background, an item can become unreadable. To correct this, add two reusable CSS classes to the user-customization.css file and alter the code so that it appends one of these classes:

```
myapp.Home.NextSessionsTemplate_postRender =
  function (element, contentItem) {
  $(element).parent("li").css("background", contentItem.value.Track.Color);
  $(element).addClass(
    (parseInt(contentItem.value.Track.Color.replace(
    "#", ""), 16) > 0xffffff / 2)
    ? 'darkForeground' : 'lightForeground'
  );
};
```

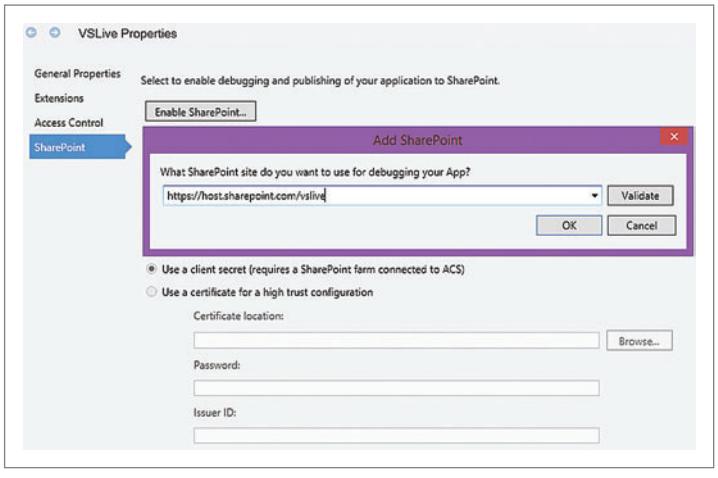


Figure 5 **Enabling SharePoint in a LightSwitch App**

Customizing the rendering process by writing PostRender methods can be used for a variety of options, ranging from simple CSS tricks such as changing the background color to applying a reusable JQuery UI widget that hides all items in the list but one, turning the list into a dynamic banner or image carousel. Visible items are changed with a set timer, as described in the blog post (written by coauthor Jan Van der Haegen), "Create a JQuery slider for LightSwitch HTML pages" (bit.ly/WJwnPw).

After turning the list of upcoming sessions into a banner with a color-coded background, it's easy to forget which track has which color. Besides altering standard LightSwitch controls using this PostRender option, you can also take complete control over the entire rendering of specific elements by indicating they're custom controls in the screen designer and then writing your own code in the similarly generated JavaScript Render method. To add a legend of tracks and their color code, add a query on all tracks as a date item to the screen view model and display it as a Tile List (a built-in alternative collection control). In the template of each tile, render the color of each track as a simple square, using this JavaScript code:

```
myapp.Home.Color_render = function (element, contentItem) {
  $("<div style='background-color: "+ contentItem.value
  +";'> </div>").appendTo($(element));
};
```

Besides various visual customizations, LightSwitch also offers numerous code extension points in the ViewModel and Model layers to help you tweak the control flow and business rules of your apps.

> You can roll up your sleeves and override, alter or append to the HTML, JavaScript and CSS mix that forms an app.

This app now has a first tab that features the upcoming sessions (in the form of a banner) and has a legend of colors per track, along with a second tab with an overview of all the sessions, including an easy-to-use filter. Because the home screen already shows all of the tracks, it's a good idea to fill in the filter on the second tab when the user taps on one of the tracks in the first tab, then focus this tab to show only the sessions in the selected track.

This control flow can't be set up directly from the screen designer, but it's easily accomplished by selecting the Tile List of tracks and in the Properties Window binding the item tab to execute to a custom method. This generates a JavaScript method stub where you can write the following:

```
myapp.Home.BrowseByTrack_ItemTap_execute = function (screen) {
  screen.getTracks().then(function (tracks) {
    screen.FilterByTrack = tracks.selectedItem;
    screen.showTab("AllSessions");
  });
};
```

The app now has three different perspectives to slice the sessions: see the upcoming sessions in a banner; click on a track to see the list of sessions filtered by that track; or adjust track, time slot, speaker or room on that same filter manually.

One last perspective you can add is an overview of your favorite sessions. To accomplish this, add a simple entity called FavoriteSessions. Favorite sessions are those in which an entry in FavoriteSessions exists with a particular name.

When the screen to view a particular session is opened, determine if such an entry exists by asynchronously executing a specialized FindFavoriteSessions query on the server and storing the existence of this result in a Boolean screen property called IsFavorite:

```
myapp.ViewSession.created = function (screen) {
  myapp.activeDataWorkspace
    .ApplicationData.FindFavoriteSessions(screen.Session.Id)
    .execute().then(
      function (result) {
        screen.IsFavorite = result.results.length != 0;
      }
    );
};
```

Then add two buttons to the tab's command bar: Favorite and Unfavorite. Only one of these buttons can be active at the same time, so use the button's CanExecute code extension point. The LightSwitch JavaScript library calls these methods at the appropriate times and makes sure the UI is updated accordingly. Although this behavior can be overridden for each screen from the Properties window, it does so by default by hiding buttons that can't be executed at the moment, saving precious screen real estate on small devices:

```
myapp.ViewSession.Favorite_canExecute = function (screen) {
  return !screen.IsFavorite;
};
myapp.ViewSession.Unfavorite_canExecute = function (screen) {
  return screen.IsFavorite;
};
```

One amazing highlight about such simple code snippets is that the LightSwitch JavaScript library keeps track of when these CanExecute methods should be reevaluated. Just like in Silverlight out-of-browser apps, you don't need to take care of throwing events when a property changes. Just set a new value for the IsFavorite property from anywhere within the app and the LightSwitch runtime will automatically know to reevaluate these particular CanExecute methods and update the view (show or hide the buttons) as required.

All that's left to do now is to write some code that creates a new FavoriteSession (or removes it) when the button is clicked:

```
myapp.ViewSession.Favorite_execute = function (screen) {
  var favored = myapp.activeDataWorkspace.ApplicationData.
    FavoriteSessions.addNew();
  favored.setSession(screen.Session);
  myapp.applyChanges().then(screen.IsFavorite = true);
};
```

After adding the list of FavoriteSessions to a new tab on the homepage, the app is ready for packaging and deployment.

It turns out that LightSwitch does a great job of keeping a balance between designing the app in a simple but powerful screen designer and the amount of customizability you need to sacrifice for this simplified and speedy development. It achieves this balance by introducing a large number of extension points where you can write custom code to alter, override or append the default design, control flow or behavior of the app.

Integrating with SharePoint 2013 and Office 365

SharePoint 2013 introduces a new cloud-based app model that allows you to create apps that extend the capabilities of a SharePoint Web site and supports a tiered architecture in which the app's

business logic, data and UI can be distributed. Users discover and download apps from the Office store or from their organization's private app catalog and install them on their SharePoint sites. Because of this new distributed model, you can build LightSwitch apps that target SharePoint 2013 on-premises and on Office 365. You can deploy the LightSwitch Silverlight in-browser clients or the new mobile-based HTML clients into SharePoint.

Building upon the Visual Studio Live! example, you can take advantage of Office 365 services in order to allow conference staff to set up the conference schedule using a SharePoint calendar within the LightSwitch app. When a new session is approved and a time slot is chosen, this will add a new calendar entry. You can then imagine a series of possible workflows that may trigger additional actions that are common with SharePoint. Thus, LightSwitch apps allow you to create touch-enabled experiences that take advantage of business data and processes that are already running in SharePoint in your business.

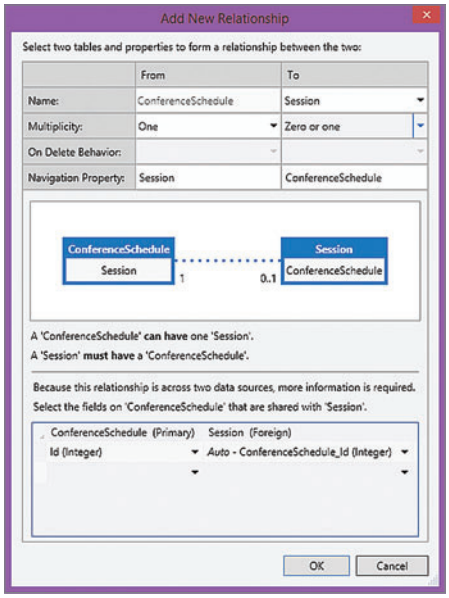


Figure 6 **Setting Up a Virtual Relationship Between Multiple Data Sources**

> When you bring SharePoint lists into the LightSwitch data model, it also allows you to access the list data via code in a streamlined way using the LightSwitch API.

Enabling SharePoint in LightSwitch Apps In the project properties you click the Enable SharePoint button and provide a local or remote SharePoint 2013 site you want to use for development. When you do this, you're indicating to LightSwitch that the app should be deployed to SharePoint and can work with SharePoint assets. This adds the appropriate SharePoint references to your project.

LightSwitch will also handle authentication to SharePoint automatically for you via OAuth. You can configure Access Control Service (ACS), which is another server that brokers authentication between your app and SharePoint. Your app will have a "secret" known to SharePoint and can use it to authenticate via ACS, as is the case with Office 365. You can also choose a high-trust configuration if you have that set up on-premises, as shown in **Figure 5**.

To get started developing SharePoint apps quickly, you can sign up for a free trial of the Office 365 developer subscription at dev.office.com. MSDN subscribers can also sign up for a free year through their subscription benefits site.

SharePoint Lists as LightSwitch Entities Using the Data Designer, you can bring in SharePoint lists as entities into the LightSwitch data model. This allows you to manipulate list data directly through the LightSwitch data context. This means you can create screens directly against SharePoint list data that you already have set up, and LightSwitch will handle the authentication and data operations automatically. For example, in the Visual Studio Live! scenario, any of the data could be stored in SharePoint lists and the UI wouldn't change.

When you bring SharePoint lists into the LightSwitch data model, it also allows you to access the list data via code in a streamlined way using the LightSwitch API. For example, in the Visual Studio Live! app, we wanted to write code to automatically add a SharePoint calendar entry when a new session is added. To do this, right-click on the Server node in the Solution Explorer and choose "Add data source," then select SharePoint as the data source type. After indicating the SharePoint site address and selecting to automatically choose the user identity, the SharePoint lists are displayed.

The site being used already contains a calendar called ConferenceSchedule. When this is brought into the data model you can set up a virtual relationship to the session table data. Simply click the Relationship button at the top of the data designer and define a One-to-Zero-or-one relationship (see **Figure 6**).

Now you can create a new screen that allows staff to enter new sessions. You could include the ConferenceSchedule data directly on the screen, but in this case some server code will be written so

Figure 7 **Code to Add a New Entry to the SharePoint Calendar Using Visual Basic .NET**

```
Private Sub Sessions_Updated(entity As Session)
  Me.UpdateCalendar(entity)
End Sub

Private Sub Sessions_Inserting(entity As Session)
  Me.UpdateCalendar(entity)
End Sub

Private Sub UpdateCalendar(entity As Session)
  If entity.Timeslot IsNot Nothing AndAlso
    entity.Room IsNot Nothing Then

      If entity.ConferenceSchedule Is Nothing Then
        entity.ConferenceSchedule = New ConferenceSchedule()
      End If

      entity.ConferenceSchedule.Title = entity.Title
      entity.ConferenceSchedule.Description = entity.Description
      entity.ConferenceSchedule.StartTime = entity.Timeslot.StartTime
      entity.ConferenceSchedule.EndTime = entity.Timeslot.EndTime
      entity.ConferenceSchedule.Location = entity.Room.Name

      Me.DataWorkspace.VSliveData.SaveChanges()
  End If
End Sub
```

Figure 8 **Code to Add a New Entry to the SharePoint Calendar Using C#**

```
private void Sessions_Updated(Session entity)
{
  this.UpdateCalendar(entity);
}

private void Sessions_Inserting(Session entity)
{
  this.UpdateCalendar(entity);
}

private void UpdateCalendar(Session entity)
{

  if (entity.Timeslot != null && entity.Room != null)
  {
    if (entity.ConferenceSchedule == null)
    {
      entity.ConferenceSchedule = new ConferenceSchedule();
    }

    entity.ConferenceSchedule.Title = entity.Title;
    entity.ConferenceSchedule.Description = entity.Description;
    entity.ConferenceSchedule.StartTime = entity.Timeslot.StartTime;
    entity.ConferenceSchedule.EndTime = entity.Timeslot.EndTime;
    entity.ConferenceSchedule.Location = entity.Room.Name;

    this.DataWorkspace.VSliveData.SaveChanges();
  }
}
```

it happens transparently in the background. When the session data is saved, use the save pipeline's _Inserting and _Updated methods on the Session entity to add a new entry to the schedule from the server using Visual Basic .NET or C# code (see **Figure 7** and **Figure 8**, respectively). There are many hooks on the server that allow you to control business logic and data processing this way.

When you debug the app (F5) you'll first be asked if you trust it. Once you confirm that you do, you'll see a SharePoint chrome control at the top that allows users to navigate up to the SharePoint site from the LightSwitch app. Add a new session, then click "Back to site" in the chrome to see the Conference Schedule. If you click on the Event item, you'll see it was created by the LightSwitch app on behalf of the user. This shows that the SharePoint user credentials flow through the LightSwitch app automatically.

The SharePoint Project and Client-Side Object Model Using lists as entities isn't the only way to access SharePoint data. When you enable SharePoint in your LightSwitch apps, you have the full SharePoint Client-Side Object Model (CSOM) available to you, as well. You can manipulate not only lists this way but also access other SharePoint assets. For example, if you wanted to create a general app that you could sell in the SharePoint store, you might need to deploy and manipulate SharePoint lists in a more generic way.

When you enable SharePoint, a SharePoint project is added to your LightSwitch solution. If you flip to File View on the Solution Explorer, you'll see it as shown in **Figure 9**. You can add a variety of SharePoint-specific items to your app this way that will get deployed when you package your app. Then you can use CSOM to manipulate these items in code. For example, you can add a custom list to your project and then interact with that list via CSOM. LightSwitch exposes a SharePoint object on the Application object that you can use to get at the host and app webs.

> When you enable SharePoint in your LightSwitch apps, you have the full SharePoint Client-Side Object Model (CSOM) available to you.

For example, we added a custom list to the VSLive.SharePoint project that's used to write audit entries anytime a session is added, updated or deleted. We can tap into the save pipeline again and use the list via CSOM. First, import the Microsoft.SharePoint.Client namespace, and then you can write the code shown in **Figure 10** (Visual Basic .NET) or **Figure 11** (C#).

For more information on using CSOM with LightSwitch, as well as other tips and tricks, see the SharePoint topics on the Visual Studio LightSwitch Team Blog at bit.ly/16JIWn6.

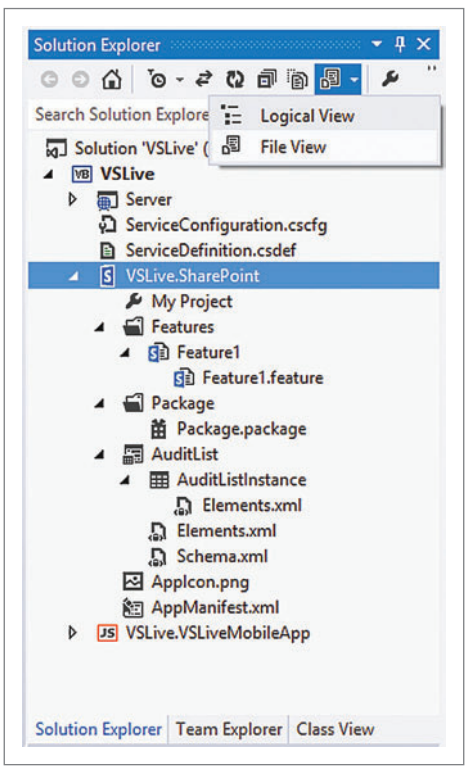


Figure 9 **Adding Custom Lists and Other SharePoint Items via the Solution Explorer**

SharePoint Deployment Considerations LightSwitch apps are just Web apps, so you can host them on your own IIS Web servers or in a cloud such as Windows Azure. When you install a SharePoint app, a manifest is installed that has information on where your app runs. As part of the new SharePoint app model, any apps that contain server-side code—which all LightSwitch apps do—must run on a separate server outside of SharePoint. This isolation provides better stability to the SharePoint farm.

LightSwitch supports two types of SharePoint deployment: autohosted and provider-hosted. You need to consider these options when building your app because they might directly affect the architecture of your data.

With autohosted apps, the Web site and database are provisioned automatically into Windows Azure each time the app is installed. The data is provisioned into SQL Azure, and the middle tier is provisioned into a Windows Azure Web site. This means that

Figure 10 **Manipulating a SharePoint List via CSOM in Visual Basic .NET**

```
Private Sub Sessions_Inserted(entity As Session)
  Me.LogAuditEntry(entity, "inserted")
End Sub

Private Sub Sessions_Updated(entity As Session)
  Me.LogAuditEntry(entity, "updated")
End Sub

Private Sub Sessions_Deleted(entity As Session)
  Me.LogAuditEntry(entity, "deleted")
End Sub

Private Sub LogAuditEntry(
  entity As Session, actionDescription As String)

  Using ctx = Me.Application.SharePoint.GetAppWebClientContext()

    Dim web As Web = ctx.Web
    Dim lists As ListCollection = web.Lists
    Dim auditLog As List = lists.GetByTitle("AuditList")
    Dim ci As New ListItemCreationInformation()
    Dim auditEntry As ListItem = auditLog.AddItem(ci)

    auditEntry("Title") = String.Format("Session {0} {1}",
                                        actionDescription,
                                        entity.Title)

    auditEntry("Name") = Me.Application.User.FullName
    auditEntry("Date") = DateTime.Now

    auditEntry.Update()
    ctx.ExecuteQuery()
  End Using
End Sub
```

Figure 11 **Manipulating a SharePoint List via CSOM in C#**

```
private void Sessions_Inserted(Session entity)
{
  this.LogAuditEntry(entity, "inserted");
}

private void Sessions_Updated(Session entity)
{
  this.LogAuditEntry(entity, "updated");
}

private void Sessions_Deleted(Session entity)
{
  this.LogAuditEntry(entity, "deleted");
}

private void LogAuditEntry(Session entity, string actionDescription)
{
  using (ClientContext ctx =
    this.Application.SharePoint.GetAppWebClientContext())
  {
    Web web = ctx.Web;
    ListCollection lists = web.Lists;
    List auditLog = lists.GetByTitle("AuditList");
    ListItemCreationInformation ci = new ListItemCreationInformation();
    ListItem auditEntry = auditLog.AddItem(ci);

    auditEntry["Title"] = string.Format("Session {0} {1}",
                                        actionDescription,
                                        entity.Title);

    auditEntry["Name"] = this.Application.User.FullName;
    auditEntry["Date"] = DateTime.Now;

    auditEntry.Update();
    ctx.ExecuteQuery();
  }
}
```

each instance of your LightSwitch app that's installed into SharePoint is isolated from all other instances on other SharePoint sites. This is a quick and easy type of deployment. However, if your app is uninstalled, then everything is deleted, even the data.

Provider-hosted apps give you the flexibility of hosting the Web app and database wherever you want. However, with this model, all SharePoint app instances share the same middle tier and data, so you'll need to provide your own tenant isolation. LightSwitch provides row-level filtering mechanisms in the query pipeline to help build multi-tenant apps. You also need to manage availability—if your server goes down, all your SharePoint apps stop working.

You also need to specify how you want your app to authenticate. This is the "ACS versus high trust" choice you make when you start developing with SharePoint. If your SharePoint server or farm is using ACS, as is the case with Office 365, then all that's required is a ClientID and secret that's shared between your app and SharePoint.

You use the LightSwitch Publish Wizard to specify these settings and package your app. Depending on the type of hosting, it will either deploy your app service and database to a server you specify (or Windows Azure), as is the case with provider-hosting, or it will wrap everything into the SharePoint deployment package. You then take that package and install it onto your SharePoint site catalog. From there, users can add your app to their sites.

For more information on hosting LightSwitch apps in SharePoint, see the following Visual Studio LightSwitch Team Blog posts: "SharePoint Hosting and Authentication Options for LightSwitch" (bit.ly/10vEJez) and "Publishing LightSwitch apps for SharePoint to the Catalog" (bit.ly/11wSFqo).

Wrapping Up

As you can see, there are a ton of new features in the latest release of LightSwitch in Visual Studio 2012 Update 2. These features complement the desktop-based Silverlight client that's been available since the first release—and still is today. However, with many employees bringing their own devices to the workplace, building a native client for every device and deploying each implementation to separate app stores is rarely feasible. LightSwitch HTML clients are built on standards-compliant HTML5 and JavaScript and provide modern, touch-first experiences for today's mobile devices.

Additionally, enabling SharePoint in your LightSwitch apps allows you to take advantage of the business data, processes, app lifecycle management, identity and access-control capabilities within SharePoint that are already embedded into many enterprises today. ■

Jan Van der Haegen *is a green geek who turns coffee into software. He's a loving husband, .NET addict, LightSwitch lover, blogger, independent consultant and columnist for* MSDN Magazine *online. He secretly dreams of becoming a professional goat herder one day. You can find his coding experiments at switchtory.com/janvan.*

Beth Massi *is a senior program manager on the Visual Studio team at Microsoft and is a community champion for business application developers. She's a frequent speaker at various software development events, and you can find her on a variety of developer sites including MSDN.com, Channel 9 and her blog, bethmassi.com. Follow her on Twitter at twitter.com/BethMassi.*

Thanks to the following technical experts for reviewing this article: *Brian Moore (Microsoft) and John Stallo (Microsoft)*

RACHEL APPEL

Mastering Controls and Settings in Windows Store Apps Built with JavaScript

Great user experiences present data to users in natural and intuitive ways regardless of form factor. Presenting data and content requires updated APIs, controls and tools for building modern experiences. In Windows Store apps, the amount of code required and the complexity of controls depends on what kind of app you create, whether it's a productivity app, game, social app or financial app. Windows Library for JavaScript (WinJS) controls are easy to master for any developer building Windows Store apps with JavaScript, and they're what I'll discuss here.

Windows 8 Brings a New UI Paradigm and New UI Controls

Windows Store apps look and behave much differently than programs running in previous versions of Windows. There have been some major renovations in Windows, starting with the new Start page full of live tiles as your first interaction with apps. Other obvious changes are that Windows Store apps run in full-screen mode or snapped view, putting content at the front and center while commands and menus stay out of sight until the user requests them.

Windows UI elements such as the minimize, maximize, and close buttons that were once ubiquitous no longer exist in Windows Store apps, as touch swipes and mouse movements have rendered them useless. To close an app you need only swipe or mouse down from the top of the screen to the bottom. Even menus are no longer a fundamental fixture at the top of every screen. In Windows Store apps menus remain tucked away until a touch swipe or mouse gesture reveals them from the bottom in an AppBar as shown in **Figure 1**, using a small countdown timer app as an example.

As you can see in **Figure 1**, menus are out, and commanding elements are graphics-first with some text rather than traditional text-first menus with occasional graphics. These elements are also perfectly finger-sized. If you need more room for options than just the bottom, you can place a nav bar, which is simply an AppBar at the top of the page.

Navigating in traditional Windows menus can be downright painful at times. We've all cringed at a program with a cascading menu that takes you 13 levels deep, causing you to forget what you were looking for in the first place. In Windows Store apps, navigation weaves itself into the content, because touch and mouse gestures on ListView items invoke other pages. Pinch gestures and control key+mouse wheel activate zooming with semantic zoom (bit.ly/16lDtdi), which is both a control and navigation paradigm in Windows Store apps. SemanticZoom is part of the complete listing of WinJS controls (bit.ly/w1jLM5).

Working with HTML and WinJS Controls

There are two primary kinds of controls in Windows Store apps using JavaScript: standard HTML elements and WinJS controls. WinJS controls are HTML combined with prebuilt JavaScript that extends how HTML elements look or behave. Because they're HTML, you can style WinJS controls with CSS. **Figure 2** is an example of a basic WinJS control, the WinJS DatePicker, which is a control comprising multiple DropDown controls representing the day, month and year, showing the default output from this code:

```
<span id="eventDate" data-win-control="WinJS.UI.DatePicker" />
```

Of course, the DatePicker control in **Figure 2** has no styling outside of the default WinJS styles, but you can change that by overriding the .win-datepicker-date, .win-datepicker-month

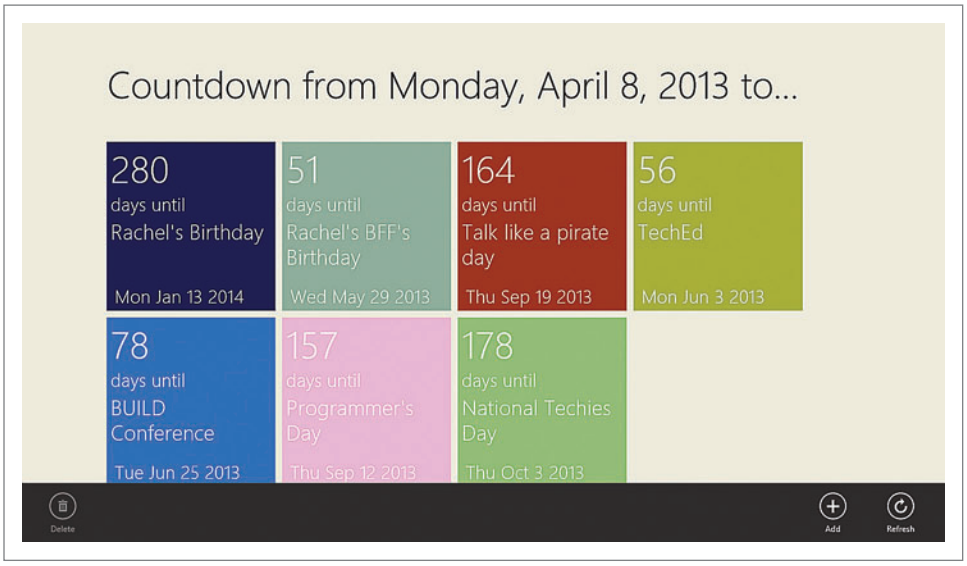


Figure 1 **The AppBar at the Bottom of the App**

Figure 2 **The WinJS DatePicker Control**

and .win-datepicker-year WinJS CSS selectors as well. Use .win-datepicker to style the entire control.

The reason the DatePicker (or any WinJS control) works the way it does is due to the HTML5 data-* attributes, namely data-win-control. The data-win-control attribute signifies the type of control that WinJS will render, in place, so when you set the value of the data-win-control attribute to WinJS.UI.DatePicker, the control renders the dropdowns in **Figure 2**. The data-win-options attribute allows you to set additional properties on controls. For example, on the DatePicker you can set data-win-options for the default display date and minimum and maximum date ranges. Although it's named DatePicker, you can change the control to capture the time instead—for example, hours, minutes and seconds.

Because WinJS builds and renders the final control output, the design-time HTML and the HTML at run time look quite different. **Figure 3** demonstrates the HTML that WinJS injects into the host element at run time. This is viewable from the DOM Explorer (Debug | Windows | DOM Explorer).

The code backing WinJS controls such as the DatePicker lives in a file located at <ProjectRoot>\References\Windows Library for JavaScript 1.0\js\ui.js, alongside some core WinJS friends. Notice that this is the same <script> reference as the one required in the <head> element of Windows Store app pages, that is:

```
<script src="//Microsoft.WinJS.1.0/js/ui.js"></script>
```

Modify these files at your own peril, as they're composed of core WinJS code.

Any WinJS control, including the DatePicker, is accessible at run time through a property named winControl. WinJS appends child properties to the winControl property at run time that are specific to the type of WinJS control. As an example, a ListView contains its list of items, or you can query a WinJS.UI.Ratings control for the user-selected rating. You can access the element's winControl property like so:

```
var control = document.getElementById("WinJSElementId").winControl
```

Buttons,CheckBoxes, RadioButtons, DropDowns, TextBoxes and the like all work exactly the way they do in any plain old HTML page; however, the WinJS.UI namespace is full of UI controls for many complex scenarios, including the ever-important list controls.

List and Grid Controls

Many types of apps need to present data in a grid or in a list, so of course there's a control for those scenarios called the ListView, which can render itself as a grid or a list, complete with grouping and variable item sizes. The ListView is not only highly flexible, but it works perfectly in the new Windows experience by doing things such as automatically scaling to fit the screen and placing list items in varying-sized rows and columns depending on resolution and device-display size.

While most other WinJS controls stand alone, ListViews work in tandem with corresponding HTML as a template. This means you need to set up both template HTML and the control container itself, as shown in **Figure 4**. Notice the template's data-win-control

Figure 3 **The DatePicker Renders Three DropDowns Filled with Options for Date/Month/Year**

```
<span class="win-datepicker" id="eventDate" role="group"
  lang="en-US" dir="ltr" data-win-control="WinJS.UI.DatePicker">
<select tabindex="0" class="win-datepicker-month win-order0"
  aria-label="Select Month">
<option value="January">January</option>
<option value="February">February</option>
<option value="March">March</option>
<option value="April">April</option>
<!-- more <options> that show the other months -->
</select>
<select tabindex="0" class="win-datepicker-date win-order1"
  aria-label="Select Day">
<option value="1">1</option>
<option value="2">2</option>
<option value="3">3</option>
<option value="4">4</option>
<!-- more <options> that  show day numbers -->
</select>
<select tabindex="0" class="win-datepicker-year win-order2"
  aria-label="Select Year">
<option value="1913">1913</option>
<option value="1914">1914</option>
<option value="1915">1915</option>
<option value="1916">1916</option>
<!—more <options> that show years -->
<option value="2112">2112</option>
<option value="2113">2113</option>
</select>
```

and the ListView's data-win-options attributes contain settings that link the ListView and its template together.

Figure 4 contains two elements, one for the template with an id of listViewTemplate and the ListView itself, named listView. The listViewTemplate element contains child elements that represent different fields for each item in the list or grid, such as the eventTitle or eventDate. Looking at the ListView in **Figure 4** reveals the itemDataSource property set to Data.items.dataSource, meaning that Data is a namespace and items is a WinJS.Binding.List object populated with data. Because JavaScript works with loosely typed data, all you need to do is stuff an array of objects into the List constructor and it's then ready to bind to ListView controls, similar to this code:

```
var items = [
  { eventTitle: "Rachel's Birthday", eventDate: new Date(2014, 1, 13) },
  { eventTitle: "Rachel's BFF's Birthday", eventDate: new Date(2013, 5, 29) }
];
var list = new WinJS.UI.list(events);
```

Alternatively, you can use the push method to push items onto the List object rather than passing an array to the List's constructor. The best way to manage data in a ListView is by exposing relevant options (add, delete and so on) via an AppBar control.

Figure 4 **The HTML Required to Create a WinJS ListView**

```
<div id="maincontent">
  <div id="listViewTemplate" data-win-control="WinJS.Binding.Template"  >
    <div data-win-bind="style.background: color" class="win-item">
      <h1 data-win-bind=" innerText: daysToGo"></h1>
      <h2 class="subtitle" data-win-bind="innerText: eventTitle"></h2><br />
      <h2 class="subtitle-bottom" data-win-bind=" innerText: eventDate"></h2>
    </div>
  </div>
  <div id="listView" data-win-control="WinJS.UI.ListView" class="win-listview"
    data-win-options="{ itemDataSource: Data.items.dataSource,
    itemTemplate: select('#listViewTemplate'),
    selectionMode: 'single'}">
  </div>
</div>
```

Figure 5 **Building an AppBar**

```
<!-- HTML -->
<div id="appbar" class="win-appbar" data-win-control="WinJS.UI.AppBar">
<button data-win-control="WinJS.UI.AppBarCommand"
  data-win-options="{id:'deleteButton', label:'Delete',
  icon:'delete', section:'selection'}" type="button"></button>
<button data-win-control="WinJS.UI.AppBarCommand"
  data-win-options="{id:'addButton', label:'Add',
  icon:'add', section:'global'}" type="button"></button>
<button data-win-control="WinJS.UI.AppBarCommand"
  data-win-options="{id:'refreshButton', label:'Refresh',
  icon:'refresh', section:'global'}" type="button"></button>
</div>
// JavaScript
document.getElementById("addButton").addEventListener(
  "click", this.addButtonClick);
document.getElementById("deleteButton").addEventListener(
  "click", this.deleteButtonClick);
document.getElementById("refreshButton").addEventListener(
  "click", this.refreshButtonClick);
```

Figure 6 **A Light-Dismiss Flyout to Collect Information with WinJS Controls**

```
<!-- HTML  -->
<div id="eventFlyoutPanel" data-win-control="WinJS.UI.Flyout">
  <table width="100%" height="100%">
    <tr><td>Event Title:</td><td><input type="text" id="eventTitle"/></td></tr>
    <tr><td>Event Date:</td><td id="eventDate"
      data-win-control="WinJS.UI.DatePicker"></td></tr>
    <tr><td> </td><td align="right">
    <input type="button" id="confirmButton" value="Submit" /></td></tr>
    </table>
</div>
// JavaScript
addButtonClick: function () {
  document.getElementById("eventFlyoutPanel").winControl.show(
  "addButton", "top");
}
```

AppBars and Commands

Content over chrome is an important Microsoft design principle. AppBars are an integral part of this design principle, as they stay out of sight, waiting to present their options when you need them. In code, an AppBar is just a that contains one or more <button> elements, named "app bar commands," with their data-win-control attributes set to WinJS.UI.AppBarCommand. The distinguishing feature between the individual AppBar commands lies in the data-win-options, as you might have guessed.

Inspecting the data-win-options in **Figure 5** for each AppBar command reveals the id, label, icon and section for each command. You can assign AppBar buttons to the global section of the AppBar (which displays on the bottom right of the app's screen), or set the section option to "selection" (to display on the bottom left). Setting the section option of AppBar commands to "selection" makes them contextual, for use when a user selects an item in the ListView by swiping or clicking.

In the HTML page's related JavaScript file, attach event listeners to the AppBar buttons just as you would any other HTML element. You need no listeners for the AppBar itself to appear, as it shows and hides itself automatically in response to user commands—although you also can invoke it programmatically. The sample in **Figure 5** shows a complete AppBar with buttons to add, delete and refresh data.

You can write code to show, hide, enable and disable AppBar buttons, as the scenario demands.

Flyouts

Because a touchscreen is a first-class citizen, you may have noticed that when UI elements and dialog boxes show themselves, you can easily send them away simply by tapping or clicking on any part of the screen other than the dialog itself. This notion of implicitly closing the dialog is called a "light dismiss," and it's the default behavior of MessageDialogs and PopupMenus in Windows 8, because a light dismiss is far easier on the user than dealing with close buttons.

Just as with the previous controls, the Flyout uses the data-win-control attribute to specify that it's indeed a WinJS.UI.Flyout control. Children of the Flyout element render inside the Flyout. For example, you might put an HTML form in a Flyout so the user can fill in a title and date for an upcoming event, as the code in **Figure 6** illustrates, producing what you see in **Figure 7**.

Notice that the Flyout in **Figure 7** is just an HTML form. When the user taps or clicks on the Add AppBar command, the Flyout appears, as directed in the addButtonClick function in **Figure 6**. Flyouts appear on screen in relation to other controls, so when you call the winControl.show method, you pass the control anchor element's name as well as where to place the control—that is, next to the anchor control's top or bottom edge.

The user can tap or click anywhere away from the Flyout to make it go away because it's a light-dismiss control. You'll notice a distinct lack of modal dialog boxes in Windows Store apps, which is part of the Microsoft design philosophy. The folks in design circles frown

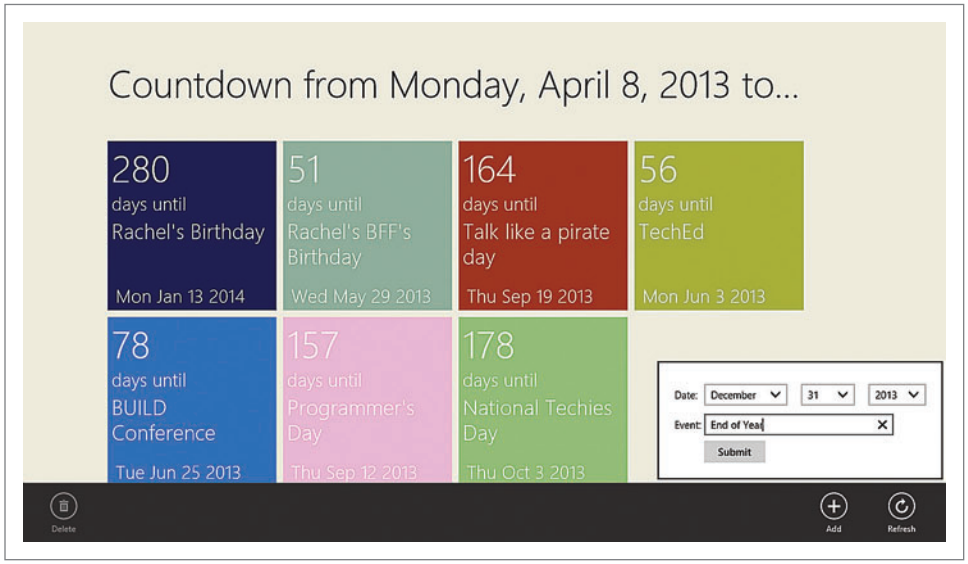


Figure 7 **A Flyout to Collect Information**

upon modal dialog boxes for good reason—anything that annoys a user or blocks him from free movement is considered poor design.

Another type of Flyout is the SettingsFlyout, which is a big shift from the way apps in earlier versions of Windows manage user preferences.

App Settings

Windows users are no strangers to the customary Tools | Options or Help | About menu options that launch dialogs with a labyrinth of settings. Fortunately, what has replaced these dialogs in the land of Windows Store apps is more intuitive for the user. Both settings and about pages operate as tall, vertical Flyout controls, invoked when the user selects the Settings icon from the Windows charms (bit.ly/146cniM).

Settings work consistently across Windows Store apps. When the user invokes the Settings charm, the same SettingsFlyout appears on the right no matter which app is running. Links to your privacy policy, user preferences, help and so on belong in the SettingsFlyout. Creating links to your privacy or options pages takes only a few lines of code in the app.onactivated event normally found in /js/default.js:

```
// In default.js, app.onactivated
WinJS.Application.onsettings = function (e) {
  e.detail.applicationcommands =
  { "privacypolicy": { title: "Privacy Policy", href: "privacy.html" } };
    WinJS.UI.SettingsFlyout.populateSettings(e);
};
```

Once the user taps or clicks on one of the Settings links, its corresponding Flyout appears. **Figure 8** contains the HTML for a SettingsFlyout containing privacy policy information (a clear and concise privacy policy is a requirement for publishing in the Windows Store).

Don't forget to name the privacy policy settings file the same as the href argument used to register the Flyout (see **Figure 8**).

Privacy policies aren't the only things you can put in Settings. SettingsFlyouts can contain any valid HTML and often are host to ToggleSwitches, CheckBoxes, and DropDowns, and function just as the Tools | Options dialogs do now. However, as mentioned, SettingsFlyouts are light-dismiss controls, so a simple tap elsewhere makes them vanish, quite the opposite of modal dialogs. Another simple yet new paradigm in Windows Store app development is the SemanticZoom control, a handy navigation helper.

Semantic Zoom

Some apps are data-intensive. Navigating those apps can be quite difficult, especially when they handle a lot of data. This is where semantic zoom comes to the rescue. Semantic zoom lets you express two modes of data visualization: zoomed in and zoomed out. The zoomed-in mode, the default, shows all the data possible, and the user must pan or scroll through it. The zoomed-out mode usually forms an aggregate representation of data, making it easy for the user to navigate to an area of data and then zoom in to a specific data item.

A SemanticZoom is a set of three controls: the host control and the two zoomed controls, as shown in **Figure 9**. The child controls must implement IZoomable to participate in semantic zoom, so for WinJS apps, the ListView is the only one that will work.

Figure 8 **The HTML for a Settings Flyout Containing Privacy Policy Information**

```
<div id="settingsFlyout" data-win-control="WinJS.UI.SettingsFlyout"
  data-win-options="{settingsCommandId:'privacypolicy', width:'narrow'}">
  <div class="win-header" style="background-color:#312e2e">
    <button type="button" onclick="WinJS.UI.SettingsFlyout.show()"
      class="win-backbutton"></button>
    <div class="win-label">Privacy Policy</div>
  </div>
  <div class="win-content">
    <div class="win-settings-section">
      <p>This application does not collect any personal information.</p>
      <p>Internet access is only used to retrieve data from the web,
        or to allow you to contact the developer:</p>
          <p>
            <a href="mailto:rachel@rachelappel.com">Email Rachel Appel </a>
 <br />
            <a href="http://rachelappel.com/privacy-policy"
            target="_blank">View privacy statement online</a>
          </p>
      </div>
  </div>
</div>
```

Figure 9 **Code for the SemanticZoom Control**

```
<div id="semanticZoomDiv" data-win-control="WinJS.UI.SemanticZoom">
  <!-- The zoomed-in view. -->
  <div id="zoomedInListView"
    data-win-control="WinJS.UI.ListView"
    data-win-options="{ itemDataSource:
      myData.groupedItemsList.dataSource,
    itemTemplate: select('#mediumListIconTextTemplate'),
    groupHeaderTemplate: select('#headerTemplate'), groupDataSource:
    myData.groupedItemsList.groups.dataSource, selectionMode: 'none',
    tapBehavior: 'none', swipeBehavior: 'none' }">
  </div>
  <!-- The zoomed-out view. -->
  <div id="zoomedOutListView"
    data-win-control="WinJS.UI.ListView"
    data-win-options="{ itemDataSource:
    myData.groupedItemsList.groups.dataSource, itemTemplate:
    select('#semanticZoomTemplate'), selectionMode: 'none',
    tapBehavior: 'invoke', swipeBehavior: 'none' }">
  </div>
</div>
```

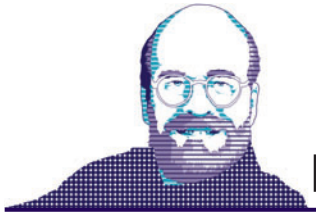
As you can see, semantic zoom is just flipping between two ListViews, making it a great alternative mode of transportation for users—and easy to implement for developers.

More Controls

There are more controls—such as the Progress bar, FlipView, pop-up menus, MessageDialog and Ratings—that are all part of the new Windows experience, but I don't have room here to discuss them all. Open standards such as HTML5 and ECMAScript 5 (ES5) are the base for all things WinJS, so everything from Web staples to anchors and inputs to HTML5 audio and video all work nicely as part of the Windows Store app development platform. ■

Rachel Appel *is a consultant, author, mentor and former Microsoft employee with more than 20 years of experience in the IT industry. She speaks at top industry conferences such as Visual Studio Live!, DevConnections, MIX and more. Her expertise lies within developing solutions that align business and technology focusing on the Microsoft dev stack and open Web. For more about Appel, visit her Web site at rachelappel.com.*

Thanks to the following technical expert for reviewing this article: *Keith Boyd (Microsoft)*



CHARLES PETZOLD

Simulating an Analog Synthesizer

About 50 years ago, a physicist and engineer named Robert Moog created an electronic music synthesizer with a rather unusual feature: an organ-type keyboard. Some composers of electronic music disparaged such a prosaic and old-fashioned control device, while other composers—and particularly performers—welcomed this development. By the end of the 1960s, Wendy Carlos' Switched-On Bach had become one of the best-selling classical albums of all time, and the Moog synthesizer had entered the mainstream.

The early Moog synthesizers were modular and programmed with patch cables. In 1970, however, the Minimoog was released—small, easy to use and play, and priced at just $1,495. (A good history of these early synthesizers is the book, "Analog Days: The Invention and Impact of the Moog Synthesizer" [Harvard University Press, 2004], by Trevor Pinch and Frank Trocco.)

We classify the Moog and similar synthesizers as "analog" devices because they create sounds using varying voltages generated from circuitry built from transistors, resistors and capacitors. In contrast, more modern "digital" synthesizers create sound through algorithmic computations or digitized samples. Older devices are further classified as "subtractive" synthesizers: Rather than building a composite sound through the combination of sine waves (a technique called additive synthesis), subtractive synthesizers begin with a waveform rich in harmonics—such as a sawtooth or square wave—and then run it through filters to eliminate some harmonics and alter the timbre of the sound.

A crucial concept pioneered by Robert Moog was "voltage control." Consider an oscillator, which is the component of a synthesizer that generates a basic audio waveform of some sort. In earlier synthesizers, the frequency of this waveform might be controlled by the value of a resistor somewhere in the circuitry, and that variable resistor might be controlled by a dial. But in a voltage-controlled oscillator (VCO), the frequency of the oscillator is governed by an input voltage. For example, each increase of one volt to the oscillator might double the oscillator's frequency. In this way, the frequency of the VCO can be controlled by a keyboard that generates a voltage that increases by one volt per octave.

In analog synthesizers, the output from one or more VCOs goes into a voltage-controlled filter (VCF) for altering the harmonic content of the waveform. Input voltages to the VCF control the filter's cutoff frequency, or the sharpness of the filter's response (the filter's quality, or Q). The output from the VCF then goes into a voltage-controlled amplifier (VCA), the gain of which is controlled by another voltage.

Code download available at archive.msdn.microsoft.com/mag201307DXF.

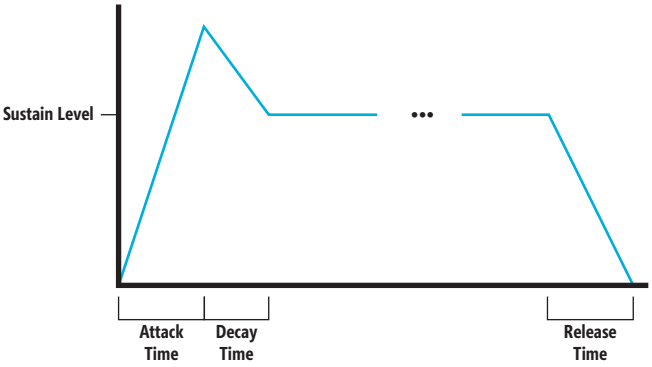


Figure 1 **An Attack-Decay-Sustain-Release Envelope**

Envelope Generators

But once you start talking about VCFs and VCAs, things get complicated, and a little background is necessary.

Back in the 19th century, some scientists (most notably Hermann von Helmholtz) began making significant inroads into the exploration of both the physics and perception of sound. Characteristics of sound such as frequency and loudness turned out to be relatively simple compared with the knotty problem of timbre—that quality of sound that allows us to distinguish a piano from a violin or trombone. It was hypothesized (and somewhat demonstrated) that timbre was related to the sound's harmonic content, which is the various degrees of intensity of the sine curves that constitute the sound.

But when 20th century researchers began investigating further, they discovered it wasn't this simple. Harmonic content changes over the course of a musical tone, and this contributes to the instrument's timbre. In particular, the very beginning of a note from a musical instrument is crucial to auditory perception. When a piano hammer or violin bow first touches a string, or vibrating air is propelled into a metal or wooden tube, very complex harmonic activity occurs. This complexity decreases very quickly, but without it, musical tones sound dull and far less interesting and distinctive.

To mimic the complexity of real musical tones, a synthesizer can't simply turn a note on and off like a switch. (To hear what such a simple synthesizer sounds like, check out the ChromaticButtonKeyboard program in the February 2013 installment of this column at msdn.microsoft.com/magazine/jj891059.) At the onset of each note, the sound must have a brief "blip" of high volume and varying timbre before stabilizing. When the note ends, the sound must not simply stop, but die out with a decrease in volume and complexity.

For loudness, there's a general pattern to this process: For a note played on a string, brass or woodwind instrument, the sound rises to a maximum loudness quickly, then dies off a bit and holds

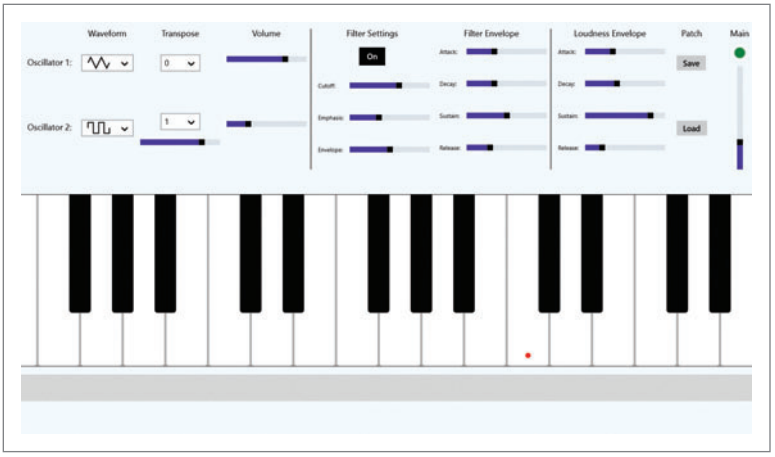


Figure 2 **The AnalogSynth Screen**

steady. When the note ends, it rapidly decreases in volume. These two phases are known as the "attack" and "release."

For more percussive instruments—including the piano—the note reaches maximum volume quickly during the attack but then dies off slowly if the instrument remains undamped, for example, while holding the piano key down. Once the key is released, the note quickly dies off.

To achieve these effects, synthesizers implement something called an "envelope generator." **Figure 1** shows a fairly standard example called an attack-decay-sustain-release (ADSR) envelope. The horizontal axis is time, and the vertical axis is loudness.

When a key on a keyboard is pressed and the note begins to sound, you hear the attack and decay sections that give a burst of sound at the outset, and then the note stabilizes at the sustain level. When the key is released and the note ends, the release section occurs. For a piano-type sound, the decay time could be a couple seconds, and the sustain level is set at zero so the sound continues to decay as long as the key is held down.

Even the simplest analog synthesizers have two ADSR envelopes: One controls the volume and the other controls the filter. This is usually a low-pass filter. As a note is struck, the cutoff frequency is rapidly increased to allow more high-frequency harmonics through, and then the cutoff frequency decreases somewhat. Emphasized a lot, this creates the distinctive analog synthesizer chirping sound.

The AnalogSynth Project

Some nine months ago, as I was contemplating using XAudio2 to program a digital simulation of a small 1970s-era analog synthesizer, I realized that the envelope generators would be one of the more challenging aspects of the job. It wasn't even clear to me whether these envelope generators would be external to the audio-processing stream (and therefore access the SetVolume and SetFilterParameters methods of an XAudio2 voice), or somehow be built in to the audio stream.

I eventually settled on implementing the envelopes as XAudio2 audio effects—more formally known as Audio Processing Objects (APOs). This means the envelope logic works directly on the audio stream. I became more confident with this approach after coding filter logic that duplicates the digital biquad filters built into XAudio2. By using my own filter code, I thought I might be able to change the filter algorithm in the future without major disruptions to the program structure.

Figure 2 shows the screen of the resultant AnalogSynth program, whose source code you can download at archive.msdn.microsoft.com/mag201307DXF. Although I was influenced by the layout of the controls on the Minimoog, I kept the actual UI rather simple, using, for example, sliders rather than dials. Most of my focus was on the internals.

The keyboard is a series of custom Key controls processing Pointer events and grouped into Octave controls. The keyboard is actually six octaves in width and can be scrolled horizontally using the thick gray stripe below the keys. A red dot identifies middle C.

The program can play 10 simultaneous notes, but that's changeable with a simple #define in MainPage.xaml.cs. (Early analog synthesizers like the Minimoog were monophonic.) Each of these 10 voices is an instance of a class I called SynthVoice. SynthVoice has methods to set all the various parameters of the voice (including frequency, volume and envelopes), as well as methods named Trigger and Release to indicate when a key has been pressed or released.

The Minimoog achieved its characteristic "punchy" sound in part by having two oscillators running in parallel and often slightly mistuned, either deliberately or as a result of the frequency drift common in analog circuitry.

For that reason, each SynthVoice creates two instances of an Oscillator class, which are controlled from the upper-left of the control panel shown in **Figure 2**. The control panel lets you set the waveform and relative volume for these two oscillators, and you can transpose the frequency by one or two octaves up or down. In addition, you can offset the frequency of the second oscillator by up to half an octave.

Each Oscillator instance creates an IXAudio2SourceVoice object, and exposes methods named SetFrequency, SetAmplitude and SetWaveform. SynthVoice routes the two IXAudio2SourceVoice outputs to an IXAudio2SubmixVoice, and then instantiates two custom audio effects called FilterEnvelopeEffect and AmplitudeEnvelopeEffect that it applies to this submix voice. These two effects share a class called EnvelopeGenerator that I'll describe shortly.

Figure 3 shows the organization of components in each SynthVoice. For the 10 SynthVoice objects, there are a total of 20 IXAudio2SourceVoice instances going into 10 IXAudio2SubmixVoice instances, which are then routed to a single IXAudio2MasteringVoice. I use a sampling rate of 48,000 Hz and 32-bit floating-point samples throughout.

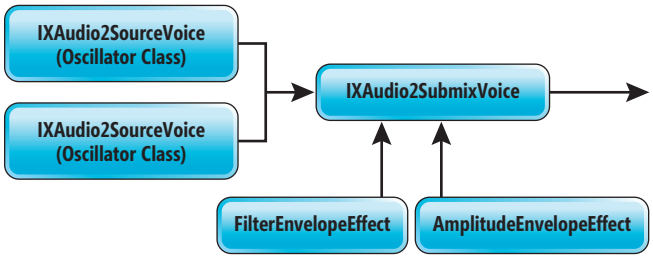


Figure 3 **The Structure of the SynthVoice Class**

Figure 4 **The EnvelopeGenerator Header File**

```
class EnvelopeGenerator
{
private:
  enum class State
  {
    Dormant, Attack, Decay, Sustain, Release
  };

  EnvelopeGeneratorParameters params;
  float level;
  State state;
  bool isReleased;
  float releaseRate;

public:
  EnvelopeGenerator();
  void SetParameters(const EnvelopeGeneratorParameters params);
  void Attack();
  void Release();
  bool GetNextValue(float interval, float& value);
};
```

The user controls the filter from the center section of the control panel. A ToggleButton allows the filter to be bypassed; otherwise, the Cutoff frequency is relative to the note that's being played. (In other words, the cutoff frequency of the filter tracks the keyboard.) The Emphasis slider controls the filter's Q setting. The Envelope slider controls the degree to which the envelope affects the filter cutoff frequency.

The four sliders associated with the filter envelope and the loudness envelope work similarly. The Attack, Decay and Release sliders are all durations from 10 milliseconds to 10 seconds in a logarithmic scale. The sliders have tool-tip value converters to display the duration associated with the settings.

AnalogSynth makes no volume adjustments for the 20 potential simultaneous IXAudio2SourceVoice instances, or to counteract the tendency of digital biquad filters to amplify audio near the cutoff frequency. Consequently, AnalogSynth makes it easy to overload the audio. To help the user avoid this, the program uses the XAudio2CreateVolumeMeter function to create an audio effect that monitors the outgoing sound. If the green dot in the upper-right corner changes to red, output audio is being clipped and you should use the slider at the far right to decrease the volume.

Early synthesizers used patch cords to connect components. As a result of this legacy, a particular synthesizer setup is still known as a "patch." If you find a patch that makes a sound you want to keep, press the Save button and assign a name. Press the Load button to get a list of previously saved patches and select one. These patches (as well as the current setup) are stored in the local settings area.

The Envelope Generator Algorithm

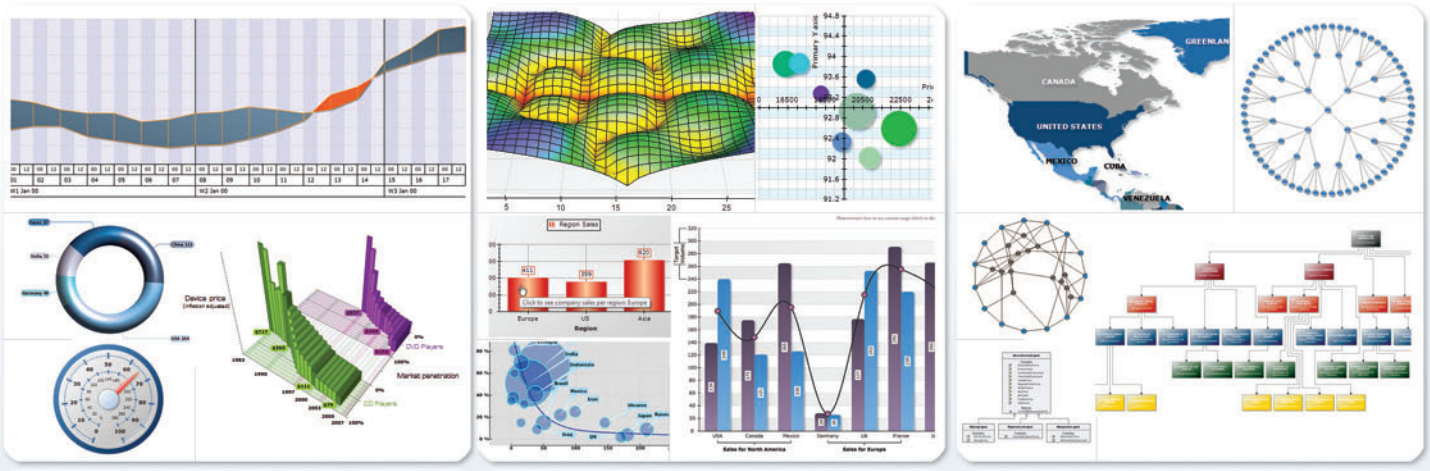
Code that implements an envelope generator is basically a state machine, with five sequential states that I called Dormant, Attack, Decay, Sustain and Release. From a UI perspective, it seems most natural to specify attack, decay, and sustain in terms of time durations, but when actually performing the calculations you need

Figure 5 **The Implementation of EnvelopeGenerator**

```
EnvelopeGenerator::EnvelopeGenerator() : state(State::Dormant)
{
  params.baseLevel = 0;
}

void EnvelopeGenerator::SetParameters(const EnvelopeGeneratorParameters params)
{
  this->params = params;
}

void EnvelopeGenerator::Attack()
{
  state = State::Attack;
  level = params.baseLevel;
  isReleased = false;
}

void EnvelopeGenerator::Release()
{
  isReleased = true;
}

bool EnvelopeGenerator::GetNextValue(float interval, float& value)
{
  bool completed = false;

  // If note is released, go directly to Release state,
  // except if still attacking
  if (isReleased &&
    (state == State::Decay || state == State::Sustain))
  {
    state = State::Release;
    releaseRate = (params.baseLevel - level) / params.releaseTime;
  }

  switch (state)
  {
  case State::Dormant:
    level = params.baseLevel;
    completed = true;
    break;

  case State::Attack:
    level += interval * params.attackRate;

    if ((params.attackRate > 0 && level >= params.peakLevel) ||
      (params.attackRate < 0 && level <= params.peakLevel))
    {
      level = params.peakLevel;
      state = State::Decay;
    }
    break;

  case State::Decay:
    level += interval * params.decayRate;

    if ((params.decayRate > 0 && level >= params.sustainLevel) ||
      (params.decayRate < 0 && level <= params.sustainLevel))
    {
      level = params.sustainLevel;
      state = State::Sustain;
    }
    break;

  case State::Sustain:
    break;

  case State::Release:
    level += interval * releaseRate;

    if ((releaseRate > 0 && level >= params.baseLevel) ||
      (releaseRate < 0 && level <= params.baseLevel))
    {
      level = params.baseLevel;
      state = State::Dormant;
      completed = true;
    }
  }

  value = level;
  return completed;
}
```

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.

to convert that to a rate—an increase or decrease of loudness (or filter cutoff frequency) per unit time. The two audio effects in AnalogSynth use these changing levels to implement the effect.

This state machine is not always as sequential as the diagram in **Figure 1** would seem to imply. For example, what happens when a key is pressed and released so quickly that the envelope has not yet reached the sustain section when the key is released? At first I thought the envelope should be allowed to complete its attack and decay sections and then go right into the release section, but this did not work well for a piano-type envelope. In a piano envelope, the sustain level is zero and the decay time is relatively long. A key quickly pressed and released still had a long decay—as if it were not released at all!

I decided that for a quick press and release, I would let the attack section complete, but then immediately jump to the release section. This meant that the final rate of decrease would need to be calculated based on the current level. This explains why there's a difference in how the release is handled in the structure for the envelope parameters, shown here:

```
struct EnvelopeGeneratorParameters
{
  float baseLevel;
  float attackRate;   // in level/msec
  float peakLevel;
  float decayRate;    // in level/msec
  float sustainLevel;
  float releaseTime;  // in msec
};
```

For the amplitude envelope, baseLevel is set to 0, peakLevel is set to 1 and sustainLevel is somewhere between those values. For the filter envelope, the three levels refer to a multiplier applied to the filter cutoff frequency: baseLevel is 1, and peakLevel is governed by the slider labeled "Envelope" and can range from 1 to 16. That frequency multiplier of 16 corresponds to four octaves.

Both AmplitudeEnvelopeEffect and FilterEnvelopeEffect share the EnvelopeGenerator class. **Figure 4** shows the EnvelopeGenerator header file. Notice the public method to set the envelope parameters, and two public methods named Attack and Release that trigger the envelope to begin and finish up. These three methods should be called in that order. The code is not written to deal with an envelope whose parameters change midway through its progress.

The current calculated value from the envelope generator is obtained through repeated calls to GetNextValue. The interval argument is in milliseconds, and the method computes a new value based on that interval, possibly switching states in the process. When the envelope has completed with the Released section, GetNextValue returns true to indicate that the envelope has completed, but I don't actually use that return value elsewhere in the program.

Figure 5 shows the implementation of the EnvelopeGenerator class. Near the top of the GetNextValue method is the code to skip directly to the Release state when a key is released, and the calculation of a release rate based on the current level and the release time.

A Pair of Audio Effects

Both the AmplitudeEnvelopeEffect and FilterEnvelopeEffect classes derive from CXAPOParametersBase so they can accept parameters, and both classes also maintain an instance of the EnvelopeGenerator class for performing the envelope calculations. The parameter

Figure 6 **The Process Override in AmplitudeEnvelopeEffect**

```
void AmplitudeEnvelopeEffect::Process(UINT32 inpParamCount,
                                      const XAPO_PROCESS_BUFFER_PARAMETERS *pInpParam,
                                      UINT32 outParamCount,
                                      XAPO_PROCESS_BUFFER_PARAMETERS *pOutParam,
                                      BOOL isEnabled)
{
  // Get effect parameters
  AmplitudeEnvelopeParameters * pParams =
    reinterpret_cast<AmplitudeEnvelopeParameters *>
      (CXAPOParametersBase::BeginProcess());

  // Get buffer pointers and other information
  const float * pSrc = static_cast<float const*>(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:
    if (!isEnabled)
    {
      for (int frame = 0; frame < frameCount; frame++)
      {
        for (int channel = 0; channel < numChannels; channel++)
        {
          int index = numChannels * frame + channel;
          pDst[index] = pSrc[index];
        }
      }
    }
    else
    {
      // Key being pressed
      if (!this->keyPressed && pParams->keyPressed)
      {
        this->keyPressed = true;
        this->envelopeGenerator.SetParameters(pParams->envelopeParams);
        this->envelopeGenerator.Attack();
      }
      // Key being released
      else if (this->keyPressed && !pParams->keyPressed)
      {
        this->keyPressed = false;
        this->envelopeGenerator.Release();
      }

      // Calculate interval in msec
      float interval = 1000.0f / waveFormat.nSamplesPerSec;

      for (int frame = 0; frame < frameCount; frame++)
      {
        float volume;
        envelopeGenerator.GetNextValue(interval, volume);

        for (int channel = 0; channel < numChannels; channel++)
        {
          int index = numChannels * frame + channel;
          pDst[index] = volume * pSrc[index];
        }
      }
    }
    break;

  case XAPO_BUFFER_SILENT:
    break;
  }

  // Set output parameters
  pOutParam[0].ValidFrameCount = pInpParam[0].ValidFrameCount;
  pOutParam[0].BufferFlags = pInpParam[0].BufferFlags;

  CXAPOParametersBase::EndProcess();
}
```

structures for these two audio effects are named AmplitudeEnvelopeParameters and FilterEnvelopeParameters.

The AmplitudeEnvelopeParameters structure is merely an EnvelopeGeneratorParameters structure and a Boolean keyPressed field that's true when the key associated with this voice is pressed and false when it's released. (The FilterEnvelopeParameters structure is just a bit more complex because it needs to incorporate a base-level filter cutoff frequency and Q setting.) Both effects classes maintain their own keyPressed data members that can be compared with the parameters value to determine when the envelope attack or release state should be triggered.

You can see how this works in **Figure 6**, which shows the code for the Process override in AmplitudeEnvelopeEffect. If the effect is enabled and the local keyPressed value is false but the keyPressed value in the effect parameters is true, then the effect makes calls to the SetParameters and Attack methods of the EnvelopeGenerator instance. If the opposite is the case—the local keyPressed value is true but the one in the parameters is false—then the effect calls the Release method.

The effect could call the GetNextValue method of EnvelopeGenerator either for every Process call (in which case the interval argument would indicate 10 milliseconds) or for every sample (in which case the interval is more like 21 microseconds). Although the first approach should be adequate, I decided on the second for theoretically smoother transitions.

The floating-point volume value returned from the GetNextValue call ranges from 0 (when a note is first beginning or ending) to 1 for the culmination of the attack. The effect simply multiplies the floating-point samples by this number.

Now the Fun Begins

I've spent so much time coding the AnalogSynth program that I haven't had much time to play around with it. It could very well be that some of the controls and parameters need some fine-tuning, or perhaps rather coarser tuning! In particular, long decay and release times on the volume don't sound quite right, and they suggest that the envelope changes to amplitudes should be logarithmic rather than linear.

I'm also intrigued by the use of touch input with the on-screen keyboard. The keys on a real piano are sensitive to the velocity with which they are struck, and synthesizer keyboards have attempted to emulate that same feel. Most touchscreens, however, can't detect touch velocity or pressure. But they can be made sensitive to slight finger movements on the screen, which is beyond the capability of a real keyboard. Can on-screen keyboards be made more responsive in this way? There's only one way to find out! ■

Charles Petzold *is a longtime contributor to* MSDN Magazine *and the author of "Programming Windows, 6th edition" (O'Reilly Media, 2012), a book about writing applications for Windows 8. His Web site is charlespetzold.com.*

Thanks to the following technical expert for reviewing this article: *James McNellis (Microsoft)*

DAVID S. PLATT

Activation Energy

Regular readers of this column know how I love analogies. Here's another good one: You can understand user needs and user acceptance of your UI strategies by applying a concept from chemical engineering—activation energy versus released energy.

Both coal and paper will burn, releasing energy, as shown in **Figure 1**. But you need to supply some energy to start the process. My hot air probably will not ignite the paper magazine in your hand; for that, you need a match. The activation energy for coal is higher. You need to get it hotter, perhaps by starting a small fire with paper and then adding some kindling. On the other hand, the energy released by burning coal is greater than that released by burning paper, which is why we burn coal in power plants.

Your users' operations are also governed by the calculus of activation energy versus released energy. Consider airlines. The ones that are making money (or at least losing it more slowly than the others) are the ones that sell the greatest proportion of their tickets online. The easiest way to do this is a simple browser interface. Just type delta.com into your address bar, and there it is, asking where you want to go and when. That activation energy is about as low as it gets. The released energy isn't bad, although you're subject to a browser interface's foibles, such as not using the back button where you're not supposed to.

Now suppose you had to download and install some sort of add-in before Delta would talk to you. Who would bother? You'd go to Orbitz, or you'd pick a different airline that valued its users' time. The success of this casual e-commerce scenario is very much limited by the height of the activation energy hump. You need a paper fire, because you have to light many new ones every day.

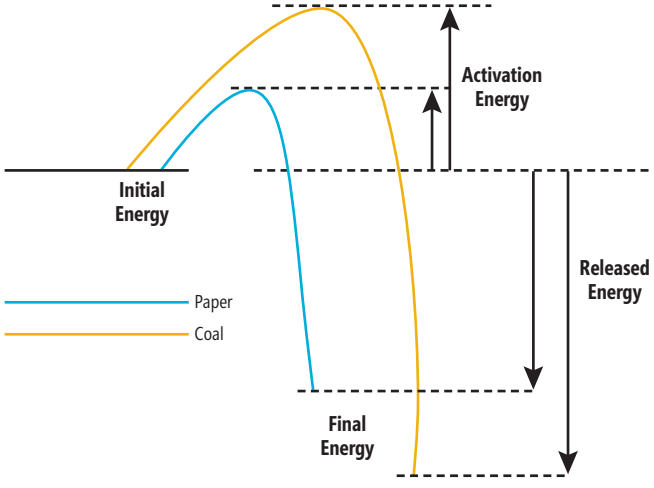


Figure 1 **Higher Initial Energy Input Can Yield Greater Overall Energy Release**

Now suppose you were a travel agent, writing airline tickets for a living: 10 minutes each, six per hour, 48 per day. Suppose that a rich client application could lower this to eight minutes per ticket, improving your productivity from 48 tickets per day to 60. The app would have a higher activation energy: time to download and install, and you might also need some training. But you'd be willing to pay that higher activation energy price (or perhaps tunnel through it—see last month's column on Heisenberg at msdn.microsoft.com/magazine/dn201756) because you harvest that released-energy advantage with each ticket you write. Once you've gotten the coal fire burning, it's easy to keep going.

A mobile app occupies an interesting middle ground. The brutal size constraints of smartphones often render a browser interface unusable, even one that's tailored for mobile. Users are comfortable with the idea of downloading and installing apps. Furthermore, a user's needs on the mobile platform often require more intimate control of the hardware than a browser allows. For example, the Amazon mobile-oriented Web site strives mightily to offer good service. But one of the main things mobile Amazon users do is comparison shop in physical stores before ordering online—a process called "showrooming" that's now killing stores such as Best Buy. The Amazon mobile app lets you find a price by scanning an item's UPC code or even just snapping a picture of the item.

MS-DOS had a high activation energy. You had to learn the basic DOS commands, and then the different commands for each program you ran. Windows was better because you didn't have to memorize as much to get started. Menus showed you what programs could do, and each program worked sort of like the other ones. So the activation energy was lower.

Now comes Windows 8. It's harder to learn because less of it is visible. The activation energy is higher than that of an iPad, for example. But the released energy is higher in Windows 8, with standardization of common application features such as searching and sharing. Microsoft desperately hopes that the promise of greater released energy will cause users to swallow the greater activation energy. The question is, will they? ■

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.*

HTML5 & JavaScript | WPF & Silverlight | Windows Forms | Windows Phone | Windows 8

HTML5

Based on industry standards & built with HTML5, jQuery, & CSS3

Adaptive widgets adapt to jQuery Mobile or jQuery UI environments

Effortless application-wide theming with built-in professionally-designed themes

Full cross-browser compatibility

Touch

Windows Store & Windows Phone components designed with a touch-first UX in mind

Support for selecting, dragging, & multi-touch gestures

Supports for Windows Phone Modern UI design & interaction guidelines specified by Microsoft

Windows Store compliant to speed the app submission process

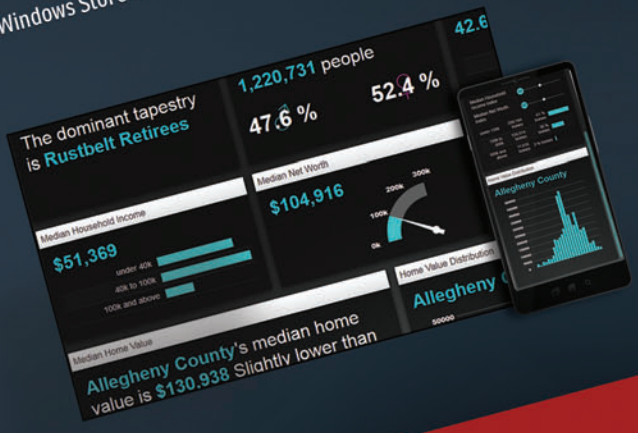
Data Visualization

Cross-platform charts, grids, maps, gauges, & more

Reporting & document-generation with pixel-perfect reporting, PDF, & print-preview controls

Award-winning grids with filtering, hierarchical data-binding, grouping, printing, & exporting to Microsoft Excel

Bring data to life with dynamic dashboards



.NET Tools for the Professional Developer

STUDIO ENTERPRISE

ComponentOne
a division of GrapeCity®

Download your free trial @
www.componentone.com