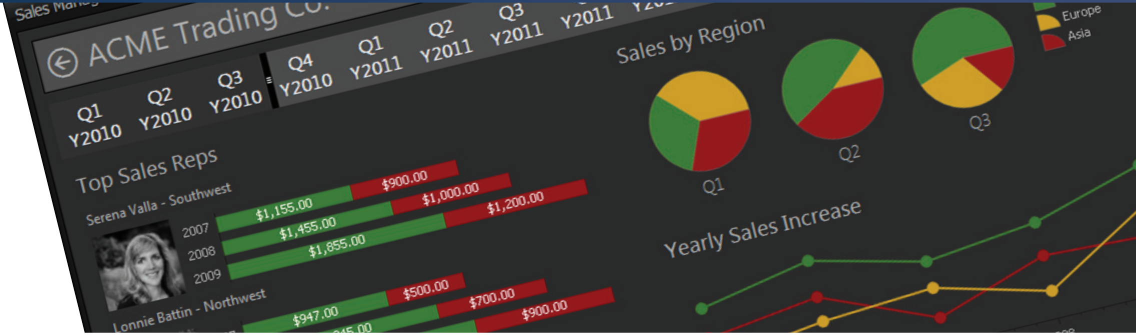

Sales Mana
ACME Trading Co.
Q1 Y2010
Q2 Y2010
Q3 Y2010
Q4 Y2010
Q1 Y2011
Q2 Y2011
Q3 Y2011
Y2011
Top Sales Reps
Serena Valla - Southwest
2007
2008
2009
$1,155.00
$900.00
$1,455.00
$1,000.00
$1,855.00
$1,200.00
Lonnie Battin - Northwest
$947.00
$500.00
$700.00
$900.00
Sales by Region
Q1
Q2
Q3
Europe
Asia
Yearly Sales Increase

THE MICROSOFT JOURNAL FOR DEVELOPERS

FEBRUARY 2012 VOL 27 NO 2

msdn magazine

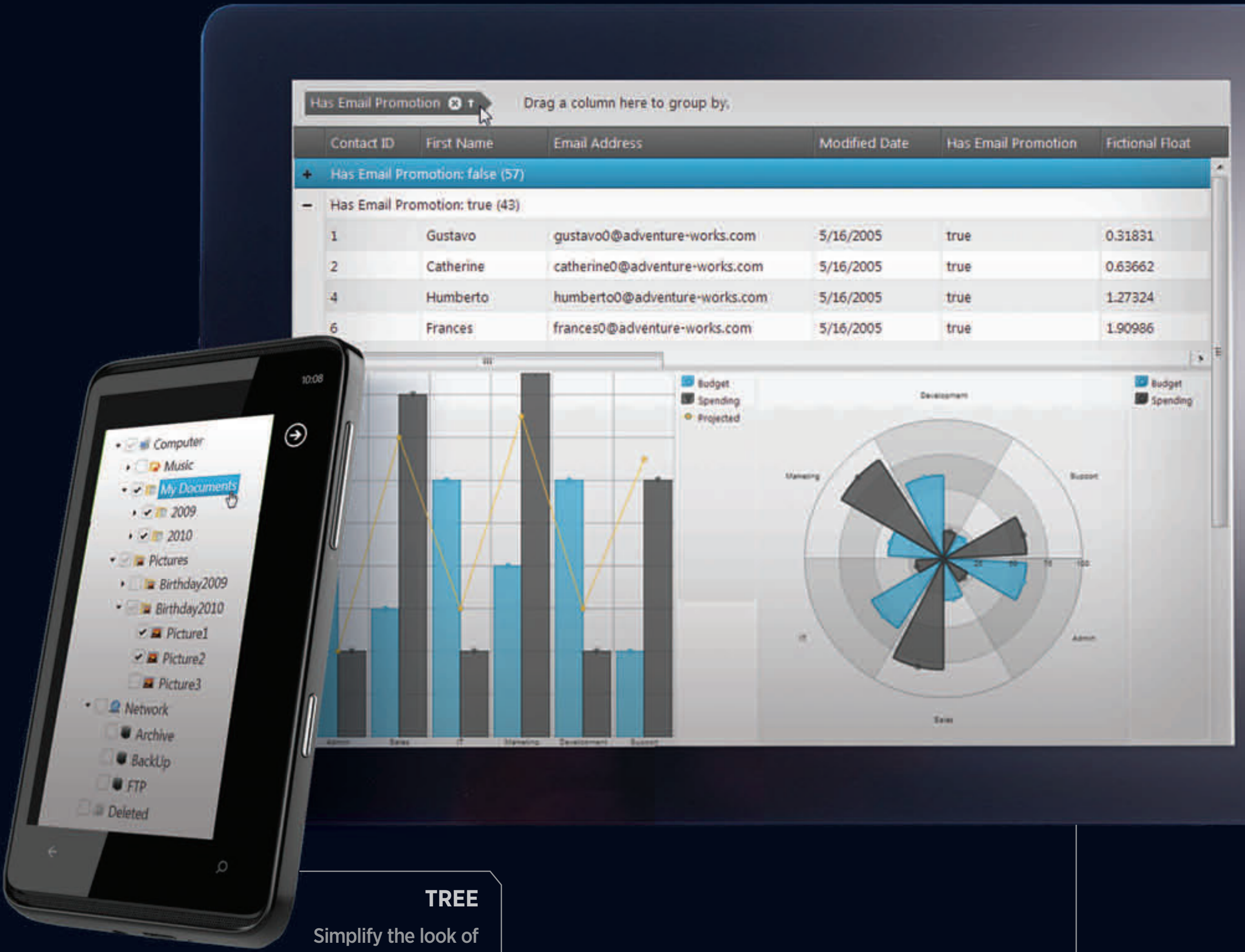
COLUMNS

Microsoft

Write Once, Experience Many

check out infragistics.com/jquery

NetAdvantage® for jQuery



Has Email Promotion ⊗ ↑ Drag a column here to group by.

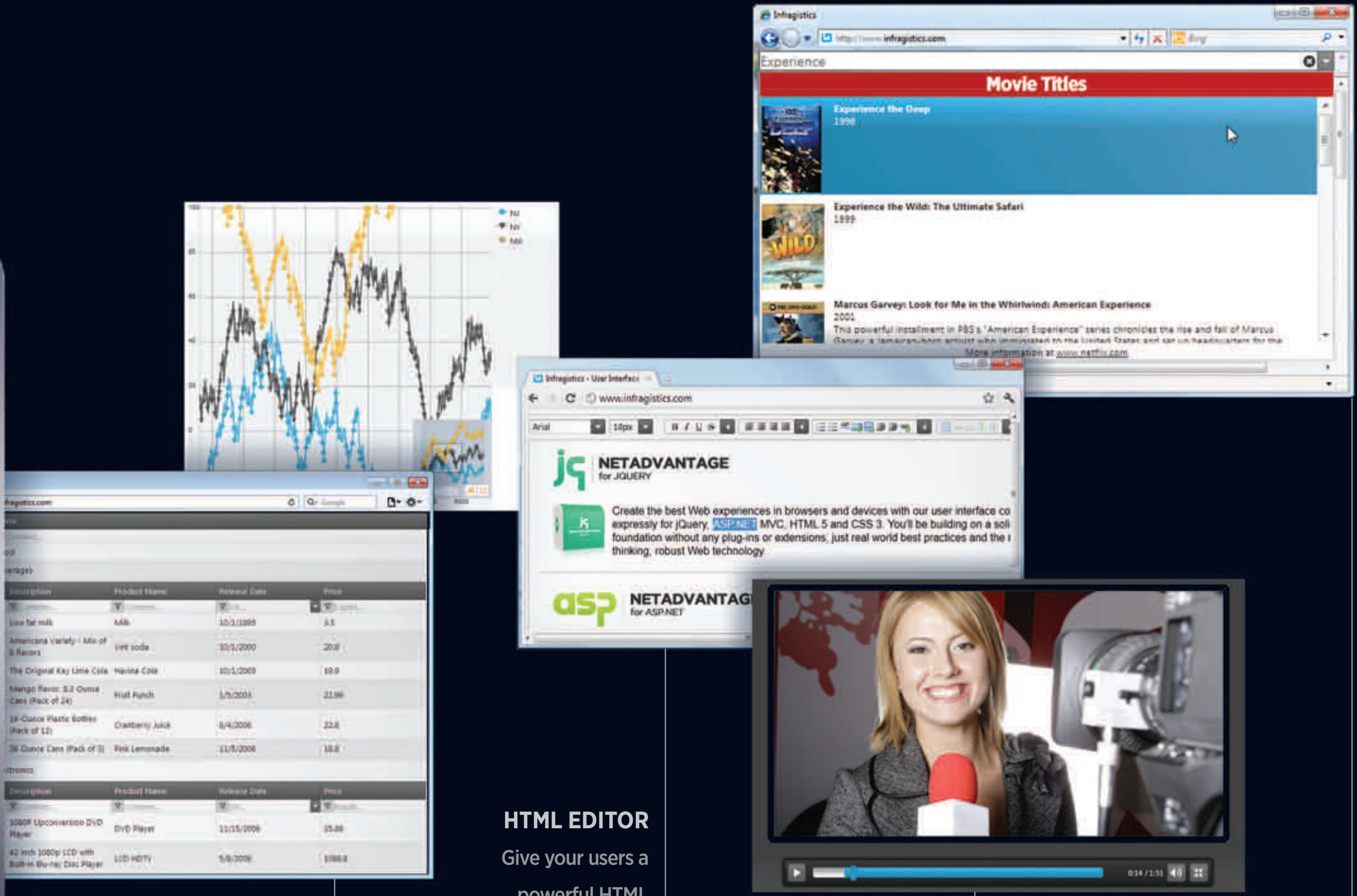
Contact ID	First Name	Email Address	Modified Date	Has Email Promotion	Fictional Float
+ Has Email Promotion: false (57)					
− Has Email Promotion: true (43)					
1	Gustavo	gustavo0@adventure-works.com	5/16/2005	true	0.31831
2	Catherine	catherine0@adventure-works.com	5/16/2005	true	0.63662
4	Humberto	humberto0@adventure-works.com	5/16/2005	true	1.27324
6	Frances	frances0@adventure-works.com	5/16/2005	true	1.90986

TREE
Simplify the look of hierarchical data, while offering the experience, design and functionality your users will love!

BUSINESS CHARTING
Combine interactive Outlook style grids with rich business charting to deliver a complete portable solution.

INFRAGISTICS™
DESIGN / DEVELOP / EXPERIENCE

Infragistics Sales 800 231 8588 • Infragistics Europe Sales +44 (0) 800 298 9055 • Infragistics India +91 80 4151 8042 • @infragistics



Experience

Movie Titles

Experience the Deep
1998

Experience the Wild: The Ultimate Safari
1999

Marcus Garvey: Look for Me in the Whirlwind: American Experience
2001
This powerful installment in PBS's "American Experience" series chronicles the rise and fall of Marcus Garvey, a Jamaican-born activist who immigrated to the United States and set up headquarters for the
More information at www.netflix.com

NETADVANTAGE for JQUERY

Create the best Web experiences in browsers and devices with our user interface co expressly for jQuery, ASP.NET MVC, HTML 5 and CSS 3. You'll be building on a soli foundation without any plug-ins or extensions, just real world best practices and the thinking, robust Web technology

NETADVANTAGE for ASP.NET

Beverages

Description	Product Name	Release Date	Price
Low fat milk	Milk	10/1/1995	3.5
Americana Variety - Mix of 6 flavors	Vint soda	10/1/2000	20.9
The Original Key Lime Cola	Havina Cola	10/1/2005	19.9
Mango flavor, 8.3 Ounce Cans (Pack of 24)	Fruit Punch	1/5/2004	22.99
16-Ounce Plastic Bottles (Pack of 12)	Cranberry Juice	8/4/2006	22.8
36 Ounce Cans (Pack of 3)	Pink Lemonade	11/5/2006	18.8

Electronics

Description	Product Name	Release Date	Price
1080P Upconversion DVD Player	DVD Player	11/15/2006	35.88
42 inch 1080p LCD with Built-in Blu-ray Disc Player	LCD HDTV	5/8/2006	1088.8

HIERARCHICAL GRID

An expandable data grid that presents multiple parent-child relationships is the backbone of your data application.

HTML EDITOR

Give your users a powerful HTML editing experience by incorporating the jQuery WYSIWYG editing tool.

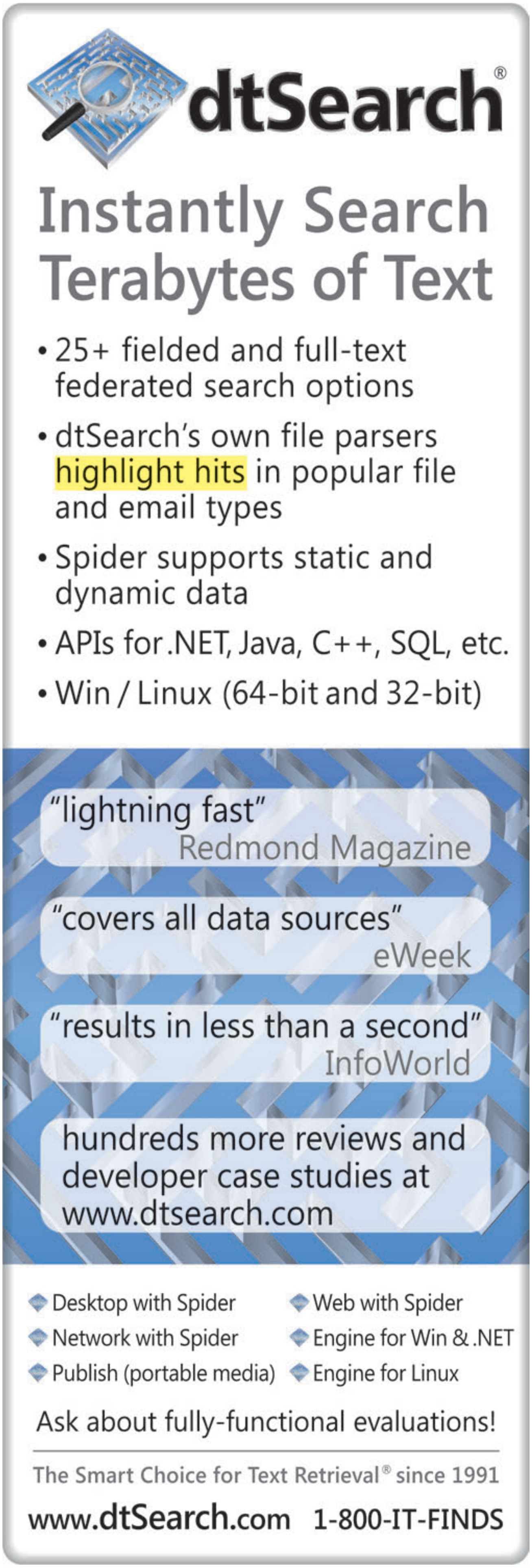
VIDEO PLAYER

When a user finds what they want to watch, our HTML5 video player adds streaming video right into your own apps.

COMBO

The fully featured combo box control offers intuitive auto-suggest, auto-complete and auto-filtering built in.

HTML 5



msdn® magazine

FEBRUARY 2012 VOLUME 27 NUMBER 2

LUCINDA ROWLEY Director
KIT GEORGE Editorial Director/mmeditor@microsoft.com
PATRICK O'NEILL Site Manager

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

SCOTT SHULTZ Creative Director
JOSHUA GOULD Art Director

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

RedmondMediaGroup

Henry Allain President, Redmond Media Group
Matt Morollo Vice President, Publishing
Doug Barney Vice President, Editorial Director
Michele Imgrund Director, Marketing
Tracy Cook Online Marketing Director

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

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

1105 MEDIA INC

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

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

Jeffrey S. Klein Chairman of the Board

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

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

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

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

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

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

List Rental: This publication's subscriber list, as well as other lists from 1105 Media, Inc., is available for rental. For more information, please contact our list manager, Merit Direct. Phone: 914-368-1000; E-mail: 1105media@meritdirect.com; Web: www.meritdirect.com/1105

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

Microsoft®

BPA WORLDWIDE

Printed in the USA

MULTIMEDIA
SDKs
LEADTOOLS
POWERED BY.
FREE 60 DAY EVALUATION!

Editor's Note

MICHAEL DESMOND

Calling out the Client

A month ago in this space, I wrote about an *MSDN Magazine* author and columnist—Charles Petzold—who marked his 25th year as a contributor to this publication. So it's appropriate, I suppose, that I dwell now on another valued columnist who left the magazine after more than a decade, only to return to the fold this month.

John Papa first began writing for *MSDN Magazine* back in December 1998, and for eight years from 2002 through 2009 authored the popular Data Points column, which continues today under the stewardship of Julie Lerman. Papa wrote his last column in February 2009, stepping down when he took on a role with Microsoft as a corporate evangelist for Silverlight and XAML. Now Papa is back, penning a new column called Client Insight, which focuses on the fast-changing arena of rich client development technologies and platforms.

"Client technology has really changed over the past few years with XAML, HTML5 and mobile devices," says Papa, who adds that he hopes to bring together the best parts of XAML and HTML5.

"I plan on exploring the good, the bad and the challenging aspects of client technologies," he continues. "Anyone interested in writing software for the next generation should enjoy the column as I plan to cover HTML5, CSS3, JavaScript, patterns, XAML, mobility, tooling and much more."

For Papa, the new column is a second chance to pursue a passion. The Data Points column got its start in 1998 because he felt at the time that data-related topics were being neglected. What started as a series of data-focused features eventually turned into a monthly column that ran for the better part of a decade.

"It always seemed like data access got very little play time in magazines, conferences and books, so I decided to do something about it," Papa recalls of launching Data Points.

Silverlight Shuffle

Of course, Papa is best known as a leading light in the Rich Internet Application (RIA) space, particularly in the arena of Silverlight development. A longtime independent developer, Papa became a fixture on the conference circuit, often appearing alongside Microsoft keynoters at events like MIX and the Microsoft Professional Developers Conference. He eventually joined Microsoft to help evangelize the company's RIA efforts.

"When I got into Silverlight I saw a great opportunity to work with patterns, data and client-side RIA technology," Papa says. "The opportunity I had to be the corporate evangelist for Silverlight/XAML was awesome. They gave me the freedom to run and try new ideas like Silverlight TV, running communities, MIXer parties, open source events, the Silverlight Firestarter and much more."

> "We need to use the right tool for the right job. A lot has changed just in the past two years, but that rule hasn't."

So why the Client Insight column, and why now?

For one thing, Papa left Microsoft in November to move back to his native Florida, where he had lived prior to relocating to the Redmond area in 2009. More to the point, the emergence of JavaScript/HTML5, alongside the evolving Microsoft XAML/Silverlight strategy, has turned the client development space on its ear. Developers are actively rethinking their positions as they weigh up-and-coming platforms and technologies against the challenges posed by an increasingly diverse client space. Papa says developers must keep things in perspective.

"We need to use the right tool for the right job. A lot has changed just in the past two years, but that rule hasn't," he says, noting that developers need to stay current even as they work with their existing tools.

"The best perspectives I've heard are from developers who have researched many alternatives. This prepares them to be armed for in-depth discussions on what technology to choose in what situation," he says. "Keep an open mind and stay in touch with the evolution."

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

Windows
iPhone
Android
Web
iPad

DATA POINTS

JULIE LERMAN

A Few of My Favorite Things ... in the Entity Framework 4.2 DbContext

Even before Entity Framework 4.1 was released in early 2011, developers were focused on only half of what was given to us in that package: Code First. Code First lets you express your Entity Data Model using your domain classes and Code First configurations, a great alternative to developers who don't want to use the visual designer to define the model. But every bit of sample code that you see for using Entity Framework (EF) with those classes and Code First-defined models is driven by another very important feature that came in EF 4.1: the DbContext class.

The ObjectContext class is part of the core EF API in the Microsoft .NET Framework 4 and is the class that allows you to perform queries, change tracking and update the database using the strongly typed classes that represent your model. The DbContext class is best described as a wrapper around ObjectContext that exposes the most commonly used features of ObjectContext as well as provides some simpler "shortcuts" to tasks that are frequently used but complicated to code directly with ObjectContext.

It's my guidance and Microsoft's that you should consider DbContext first when beginning new projects using EF. If you find that you occasionally need to access some of the more granular logic that the ObjectContext class provides, there's a hook to get from a DbContext instance to its underlying ObjectContext:

```
var objectContext = (myDbContextInstance as IObjectContextAdapter).ObjectContext
```

If you know that you'll be doing work that requires frequent use of ObjectContext features directly, you might prefer to use that rather than DbContext. But in general, the EF team recommends that you avoid using ObjectContext directly unless you're prevented from using DbContext for some reason.

I'll add the caveat that this guidance is meant for new projects. When working with the DbContext API, you get not only the new slimmer and smarter DbContext class, but also equally improved DbSet and DbQuery classes (counterparts to ObjectSet and ObjectQuery).

Although I'm a big fan of the DbContext, a few of its features have become my favorite little pets.

DbSet.Find

One of the new methods in the API is DbSet.Find. This helps with a common pattern developers use for data access: retrieving a single object based on its primary key.

With ObjectContext, you would have to create a full query and then execute the query using a LINQ method such as SingleOrDefault.

That would look like:

```
var partyHatQuery = from p in context.PartyHats where p.Id == 3 select p;
var partyHatInstance = partyHatQuery.SingleOrDefault();
```

You could write that more efficiently with LINQ methods and a lambda:

```
var partyHatInstance = context.PartyHats.SingleOrDefault(p => p.Id == 3);
```

How often have you executed queries that perform this simple task? You might have even abstracted this code in your own simpler method.

This is just what the EF team did for you in the DbContext API. When working with DbContext, PartyHats would be a DbSet<PartyHat>, and you can use the DbSet.Find method to quickly achieve the same query execution with:

```
context.PartyHats.Find(3)
```

This method presumes the value you provide is the key value for the class you're searching—in this case, PartyHat. EF will then execute a SingleOrDefault query on your behalf, searching for the data where Id is equal to the value passed in—in this case, 3. You'll probably pass in a variable, not an actual value.

There's another benefit to the DbSet.Find method that you can't achieve with a query. The Find method will first look in memory for a matching object that's being tracked by the context. If that's found, then EF won't bother querying the database. This is much more efficient than executing a query on the database only to throw away the results of the query if the object instance is already in memory—a wasted trip to the database that many developers trigger without realizing it.

You can also use DbSet.Find with composite keys. The signature of Find is not to take a single object but to take a parameter array. Therefore you can pass in a list of values to represent the values that make up the key.

DbSet.Local

When working with EF, I frequently found myself wanting to do something with objects that were already in memory and being tracked by a context. Typical places for this logic are in the SaveChanges override or SavingChanges method, where I'll perform some validation. (Thanks to the new Validation API that's available along with DbContext, I've been able to reduce much of this logic. But I won't discuss Validation in this column.)

ObjectContext does provide a way to discover the objects that it's tracking, but the API logic to do this is neither easy to find nor easy to code. In fact, in my book, "Programming Entity Framework" (O'Reilly Media, 2010), I wrote a set of four method extensions to help make this task simpler and more flexible.

More commonly, however, developers don't realize the difference between executing a LINQ to Entities query on the context and

interacting with those objects that the context is already tracking. For example, I've seen plenty of code where a developer retrieves data using a query and then attempts to perform logic on what's now being managed by the query:

```
  var balloons = context.Balloons.Where(b => b.Size == "L").ToList();
  var balloonCount = context.Balloons.Count();
```

In fact, these are two separate queries. The second line of code executes another query on the database and returns a count of all balloons. Typically, what the developer had intended was to get a count of the results—that is, balloons.Count.

If you don't have access to a variable but still want to find out how many Balloon objects an ObjectContext is tracking, there's a way to find out, but it's not easy: ObjectContext exposes an ObjectStateManager, which has a method called GetObjectStateEntries. This method requires that you pass in one or more EntityState enums (for example, Added, Modifed and so on) so it knows which entries to return. Although the results are queryable, filtering is unwieldy and even then what it returns is not your entities, but the ObjectStateEntry instances that represent state information about your objects.

This means that without the use of my extension methods, code to help get the count of the balloons in memory looks like this:

```
objectContext.ObjectStateManager
 .GetObjectStateEntries(EntityState.Added |
                        EntityState.Modified | EntityState.Unchanged)
 .Where(e => e.Entity is Balloon).Count();
```

If you want to capture those Balloon objects, not just the ObjectStateEntry instances, then you have to add some casting to return the ObjectStateEntry.Entity types as Balloons:

```
objectContext.ObjectStateManager
 .GetObjectStateEntries(EntityState.Added |
                        EntityState.Modified | EntityState.Unchanged)
 .Where(e => e.Entity is Balloon)
 .Select(e => e.Entity as Balloon);
```

Seeing this code might make you appreciate the new property DbSet.Local almost as much as I do.

Using DbSet.Local to get all of the tracked Balloon instances from the context, you can simply call:

```
context.Balloons.Local;
```

"Local" returns an ObservableCollection that provides two benefits. The first is that it's queryable, so you can return whatever subset of the locally cached Balloons you want. The second is that your code (or components such as data-binding controls) can listen for and react to objects being added to or removed from the cache.

Besides the discoverable property and the reduced code, there are two other notable differences between using DbSet.Local and GetObjectStateEntries. One is that Local returns objects from the particular DbSet only, whereas GetObjectStateEntries returns entries regardless of the type of objects they represent. The other difference is that Local won't return objects that the context knows are marked as Deleted. With GetObjectStateEntries, you have access to Added, Modified, Unchanged and Deleted objects as specified in the parameter list that you provide to the method.

NoTracking LINQ Queries

When discussing performance with clients, I often recommend they take advantage of the EF ability to return data that doesn't need to be tracked by the context. For example, you may have data you need to supply for a drop-down selection list. You'll never need to make changes to that data, much less persist it to the database. Therefore, it's smart to avoid the performance hit taken when EF creates ObjectStateEntry instances for each object it's tracking, as well as forcing the context to be aware of any changes made to those objects.

But with ObjectContext, the NoTracking support is available only through the ObjectQuery class, not from LINQ to Entities queries.

Here's a typical example of getting a NoTracking query using an ObjectContext (called context):

```
string entitySQL = " SELECT p, p.Filling " +
                         "FROM PartyContext.Pinatas AS p " +
                         "WHERE p.Filling.Description='Candy'";
var query=context.CreateQuery<DbDataRecord>(entitySQL);
query.MergeOption = System.Data.Objects.MergeOption.NoTracking;
var pinatasWithFilling=query.ToList();
```

The retrieved piñatas and fillings would be objects in memory, but the context would have no knowledge of them.

However, if you were to use the following LINQ to Entities query, which returns an IQueryable, not an ObjectQuery, there would be no MergeOption property:

```
context.Pinatas.Include("Filling")
  .Where(p=>p.Filling.Description=="Candy")
```

One solution is to cast the LINQ query to an ObjectQuery and then set the MergeOption. This is not only not obvious but also clunky.

Recognizing this, the EF team found a way to let you have your party cake and eat it, too, with the new AsNoTracking extension method for IQueryables that's part of the DbContext API. Now I can tack it on to my LINQ query:

```
context.Pinatas.Include("Filling")
  .Where(p=>p.Filling.Description=="Candy")
  .AsNoTracking();
```

This will return a set of Pinatas and Fillings that will be ignored by the context. EF won't wastethe effort of instantiating DbEntityEntry objects (the DbContext API version of ObjectStateEntry) for each object. Nor will it waste the effort of forcing the context to inspect those objects when DetectChanges is called.

It's simple to code and very discoverable through IntelliSense.

Icing on the Cake

These three features—Find, Local and AsNoTracking—don't enable me to perform tasks that weren't achievable with the ObjectContext. But they do make me happy every time I use them. There are so many coding tasks that the DbContext API simplifies (compared to using the ObjectContext) that it has streamlined my application development quite a bit. I've also returned to old ObjectContext code and refactored it to use DbContext along with Code First and have been able to significantly reduce the amount of code in those apps. But for developers who aren't as intimately familiar with the EF as I am, the discoverability of so many of its capabilities will make a big difference for getting up and running with it. ■

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) and "Programming Entity Framework: Code First" (2011), both from O'Reilly Media. Follow her on Twitter at twitter.com/julielerman.*

THANKS to the following technical expert for reviewing this article: *Arthur Vickers*

ComponentOne Ultimate™ delivers the tools and resources to build everything …everywhere. Whether you're a Windows, Web, or XAML developer, this ultimate dev tool collection delivers. Inside you'll find: 100s of .NET controls, OLAP data analysis controls, SharePoint Web Parts, documentation tools, LightSwitch extensions, and tools for ADO.NET Entity Framework and RIA Services. No job is too big. Bring speed, style, and functionality to your all your applications ...it is your destiny.

[BUILD EVERYTHING EVERYWHERE]

THE GALAXY IS YOURS: 100S OF .NET CONTROLS & TOOLS

COMPONENTONE
ULTIMATE
ComponentOne

MEET YOUR DESTINY AT:
COMPONENTONE.COM/GALAXY

Executive Dashboard
CEO Dashboard
$788.9 Million

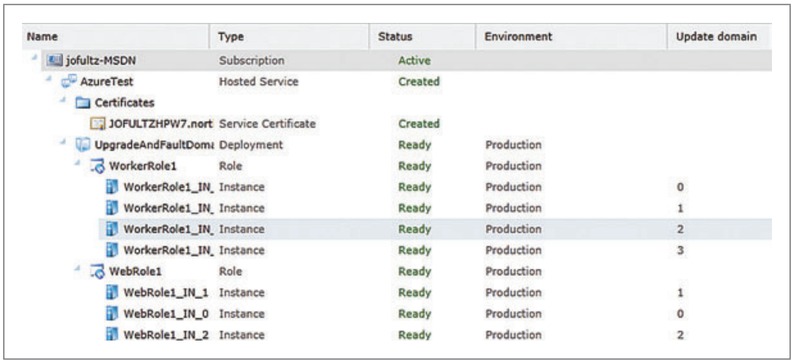
Start
Sales Dashboard
$20,573

Large Datasets
Data Points: 1,000,000

Windows Azure Deployment Domains

Lately, I've been giving a lot of thought to the deployment of applications. It turns out that for applications, the matrix for fault tolerance and upgrade path gets a bit tricky—and even trickier when applications have a mix of services, a Web UI and back-end processes. Add in geographic distribution and the logistics become even more muddied.

In large IT organizations, a minimum deployment of any Web or application server often involves two servers that are geographically separated. This easily moves up to four servers if two servers are specified for the expected load and you have a mirror site with the same setup (of course, database and other supporting server infrastructure can push the number higher still). What if the company serves multiple locations, such as North America and Europe, the Middle East and Africa (EMEA)? Now the setup gets replicated to both sides of the Atlantic, turning what started as two Web servers into eight servers for geo failover and for staging assets closer to consumers.



Name	Type	Status	Environment	Update domain
jofultz-MSDN	Subscription	Active		
AzureTest	Hosted Service	Created		
Certificates				
JOFULTZHPW7.nort	Service Certificate	Created		
UpgradeAndFaultDom:	Deployment	Ready	Production	
WorkerRole1	Role	Ready	Production	
WorkerRole1_IN_	Instance	Ready	Production	0
WorkerRole1_IN_	Instance	Ready	Production	1
WorkerRole1_IN_	Instance	Ready	Production	2
WorkerRole1_IN_	Instance	Ready	Production	3
WebRole1	Role	Ready	Production	
WebRole1_IN_1	Instance	Ready	Production	1
WebRole1_IN_0	Instance	Ready	Production	0
WebRole1_IN_2	Instance	Ready	Production	2

Figure 1 **The Windows Azure Management Console**

Eventually, an application is deployed on all these servers and everything is running along smoothly—and then some cheeky developer creates new functionality and wants to update the deployment.

As you can imagine, it takes a good bit of planning to determine the order in which servers will drain connections, get updated and tested, and then be put back into the pool. Some folks spend late nights working through upgrade plans, and that's even when there are no real problems.

Windows Azure doesn't eliminate the need for an upgrade plan, but it does take much of the complexity out of upgrading by handling most of it as part of the fabric. In this column, I'm going to cover fault domains and upgrade domains, and write a little bit of code to apply an upgrade across the deployment.

Fault and Upgrade Domains

Windows Azure includes the concepts of *fault domains* and *upgrade domains*, both of which are almost fully described by their names. Fault domains define a physical unit of deployment for an application and are typically allocated at the rack level. By placing fault domains in separate racks, you separate instances of application deployment to hardware enough that it's unlikely all would fail at the same time. Further, a failure in one fault domain should not precipitate the failure of another. When you deploy a role with two configured instances, the fabric ensures the instances are brought up in two different fault domains. Unfortunately, with fault domains, you have no control over how many domains are used or how roles are allocated to them.

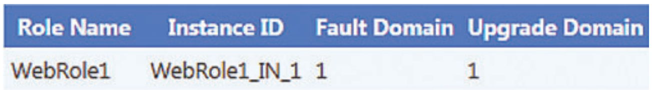
Upgrade domains are another matter. You have control over these domains and can perform incremental or rolling upgrades across a deployment by upgrading a group of instances at a time. Whereas fault domains are about physical deployment of the roles, upgrade domains relate to logical deployment. Because an upgrade domain is a logical grouping of roles, a single Web application could easily exist in five different upgrade domains divided into only two separate physical deployments (fault domains). In this case, to update a Web application, you might update all roles in group 0 (upgrade domain 0) and then all roles in group 1 and so on. You can exercise more finite control by updating individual roles one at a time in each Update Domain.

In summary, an application that requires more than one instance will be split into at least two fault domains. To make upgrading a Web application across the whole farm easier, roles are combined into logical groupings that are updated at the same time.

Figure 2 **Finding Role Information**

```
protected void GetRoleInfo()
{
  List<RoleInfo> RoleInfos = new List<RoleInfo>();

  foreach (var role in RoleEnvironment.Roles)
  {
    RoleInfo info = new RoleInfo();
    info.RoleName = role.Value.Name;

    foreach (RoleInstance roleInstance in role.Value.Instances)
    {
      info.InstanceId = roleInstance.Id;
      info.FaultDomain = roleInstance.FaultDomain.ToString();
      info.UpgradeDomain = roleInstance.UpdateDomain.ToString();

    }
    RoleInfos.Add(info);
  }
  GridView1.DataSource = RoleInfos;
  GridView1.DataBind();

}
```

AUTOSOLUTIONS
YOU NEED IT... WE HAVE IT!
JANUARY 14, 2011
Q1 STATUS REPORT 2011
SALES VOLUME & REVENUE BY REGION 2010
SALES VOLUME & REVENUE BY REGION 2011
TOP 5 MODELS IN SALE
REVENUE (by Region)
VOLUME (by Region)
Orders Repor
Company:Ana Trujillo Emparedados y helados
TOTALS
Company:Antonio Moreno Taqueria
SalesDashboard.igr
2011
Sales Performance
Top 5 countries' Sales
Annual Sales by Month
Chart Properties
Data
DataSource
Series
Designer
Preview



Role Name	Instance ID	Fault Domain	Upgrade Domain
WebRole1	WebRole1_IN_1	1	1

Figure 3 **Current WebRole Information**

Viewing the Deployment Configuration

The Windows Azure Management Console shows an Update Domain column, but not a Fault Domain column (see **Figure 1**). (Note that upgrade domain and update domain are interchangeable terms. The documentation often refers to upgrade domains, but in the API it's called an update domain.)

In **Figure 1** you can see that the numbers for my four deployments run from 0 to 3. By default, Windows Azure uses five update domains for each service and assigns them in a round-robin style. This is something you can change in the service definition file by assigning the desired number of upgrade domains to the upgradeDomainCount attribute of the ServiceDefinition element. You'll find links for each of the schemas for Web and Worker roles at msdn.microsoft.com/library/ee758711. To force a WebRole to use only three upgrade domains, for example, you set the upgradeDomainCount in the service definition file:

```
<ServiceDefinition name="<service-name>" xmlns="http://schemas.
microsoft.com/ServiceHosting/2008/10/
      ServiceDefinition" upgradeDomainCount="3">
  <WebRole name="<web-role-name>" vmsize="[ExtraSmall|Small|Medium|Larg
e|ExtraLarge]"
    enableNativeCodeExecution="[true|false]">
    ...
  </WebRole>
</ServiceDefinition>
```

This is important, because the number of update domains ultimately affects your plan and execution. Unfortunately, there's no column that lets you see fault domain assignments. By writing a little code, however, you can pull back the curtain a bit on the deployment and see both update domain and fault domain assignments, as **Figure 2** shows.

This code doesn't show a small class I defined to store the relevant information. And unfortunately, though I have this nice nested loop that goes through the roles and the instances, the API allows the code running in the page to return data related to only the specific instance running the code. Thus, the code produces a small grid with just the current WebRole information in it (see **Figure 3**), without any other instance information.

This code provides a quick look at the current WebRole's fault and upgrade domains, but you'll need to use the Get Deployment REST URI to get more comprehensive data. It returns the deployment XML, which contains, among other things, elements for <Configuration/> and for each of the <RoleInstances />. Once you've fetched the configuration, you can change it and put it back. Take a look at my October 2010 column (msdn.microsoft.com/magazine/gg232759) for examples that show many of the same operations that would be involved here.

Upgrade Strategies

There are two basic strategies for updating a Windows Azure deployment: in-place upgrades and virtual IP (or VIP) swap. VIP swap is the simpler approach and allows for full testing of the new or updated application before opening the gates to the public. Moreover, the application can run at full capacity as soon as it's live. Should there be issues when the swap is complete, you can quickly put the previous version back in place while the new deployment is being worked on.

You'll find a good reference describing what can and can't be done in each deployment model at bit.ly/x7IRO4. Here are the points that might force a choice:

- In-place update or delete and deploy are required when changing the type or number of endpoints.
- VIP swap or delete and deploy are required when changing the role name or update domain count, or when decreasing the size of local resources.

Other than these points and some SDK version considerations, it's up to you to decide.

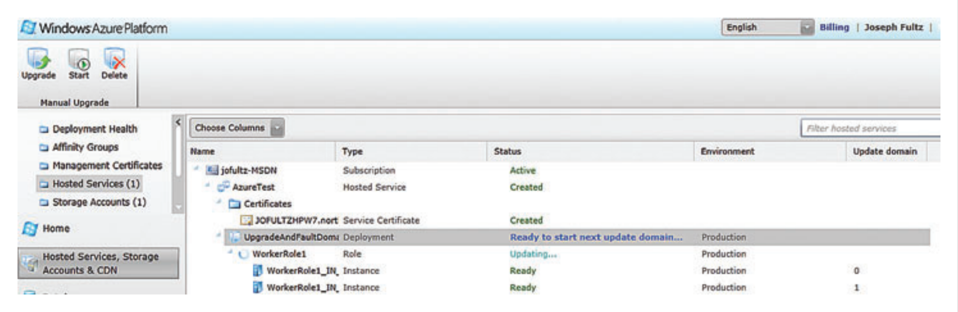
Swapping the VIP of the staging and production environments is a pretty good solution for many, if not most, cases when rolling out a new version. Sometimes it's the only way to keep the site mostly available while making changes, though if you're upgrading a large deployment, bringing up another full deployment can be cumbersome. There's also a cost associated with deploying a complete copy—one compute hour charge for each deployed instance and then the additional compute hours for the two running copies.

In Web farms nowadays, updates are generally rolled out through a farm by either: taking one server offline at a time, upgrading, bringing the server online and returning it to the farm pool; or dividing the farm into segments and draining the connections on one segment at a time, then upgrading each segment, bringing it online and returning it to the farm, and finally moving on to the next segment.

An in-place update works like the second pattern. However, the more upgrade domains used, the more the pattern resembles the first option. The upside of using a larger number of upgrade domains is that the site capacity decreases only by the size of the segment during the entire upgrade.

Figure 4 **Upgrade Decision Matrix**

Deployment Strategy	Pros	Cons
Delete and Deploy	All changes can be made	Application unavailable during process
VIP Swap	• Full application capacity • Most service changes can be made • Can test the new deployment in staging • Quick to undo by performing VIP swap again	• Hiccup in service at time of swap • Cumbersome to bring up two full deployments for larger deployments • Can't change the number or type of endpoints
In-place Update: 2 Update Domains	• Only one version running at a time • Can change number and type of endpoints • Doesn't require full deployment	• Site capacity decreased by half • A few operations can't be performed
In-place Update: 3+ Update Domains	• More site capacity during update • Can change number and type of endpoints • Doesn't require full deployment	• Multiple versions running simultaneously • A few operations can't be performed



Windows Azure Platform

English | Billing | Joseph Fultz |

Upgrade Start Delete

Manual Upgrade

- Deployment Health
- Affinity Groups
- Management Certificates
- Hosted Services (1)
- Storage Accounts (1)

Home

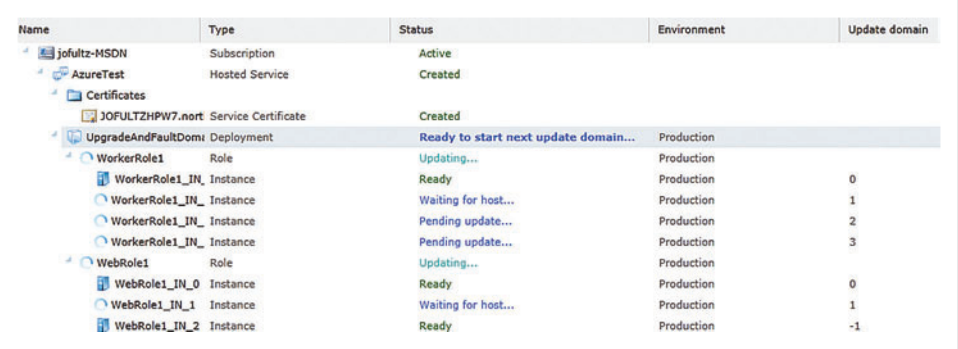
Hosted Services, Storage Accounts & CDN

Choose Columns

Filter hosted services

Name	Type	Status	Environment	Update domain
jofultz-MSDN	Subscription	Active		
AzureTest	Hosted Service	Created		
Certificates				
JOFULTZHPW7.nort	Service Certificate	Created		
UpgradeAndFaultDom:	Deployment	Ready to start next update domain...	Production	
WorkerRole1	Role	Updating...	Production	
WorkerRole1_IN_	Instance	Ready	Production	0
WorkerRole1_IN_	Instance	Ready	Production	1

Figure 5 **Windows Azure Management Console**



Name	Type	Status	Environment	Update domain
jofultz-MSDN	Subscription	Active		
AzureTest	Hosted Service	Created		
Certificates				
JOFULTZHPW7.nort	Service Certificate	Created		
UpgradeAndFaultDom:	Deployment	Ready to start next update domain...	Production	
WorkerRole1	Role	Updating...	Production	
WorkerRole1_IN_	Instance	Ready	Production	0
WorkerRole1_IN_	Instance	Waiting for host...	Production	1
WorkerRole1_IN_	Instance	Pending update...	Production	2
WorkerRole1_IN_	Instance	Pending update...	Production	3
WebRole1	Role	Updating...	Production	
WebRole1_IN_0	Instance	Ready	Production	0
WebRole1_IN_1	Instance	Waiting for host...	Production	1
WebRole1_IN_2	Instance	Ready	Production	-1

Figure 6 **Update Activity**

The primary challenge that traditional non-cloud deployments face is the same for cloud deployments: when you perform rolling upgrades, mixed versions of the application will be running. The instances might deliver different visuals, use different data and service connections, and so forth. This can lead to site errors or even undesirable user experiences, and it may be completely unacceptable for your business. Moreover, it puts a heavy burden on the development and test teams to make sure the application will run when there are multiple versions in play.

What do you do if you can't use VIP swap and availability requirements preclude a delete and deploy? You might try using only two update domains and performing an in-place update, which keeps a single version of the application running during the deployment. The downside: half of your site's capacity will be unavailable during the transition.

The grid in **Figure 4** might help you consider which approach to employ in performing an upgrade.

In-Place Upgrade

Nice advancements have been made in the ability to perform the upgrade both within the management console and via scripting. For small to midsize organizations with a relatively modest number of deployments, it's easiest to manage the updates through the Windows Azure Management Console, shown in **Figure 5**.

Figure 7 **Domain Matrix**

Instance	Upgrade Domain	Fault Domain
0	0	0
1	1	1
2	2	0

As you can see in the upper left corner of the screen, a Manual Upgrade is running. This requires clicking the Start button to initiate the process for each upgrade domain—that's the manual part of it. Once the update is started, the console displays what's going on in the instances in each domain, as shown in **Figure 6**.

The manual, push-button method works well for smaller deployments. For larger deployments or those where you want to automate the build-test-deploy process, you should choose a scripted approach. You can automate the process using the CSManage command-line tool, which you can download from bit.ly/A6uQRi. CSManage will initiate the upgrade and walk through the process of upgrading one update domain at a time from the command line. Though this is helpful, there's a level of fine control that can only be accomplished using the REST API directly.

Customizing Your Upgrade Strategy with Fault Domains

If for one reason or another you've decided to not walk the update domains from 0 – n and instead plan to use your own starting point or order, you'll need to take a look at the combination of update and fault domains. The grid in **Figure 7** makes it obvious that if you were to update Upgrade Domain 1, and Fault Domain 0 faulted during the update, the site would be completely down. This should normally be covered by the fabric, though, and the grid shows that if the update happens in order, there will always be different fault domains running.

The lesson here is to consider potential consequences during planning, and to not "fix" something that's already working.

Wrapping Up

When you're designing a Windows Azure application, you need to take deployment architecture into account. Windows Azure provides the functionality of the fabric to ensure that an application will not fault due to a single hardware failure, while providing an easy, automatic way to incrementally update the deployment. Still, support for an in-place update is something that has to be designed into the application—and the update that's being pushed.

You can update a Windows Azure service using VIP swap or a two-upgrade-domain, in-place plan where a full in-place update can't be supported. Last, there are both UI and programmatic means to control the deployment and updates so that you can perform a scheduled update or even use a build-test-deploy schedule or a scheduled update. ■

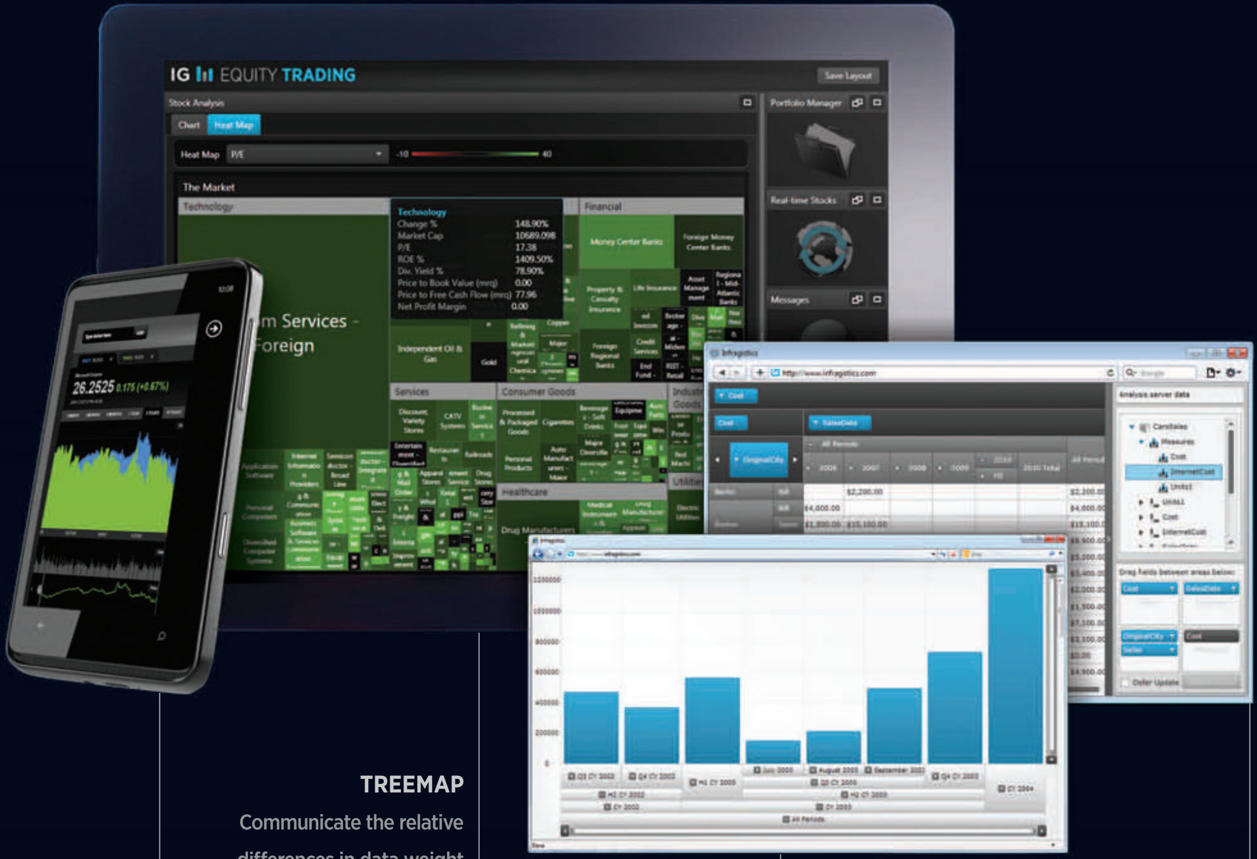
JOSEPH FULTZ *is a software architect at Hewlett-Packard Co., working as part of the HP.com Global IT group. Previously he was a software architect for Microsoft working with its top-tier enterprise and ISV customers defining architecture and designing solutions.*

THANKS to the following technical expert for reviewing this article: *Don Glover*

Deliver the Ultimate User Experience

check out infragistics.com/ultimate

NetAdvantage® ULTIMATE



FINANCIAL CHARTING
With support for multiple chart styles, and technical indicators built in, financial charting capabilities are on the money.

TREEMAP
Communicate the relative differences in data weight more effectively, with customizable color and flexible layouts.

OLAP AXIS CHART
Take your data to new depths with the seemingly endless drilldown capability of the OLAP Axis Chart.

OLAP GRID
Provide highly-interactive pivot grid functionality in all of your applications.

INFRAGISTICS™
DESIGN / DEVELOP / EXPERIENCE

Infragistics Sales 800 231 8588 • Infragistics Europe Sales +44 (0) 800 298 9055 • Infragistics India +91 80 4151 8042 • @infragistics

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

YOUR MAP TO THE .NET DEVELOPMENT PLATFORM

WHAT YOU LEARN IN VEGAS WON'T STAY IN VEGAS

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

Las Vegas | **March 26-30** | Mirage Resort and Casino

PLATINUM SPONSOR: Microsoft

SUPPORTED BY: Microsoft Visual Studio | msdn | Visual Studio MAGAZINE

PRODUCED BY: 1105 MEDIA INC

A Peek at What You'll Learn at Visual Studio Live! Las Vegas:

Keynote:

The Future of User Experience: The Natural User Interface (NUI)

Tim Huckaby,
Microsoft RD & MVP, Chairman/Founder, InterKnowlogy,
CEO/Chairman, Actus Interactive Software

Track: Visual Studio 2010+/.NET 4+

Session: Application Lifecycle Management and Visual Studio: What's Next

Explore the new features in Team Foundation Server as well as the new Azure-based Team Foundation Service. You'll learn what's new for work management, including task boards, how to gather user requirements and ensure better products and much more.

Track: Silverlight / WPF

Session: Top 7 Lessons Learned On My First Big Silverlight Project

Hear 7 things instructor Ben Day learned while leading his first big Silverlight application – from unit tests and the architectural havoc caused by async WCF to real-world ViewModel tips and "x:Name" code smells.

Track: Web

Session: Hack Proofing Your Asp.Net Web Forms and MVC

Developers are notoriously lax with including security in their applications. In an age of hacking, this talk aims to arm the developer with an arsenal of protections to use while developing. Techniques such as Cross Site Scripting, SQL Injection, Session Hijacking, and Cross Site Request Forgery will be covered.

Track: Cloud Computing

Session: Architecture Best Practices on Windows Azure

Learn Architecture Best Practices that will help make your solutions better in performance, cost, integration, and security. Leave this session with the knowledge needed to start using Windows Azure quickly and the best practices that can help your business.

Track: Data Management

Session: Entity Framework Code First - Beyond the Basics

Go beyond the basic scenarios – understand performance analysis tips for Entity framework, learn database scheme evolution handling techniques and about the organization of database and conceptual models via complex types and hierarchies.

Track: HTML5

Session: Building Windows 8 Applications with HTML5 and jQuery

One of the many new advances in Windows 8 is the ability to create Windows applications using HTML, CSS and JavaScript. In this session, take a look at the Windows 8 technology stack on which these applications run, how HTML/CSS/JS apps actually run, and discuss the implications of the different ways to utilize third party libraries such as jQuery. By the end of this session, you'll have a solid idea of what it means to have a Metro-style application built with web technologies.

Track: Windows 8/WinRT

Session: Windows 8 Metro-style Application Contracts and Extensibility

Explore contracts – a new feature of Windows 8 Metro-style applications that allow applications to interact with the operating system and other Metro-style applications in a standard way – and learn how they can be used in any Windows 8 Metro-style application to provide a seamless, integrated experience for users.

Track: Windows Phone 7

Session: Making Money on Your WP7 Apps & Games with the Advertising SDK

Learn how to use the Microsoft Advertising SDK and the Microsoft PubCenter to create and display ads in your Windows Phone 7 apps and games.

Track: Cross Platform Mobile

Session: Building Mobile Apps with CSLA .NET

Learn how to create business classes that compile and run on all three mobile platforms, iPhone/iPad (using MonoTouch), Android (using Mono for Android), and Windows Phone (using Silverlight), as well as on .NET. The result is the ability to reuse large portions of your code when building mobile apps.

Register at vslive.com/lasvegas
Use Promo Code FEBAD

VISUAL STUDIO LIVE! LAS VEGAS AGENDA AT-A-GLANCE

HTML5	Web	Visual Studio 2010+/.NET 4+	Cloud Computing	Data Management	Silverlight / WPF	Windows Phone 8/WinRT	Windows Phone 7	Cross Platform Mobile

Visual Studio Live! Pre-Conference Workshops: Monday, March 26, 2012 *(Separate entry fee required)*

MWKS1 **Workshop: Full Application Lifecycle with TFS and CSLA .NET** *Rockford Lhotka & Brian Randell*	MWKS2 **Workshop: Creating Today's User Experiences - An Entry Point for Developers** *Billy Hollis*	MWKS3 **Workshop: SQL Server for Developers** *Andrew Brust & Leonard Lobel*

Visual Studio Live! Day 1: Tuesday, March 27, 2012

Keynote *Microsoft To Be Announced*

T1 **Introduction to the Windows Runtime** *Rockford Lhotka*	T2 **HTML5 and Internet Explorer: A Developer Overview** *Ben Hoelting*	T3 **Introducing SQL Server Data Tools (Codenamed "Juneau")** *Leonard Lobel*	T4 **Application Lifecycle Management and Visual Studio: What's Next** *Brian Randell*
T5 **Windows 8 Metro-style Application Contracts and Extensibility** *Brian Peek*	T6 **Advanced ASP.NET MVC, HTML5 and the .NET Stack** *Ben Hoelting*	T7 **So Many Choices, So Little Time: Understanding Your .NET 4.0 Data Access Options** *Leonard Lobel*	T8 **Microsoft - To Be Announced**

Lunch & Expo Hall

CTT1 **Chalk Talk: Improve Your Code with Anonymous Types and Lamda Expressions** *Deborah Kurata*	CTT2 **Chalk Talk: Slice Development Time with ASP.NET MVC and Razor** *Philip Japikse*

T9 **Building Data Driven Applications Using WinRT and XAML** *Sergey Barskiy*	T10 **HTML5/jQuery On-Ramp** *Rich Dudley*	T11 **Microsoft - To Be Announced**	T12 **Microsoft - To Be Announced**
T13 **A Look at Windows 8 Metro Apps and WinRT Internals** *Vishwas Lele*	T14 **Building Windows 8 Applications with HTML5 and jQuery** *Rich Dudley*	T15 **Entity Framework Code First - Beyond the Basics** *Sergey Barskiy*	T16 **What's New in the .NET 4.5 BCL** *Jason Bock*

Welcome Reception

Visual Studio Live! Day 2: Wednesday, March 28, 2012

Keynote: The Future of User Experience: The Natural User Interface (NUI)
Tim Huckaby, Microsoft RD & MVP, Chairman/Founder, InterKnowlogy, CEO/Chairman, Actus Interactive Software

W1 **Windows Presentation Foundation for Developers** *Philip Japikse*	W2 **Creating a Data Driven Web Site Using WebMatrix and ASP.NET Razor** *Rachel Appel*	W3 **Windows Azure Platform Overview** *Vishwas Lele*	W4 **XNA Games for Windows Phone** *Brian Peek*
W5 **MVVM in Practice aka "Code Behind"- Free WPF** *Tiberiu Covaci*	W6 **MVC for WebForms Developers: Comparing and Contrasting** *Miguel Castro*	W7 **Building Your First Azure Application** *Michael Stiefel*	W8 **Building Mobile Apps with CSLA .NET** *Rockford Lhotka*

Birds-of-a-Feather Lunch & Expo Hall

CTW1 **Chalk Talk: How Orchard CMS Works** *Rachel Appel*	CTW2 **Chalk Talk: Parallel Programming 101** *Tiberiu Covaci*

W9 **Silverlight, WCF RIA Services and Your Business Objects** *Deborah Kurata*	W10 **Getting Started with ASP.NET MVC3 with a Dash of 4** *Philip Japikse*	W11 **Deciding Between Relational Databases and Tables in the Cloud** *Michael Stiefel*	W12 **Making Money on Your WP7 Apps & Games with the Advertising SDK** *Chris G. Williams*
W13 **Top 7 Lessons Learned On My First Big Silverlight Project** *Ben Day*	W14 **Fast, Faster ... Async ASP.NET** *Tiberiu Covaci*	W15 **Architecture Best Practices on Windows Azure** *Nuno Godinho*	W16 **Mobile + Cloud: Using the Windows Azure Toolkit for Mobile Devices** *Eric D. Boyd*

Wild Wednesday

Visual Studio Live! Day 3: Thursday, March 29, 2012

TH1 **WPF Validation - Techniques & Styles** *Miguel Castro*	TH2 **Entity Framework 4.1 for Real Web Applications** *Adam Tuliper*	TH3 **Tips & Tricks on Architecting Windows Azure for Costs** *Nuno Godinho*	TH4 **Consuming Async Web Services In Your Windows Phone Apps & Games** *Chris G. Williams*
TH5 **Infinite Whitespace: Implementing Viewport Navigation in XAML** *Billy Hollis*	TH6 **Hack Proofing Your ASP.NET Web Forms and MVC Applications** *Adam Tuliper*	TH7 **Moving Web Apps to the Cloud** *Eric D. Boyd*	TH8 **Reach The Mobile Masses With ASP.NET MVC 4 and jQuery Mobile** *Keith Burnell*
TH9 **Writing Asynchronous Code Using .NET 4.5 and C# 5.0** *Brian Peek*	TH10 **Introduction to jQuery QUnit** *John Petersen*	TH11 **SQL Azure Intro and What's New** *Eric D. Boyd*	TH12 **LightSwitch Onramp** *Rich Dudley*

Lunch

TH13 **Static Analysis in .NET** *Jason Bock*	TH14 **Busy Developer's Guide to NodeJS** *Ted Neward*	TH15 **Power View: Analysis and Visualization for Your Application's Data** *Andrew Brust*	TH16 **Incorporating LightSwitch Into Your Existing ASP.NET Applications** *Michael Washington*
TH17 **How to Be a C# Ninja in 10 Easy Steps** *Ben Day*	TH18 **Extending ASP.NET MVC with jQuery/Ajax and jSON** *John Petersen*	TH19 **Microsoft's Big Play for Big Data** *Andrew Brust*	TH20 **Creating LightSwitch Control Extensions** *Michael Washington*

Visual Studio Live! Post-Conference Workshops: Friday, March 30, 2012 *(Separate entry fee required)*

FWKS1 **Workshop: Programming with WCF in One Day** *Miguel Castro*	FWKS2 **Workshop: Architecture Katas** *Ted Neward*

For the complete session schedule and full session descriptions, please check the Visual Studio Live! Las Vegas web site at ***vslive.com/lasvegas***
**Speakers and Sessions Subject to Change.*

Asynchronous Programming in C++ Using PPL

Artur Laksberg

Hollywood casting directors are often said to brush off aspiring performers with a dismissive "don't call us; we'll call you." For developers, however, that phrase describes the way many software frameworks work—instead of letting the programmer drive the flow of control for the whole application, the framework controls the environment and invokes callbacks or event handlers provided by the programmer.

In asynchronous systems, this paradigm lets you decouple the start of the asynchronous operation from its completion. The programmer initiates the operation and then registers a callback that will be invoked when the results are available. Not having to wait for completion means you can do useful work while the operation is in progress—service the message loop or start other asynchronous operations, for example. The "frosted window," the "spinning donut" and other such phenomena will become relics of the past if you follow this pattern rigorously for all potentially blocking operations. Your apps will become—you've heard this one before—fast and fluid.

In Windows 8, asynchronous operations are ubiquitous, and WinRT offers a new programming model for dealing with asynchrony in a consistent way.

Figure 1 demonstrates the basic pattern of working with asynchronous operations. In the code, a C++ function reads a string from a file.

The first thing to notice is that the return type of ReadString is void. That's right: The function doesn't return a value; instead it takes a user-provided callback and invokes it when the result is available. Welcome to the world of asynchronous programming—don't call us; we'll call you!

This article uses prerelease versions of Windows 8 and Visual Studio 2012. All information is subject to change.

This article discusses:

- Asynchronous programming in WinRT
- The anatomy of a WinRT asynchronous operation
- Composing multiple asynchronous operations
- Using Parallel Patterns Library tasks
- Error handling and cancellation

Technologies discussed:

C++, Windows 8, Visual Studio 2012, Visual Studio Parallel Patterns Library

The Anatomy of a WinRT Asynchronous Operation

At the heart of the asynchrony in WinRT are the four interfaces defined in the Windows::Foundation namespace: IAsyncOperation, IAsyncAction, IAsyncOperationWithProgress and IAsyncActionWithProgress. All potentially blocking or long-running operations in WinRT are defined as asynchronous. By convention, the name of the method ends with "Async" and the return type is one of the four interfaces. Such is the method GetFileAsync in the example in Figure 1, returning an IAsyncOperation<StorageFile^>. Many asynchronous operations do not return a value and their type is IAsyncAction. The operations that can report progress are exposed via IAsyncOperationWithProgress and IAsyncActionWithProgress.

Figure 1 **Reading from a File**

```
template<typename Callback>
void ReadString(String^ fileName, Callback func)
{
  StorageFolder^ item = KnownFolders::PicturesLibrary;

  auto getFileOp = item->GetFileAsync(fileName);
  getFileOp->Completed = ref new AsyncOperationCompletedHandler<StorageFile^>
    ([=](IAsyncOperation<StorageFile^>^ operation, AsyncStatus status)
  {
    auto storageFile = operation->GetResults();
    auto openOp = storageFile->OpenAsync(FileAccessMode::Read);
    openOp->Completed =
      ref new AsyncOperationCompletedHandler <IRandomAccessStream^>
      ([=](IAsyncOperation<IRandomAccessStream^>^ operation, AsyncStatus status)
    {
      auto istream = operation->GetResults();
      auto reader = ref new DataReader(istream);
      auto loadOp = reader->LoadAsync(istream->Size);
      loadOp->Completed = ref new AsyncOperationCompletedHandler<UINT>
        ([=](IAsyncOperation<UINT>^ operation, AsyncStatus status)
      {
        auto bytesRead = operation->GetResults();
        auto str = reader->ReadString(bytesRead);
        func(str);
      });
    });
  });
}
```

To specify the completion callback for an asynchronous operation, you set the Completed property. This property is a delegate that takes the asynchronous interface and the status of the completion. Though the delegate can be instantiated with a function pointer, most often you'd use a lambda (I expect that by now you're familiar with this part of C++11).

> Very often, you'll find yourself using multiple asynchronous operations together.

To get the value of the operation, you call the GetResults method on the interface. Notice that though this is the same interface returned to you from the GetFileAsync call, you can only call GetResults on it when you're within the completion handler.

The second parameter to the completion delegate is AsyncStatus, which returns the status of the operation. In a real world application, you'd check its value before calling GetResults. In **Figure 1**, I omitted this part for brevity.

Very often, you'll find yourself using multiple asynchronous operations together. In my example, I first get an instance of StorageFile (by calling GetFileAsync), then open it using OpenAsync and getting IInputStream. Next, I load the data (LoadAsync) and read it using the DataReader. Finally, I get the string and call the user-provided callback func.

Composition

Separating the start of the operation from the completion is essential for eliminating blocking calls. The problem is, composing multiple callback-based asynchronous operations is hard, and the resulting code is difficult to reason about and debug. Something has to be done to rein in the ensuing "callback soup."

Let's consider a concrete example. I want to use the ReadString function from the previous sample to read from two files sequentially and concatenate the result into a single string. I'm going to again implement it as a function taking a callback:

```
template<typename Callback>
void ConcatFiles1(String^ file1, String^ file2, Callback func)
{
  ReadString(file1, [func](String^ str1) {
    ReadString(file2, [func](String^ str2) {
      func(str1+str2);
    });
  });
}
```

Not too bad, right?

If you don't see a flaw in this solution, though, think about this: When will you start reading from file2? Do you really need to finish reading the first file before you start reading the second one? Of course not! It's far better to start multiple asynchronous operations eagerly and deal with the data as it comes in.

Let's give it a try. First, because I start two operations concurrently and return from the function before the operations complete, I'll need a special heap-allocated object to hold the intermediate results. I call it the ResultHolder:

```
ref struct ResultHolder
{
  String^ str;
};
```

As **Figure 2** shows, the first operation to succeed will set the results->str member. The second operation to complete will use that to form the final result.

This will work … most of the time. The code has an obvious race condition, and it doesn't handle errors, so we still have a long way to go. For something as simple as joining two operations, that's an awful lot of code—and it's tricky to get right.

Tasks in the Parallel Patterns Library

The Visual Studio Parallel Patterns Library (PPL) is designed to make writing parallel and asynchronous programs in C++ easy and productive. Instead of operating at the level of threads and thread

Figure 2 **Reading from Two Files Concurrently**

```
template<typename Callback>
void ConcatFiles(String^ file1, String^ file2, Callback func)
{
  auto results = ref new ResultHolder();

  ReadString(file1, [=](String^ str) {
    if(results->str != nullptr) { // Beware of the race condition!
      func(str + results->str);
    }
    else{
      results->str = str;
    }
  });

  ReadString(file2, [=](String^ str) {
    if(results->str != nullptr) { // Beware of the race condition!
      func(results->str + str);
    }
    else{
      results->str = str;
    }
  });
}
```

Figure 3 **Reading from Files Using Nested PPL Tasks**

```
task<String^> ReadStringTask(String^ fileName)
{
  StorageFolder^ item = KnownFolders::PicturesLibrary;
  task<StorageFile^> getFileTask(item->GetFileAsync(fileName));
  return getFileTask.then([](StorageFile^ storageFile) {
    task<IRandomAccessStream^> openTask(storageFile->OpenAsync(
      FileAccessMode::Read));
    return openTask.then([](IRandomAccessStream^ istream) {
      auto reader = ref new DataReader(istream);
      task<UINT> loadTask(reader->LoadAsync(istream->Size));
      return loadTask.then([reader](UINT bytesRead) {
        return reader->ReadString(bytesRead);
      });
    });
  });
}
```

pools, users of PPL get to use higher-level abstractions such as tasks, parallel algorithms like parallel_for and the parallel_sort and concurrency-friendly containers such as concurrent_vector.

New in Visual Studio 2012, the *task* class of the PPL allows you to succinctly represent an individual unit of work to be executed asynchronously. It allows you to express your program logic in terms of independent (or interdependent) tasks and let the runtime take care of scheduling these tasks in the optimal manner.

What makes tasks so useful is their composability. In its simplest form, two tasks can be composed sequentially by declaring one task to be a *continuation* of another. This seemingly trivial construct enables you to combine multiple tasks in interesting ways. Many higher-level PPL constructs such as join and choice (which I'll talk about in moment) are themselves built using this concept. Task continuations can also be used to represent completions of asynchronous operations in a more concise way. Let's revisit the sample from **Figure 1** and now write it using PPL tasks, as shown in **Figure 3**.

Because I'm now using tasks instead of callbacks to represent asynchrony, the user-provided callback is gone. This incarnation of the function returns a task instead.

In the implementation, I created the getFileTask task from the asynchronous operation returned by GetFileAsync. I then set up the completion of that operation as a continuation of the task (the then method).

Figure 4 **Chaining Multiple Tasks**

```
ref struct Holder
{
  IDataReader^ Reader;
};
task<String^> ReadStringTask(String^ fileName)
{
  StorageFolder^ item = KnownFolders::PicturesLibrary;

  auto holder = ref new Holder();

  task<StorageFile^> getFileTask(item->GetFileAsync(fileName));
  return getFileTask.then([](StorageFile^ storageFile) {
    return storageFile->OpenAsync(FileAccessMode::Read);
  }).then([holder](IRandomAccessStream^ istream) {
    holder->Reader = ref new DataReader(istream);
    return holder->Reader->LoadAsync(istream->Size);
  }).then([holder](UINT bytesRead) {
    return holder->Reader->ReadString(bytesRead);
  });
}
```

The then method deserves a closer look. The parameter to the method is a lambda expression. Actually, it could also be a function pointer, a function object, or an instance of std::function—but because lambda expressions are ubiquitous in PPL (and indeed in modern C++), from here on I'll just say "the lambda" whenever I mean any type of a callable object.

The return type of the then method is a task of some type T. This type T is determined by the return type of the lambda passed to then. In its basic form, when the lambda returns an expression of type T, the then method returns a task<T>. For example, the lambda in the following continuation returns an int; therefore, the resulting type is a task<int>:

```
task<int> myTask = someOtherTask.then([]() { return 42; });
```

> It's far better to start multiple asynchronous operations eagerly and deal with the data as it comes in.

The type of the continuation used in **Figure 3** is slightly different. It returns a task and performs the *asynchronous unwrapping* of that task so that the resulting type is not a task<task<int>> but a task<int>:

```
task<int> myTask = someOtherTask.then([]() {
  task<int> innerTask([]() {
    return 42;
  });
  return innerTask;
});
```

If all this feels a bit dense, don't let that slow you down. I promise after a few more motivating examples it will make more sense.

Task Composition

Armed with what was covered in the previous section, let's continue to build on the file-reading example.

Recall that in C++ all local variables residing in functions and lambdas are lost on returning. To keep the state around, you must manually copy the variables into the heap or some other long-lived storage. That's the reason I created the holder class earlier. In lambdas that run asynchronously, you need to be careful not to capture any state from the enclosing function by pointer or reference; otherwise, when the function finishes, you'll end up with a pointer to an invalid memory location.

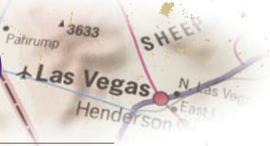
I will capitalize on the fact that the then method performs the unwrapping on the asynchronous interfaces, and rewrite our sample in a more succinct form—albeit at the cost of introducing another holder struct, shown in **Figure 4**.

Compared with the sample in **Figure 3**, this code is easier to read because it resembles sequential steps as opposed to a "staircase" of nested operations.

In addition to the then method, PPL has several other compositional constructs. One is the join operation, implemented by the when_all method. The when_all method takes a sequence of tasks

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS



YOUR MAP TO THE .NET DEVELOPMENT PLATFORM

Las Vegas | **March 26-30** | Mirage Resort and Casino

WHAT YOU LEARN IN VEGAS WON'T STAY IN VEGAS

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

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

- **Windows 8 / WinRT**
- **Silverlight / WPF**
- **Web**
- **Visual Studio 2010+ / .NET 4.0**
- **Cloud Computing**
- **Data Management**
- **HTML5**
- **Windows Phone 7**
- **Cross Platform Mobile**

vslive.com/lasvegas

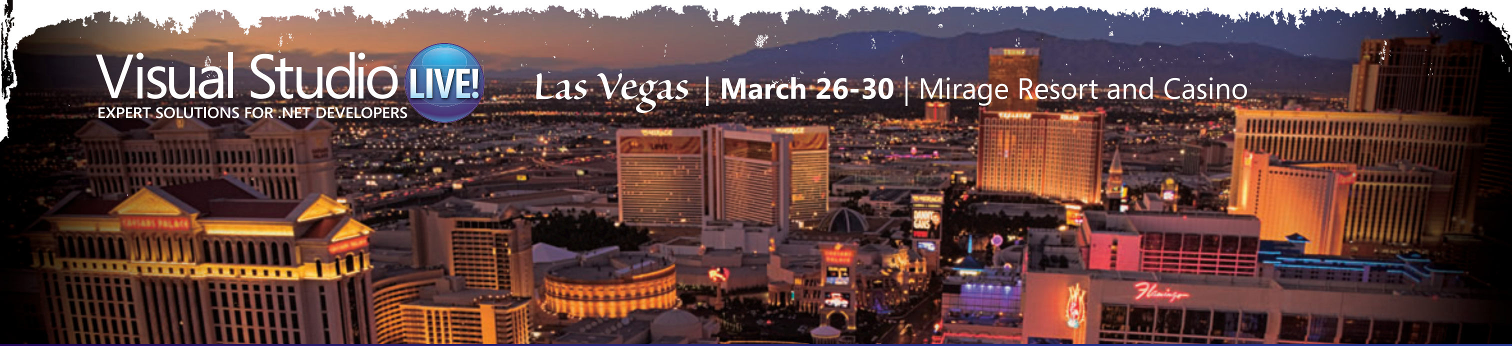


Figure 5 **Authoring Asynchronous Operations in PPL**

```
IAsyncOperation<float>^ operation = create_async([]() {
  return 42.0f;
});

IAsyncAction^ action = create_async([]() {
    // Do something, return nothing
});

IAsyncOperationWithProgress<float,int>^ operation_with_progress =
  create_async([](progress_reporter<int> reporter) {
    for(int percent=0; percent<100; percent++) {
      reporter.report(percent);
    }
    return 42.0f;
  });

IAsyncActionWithProgress<int>^ action_with_progress =
  create_async([](progress_reporter<int> reporter) {
    for(int percent=0; percent<100; percent++) {
      reporter.report(percent);
    }
  });
```

and returns the resulting task, which collects the output of all the constituent tasks into an std::vector. For the common case of two arguments, PPL has a convenient shorthand: the operator &&.

This is how I used the join operator to re-implement the file concatenation method:

```
task<String^> ConcatFiles(String^ file1, String^ file2)
{
  auto strings_task = ReadStringTask(file1) && ReadStringTask(file2);
  return strings_task.then([](std::vector<String^> strings) {
    return strings[0] + strings[1];
  });
}
```

The choice operation is also useful. Given a series of tasks, choice (implemented by the *when_any* method) completes when the first task in the sequence completes. Like join, choice has a two-argument shorthand in the form of the operator ||.

Choice is handy in scenarios such as redundant or speculative execution; you launch several tasks and the first one to complete delivers the required result. You could also add a timeout to an operation—start with an operation that returns a task and combine it with a task that sleeps for a given amount of time. If the sleeping task completes first, your operation has timed out and can therefore be discarded or canceled.

PPL has another construct that helps with composability of tasks—the task_completion_event, which you can use for interoperability of tasks and non-PPL code. A task_completion_event can be passed to a thread or to an IO completion callback that's expected to eventually set it. A task created from the task_completion_event will be completed once the task_completion_event is set.

Figure 6 **Consuming the Image Transformation Routine in JavaScript**

```
var transformer = new ImageCartoonizerBackend.ImageTransformer();
...
transformer.getTransformImageAsync(copiedFile.path, dstImgPath).then(
  function () {
    // Handle completion…
  },
  function (error) {
    // Handle error…
  },
  function (progressPercent) {
    // Handle progress:
    UpdateProgress(progressPercent);
  }
);
```

Authoring Asynchronous Operations with PPL

Whenever you need to extract the last ounce of performance from your hardware, C++ is the language of choice. Other languages have their place in Windows 8: The JavaScript/HTML5 combo is great for writing GUIs; C# offers a productive developer experience; and so on. To write a Metro style app, use what works for you; use what you know. In fact, you can use many languages in the same app.

Often, you'll find yourself writing the front-end of the application in a language like JavaScript or C#, and the back-end component in C++ for maximum performance. If the operation exported by your C++ component is either compute-bound or I/O-bound, it's a good idea to define it as an asynchronous operation.

To implement the four WinRT asynchronous interfaces mentioned earlier—IAsyncOperation, IAsyncAction, IAsyncOperationWithProgress and IAsyncActionWithProgress—PPL defines the create_async method and the progress_reporter class, both in the concurrency namespace.

> The PPL is designed to make writing parallel and asynchronous programs in C++ easy and productive.

In its simplest form, create_async takes a lambda or a function pointer that returns a value. The type of the lambda determines the type of the interface returned from create_async.

Given a lambda with no parameters that returns a non-void type T, create_async returns an implementation of the IAsyncOperation<T>. For a lambda returning void, the resulting interface is IAsyncAction.

The lambda can take a parameter of type progress_reporter<P>. The instance of this type is used to post progress reports of type P back to the caller. For example, the lambda taking a progress_reporter<int> can report the percentage of completion as an

Figure 7 **Error-handling Continuation**

```
task<image> take_picture([]() {
  if (!init_camera())
    throw std::exception("can’t init camera");
  return get_image();
});

take_picture.then([](task<image> antecedent) {
  try
  {
    image img = antecedent.get();
  }
  catch (std::exception ex)
  {
    // Handle exception here
  }
});
```

Figure 8 **Read String from File with Error Handling**

```
task<String^> ReadStringTaskWithErrorHandling(String^ fileName)
{
  StorageFolder^ item = KnownFolders::PicturesLibrary;

  auto holder = ref new Holder();

  task<StorageFile^> getFileTask(item->GetFileAsync(fileName));
  return getFileTask.then([](StorageFile^ storageFile) {
    return storageFile->OpenAsync(FileAccessMode::Read);
  }).then([holder](IRandomAccessStream^ istream) {
    holder->Reader = ref new DataReader(istream);
    return holder->Reader->LoadAsync(istream->Size);
  }).then([holder](task<UINT> bytesReadTask) {
    try
    {
      UINT bytesRead = bytesReadTask.get();
      return holder->Reader->ReadString(bytesRead);
    }
    catch (Exception^ ex)
    {
      String^ result = ""; // return empty string
      return result;
    }
  });
}
```

integer value. The return type of the lambda in this case determines whether the resulting interface is IAsyncOperationWithProgress <T,P> or IAsyncAction<P>. See **Figure 5**.

To expose an asynchronous operation to other WinRT languages, define a public ref class in your C++ component and have a function that returns one of the four asynchronous interfaces. You'll find a concrete example of a hybrid C++/JavaScript application in the PPL Sample Pack (to get it, search online for "Asynchrony with PPL"). Here's a snippet that exposes the image transformation routine as an asynchronous action with progress:

```
public ref class ImageTransformer sealed
{
public:
  //
  // Expose image transformation as an asynchronous action with progress
  //
  IAsyncActionWithProgress<int>^ GetTransformImageAsync(String^ inFile,
String^ outFile);
}
```

As **Figure 6** shows, the client part of the application is implemented in JavaScript using the promise object.

Error Handling and Cancellation

Attentive readers might have noticed that this treatise on asynchrony so far completely lacks any notion of error handling and cancellation. This subject can be neglected no longer!

Inevitably, the file-reading routine will be presented with a file that doesn't exist or can't be opened for one reason or another. The dictionary-lookup function will encounter a word it doesn't know. The image transformation won't produce a result fast enough and will be canceled by the user. In these scenarios, an operation terminates prematurely, before its intended completion.

In modern C++, exceptions are used to indicate errors or other exceptional conditions. Exceptions work wonderfully within a single thread—when an exception is thrown, the stack is unwound until the appropriate catch block down the call stack is encountered. Things get messy when concurrency is thrown into the mix, because an exception originating from one thread can't be easily caught in another thread.

Consider what happens with tasks and continuations: when the body of a task throws an exception, its flow of execution is interrupted and it can't produce a value. If there's no value that can be passed to the continuation, the continuation can't run. Even for void tasks that yield no value, you need be able to tell whether the antecedent task has completed successfully.

That's why there's an alternative form of continuation: For a task of type T, the lambda of the error-handling continuation takes a task<T>. To get the value produced by the antecedent task, you must call the get method on the parameter task. If the antecedent task completes successfully, so will the get method. Otherwise, get will throw an exception.

I want to emphasize an important point here. For any task in PPL, including a task created from an asynchronous operation, it is *syntactically* valid to call get on it. However, before the result is available, get would have to block the calling thread, and of course that would fly in the face of our "fast and fluid" mantra. Therefore, calling get on a task is discouraged in general and prohibited in an STA (the runtime will throw an "invalid operation" exception). The only time you can call get is when you've got the task as a parameter to a continuation. **Figure 7** shows an example.

> What makes tasks so useful is their composability.

Every continuation in your program can be an error-handling one, and you may choose to handle exceptions in every continuation. However, in a program composed of multiple tasks, handling exceptions in every continuation can be overkill. Fortunately, this doesn't have to happen. Similar to unhandled exceptions working their way down the call stack until the frame where they're caught, exceptions thrown by tasks can "trickle down" to the next continuation in the chain to the point where they are eventually handled. And handled they must be, for if an exception remains unhandled past the lifetime of the tasks that could have handled it, the runtime throws the "unobserved exception" exception.

Let's now return to our file-reading example and augment it with error handling. All the exceptions thrown by WinRT are of type

Figure 9 **Canceling and Reaction to the Cancellation Request in a Task**

```
cancellation_token_source ct;

task<int> my_task([]() {
  // Do some work
  // Check if cancellation has been requested
  if(is_task_cancellation_requested())
  {
        // Clean up resources:
        // ...
        // Cancel task:
        cancel_current_task();
  }
  // Do some more work
  return 1;
}, ct.get_token());
...
ct.cancel(); // attempt to cancel
```

Platform::Exception, so this is what I'm going to catch in my last continuation, as shown in **Figure 8**.

Once the exception has been caught in a continuation, it's considered "handled," and the continuation returns a task that completes successfully. So, in **Figure 8**, the caller of the ReadStringWithErrorHandling will have no way of knowing whether the file reading completed successfully. The point I'm trying to make here is that handling exceptions too early isn't always a good thing.

Cancellation is another form of premature termination of a task. In WinRT, as in the PPL, cancellation requires the cooperation of two parties—the client of the operation and the operation itself. Their roles are distinct: The client requests the cancellation, and the operation acknowledges the request—or not. Because of a natural race between the client and the operation, the cancellation request isn't guaranteed to succeed.

In PPL, these two roles are represented by the two types, the cancellation_token_source and the cancellation_token. An instance of the former is used to request the cancellation by calling the cancel method on it. An instance of the latter is instantiated from the cancellation_token_source and passed as the last parameter into the constructor of the task; the then method; or in the lambda of the create_async method.

Inside the task's body, the implementation can poll the cancellation request by calling the is_task_cancellation_requested method, and acknowledge the request by calling the cancel_current_task method. Because the cancel_current_task method throws an exception under the covers, some resource cleanup is appropriate before calling cancel_current_task. **Figure 9** shows an example.

Notice that many tasks can be canceled by the same cancellation_token_source. This is very convenient when working with chains and graphs of tasks. Instead of canceling every task individually, you can cancel all the tasks governed by a given cancellation_token_source. Of course, there's no guarantee that any of the tasks will actually respond to the cancellation request. Such tasks will complete, but their normal (value-based) continuations will not run. The error-handling continuations will run, but an attempt to get the value from the antecedent task will result in the task_canceled exception.

Finally, let's look at using cancellation tokens on the production side. The lambda of the create_async method can take a cancellation_token parameter, poll it using the is_canceled method, and cancel the operation in response to the cancellation request:

```
IAsyncAction^ action = create_async( []
(cancellation_token ct) {
  while (!ct.is_canceled()); // spin until
canceled
  cancel_current_task();
});
...
action->Cancel();
```

Notice how in the case of the task continuation, it's the then method that takes the cancellation token, whereas in the case of create_async, the cancellation token is passed into the lambda. In the latter case, cancellation is initiated by calling the cancel method on the resulting asynchronous interface, and that gets plumbed by the PPL into a cancellation request through the cancellation token.

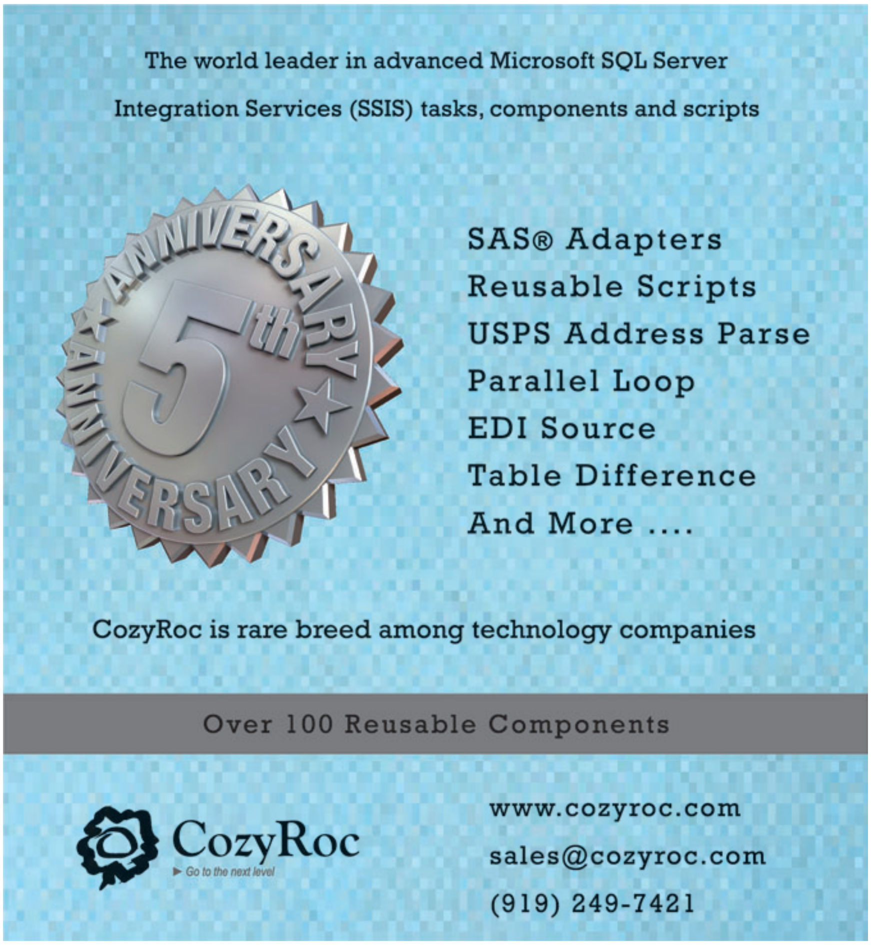
Wrapping Up

As Tony Hoare once quipped, we need to teach our programs to "wait faster." And yet, wait-free asynchronous programming remains difficult to master and its benefits are not immediately obvious, so developers shun it.

In Windows 8, all blocking operations are asynchronous, and if you're a C++ programmer, PPL makes asynchronous programming quite palatable. Embrace the world of asynchrony, and teach your programs to wait faster! ■

ARTUR LAKSBERG *leads a group of developers working on the Parallel Patterns Library and the Concurrency Runtime at Microsoft. In the past, he has worked on the C++ compiler front end and was involved in the implementation of the Axum programming language. Artur can be reached at arturl@microsoft.com.*

THANKS to the following technical experts for reviewing this article: *Genevieve Fernandes and Krishnan Varadarajan*



Building a Massively Scalable Platform for Consumer Devices on Windows Azure

Bruno Terkaly and Ricardo Villalobos

This article is about scalability and interoperability, two characteristics that are required in architectures to support the diversity of today's popular mobile platforms, which potentially have millions of users. **Figure 1** depicts this diversity, a common—yet challenging—scenario for today's developers. Supplying Web-based services to mobile devices is a daunting task, requiring distinct and diverse tooling, languages and IDEs. Beyond this diversity is the need for elastic scale—in terms of available Web services and for data that can reach terabytes in size.

Developers need to scale their Web applications in two different dimensions. The first dimension is compute, which simply boils down to the number of Web service instances made available by the hosting provider to respond to mobile Web requests. The second dimension is scalable data: Some cloud platforms offer scalable data through dedicated storage services, letting developers scale terabytes of data to millions of mobile users and effortlessly partition it across multiple servers, resulting in fast performance, redundancy and support for petabytes of capacity.

This article discusses:

- Using the Windows Azure Portal to provision a RESTful Web service
- Building the Web service
- Deploying the Web service
- Consuming the Web service

Technologies discussed:

Windows Azure, RESTful Web services, JSON

Code download available at:

bit.ly/syTize

To support communication to as many diverse clients as possible, an interoperable approach is crucial. Everything from data formats to network protocols needs careful consideration. A solution must minimize custom coding and leverage open standards to the greatest extent possible.

We're using RESTful Web services hosted in Windows Azure—the Microsoft cloud platform—in this article to solve both the interoperability challenges and the elastic scale problem.

A reference architecture based on RESTful Web services is depicted in **Figure 2**. RESTful architectures are interoperable because they're developed alongside HTTP/1.x and provide consistent communication across a vast array of clients. The architecture alternative to REST is SOAP. We chose not to use SOAP because it has larger, slower data payloads and additional complexities.

Figure 1 **A Diverse Set of Mobile Technologies Is a Challenge for Developers**

Application Type	Platform	Development Environment	Language
Mobile	Windows Phone	Visual Studio	C#
Mobile	Android	Eclipse	Java
Mobile	iOS	Xcode	Objective-C
Cloud-Based Web Server	Windows Azure	Visual Studio	C#

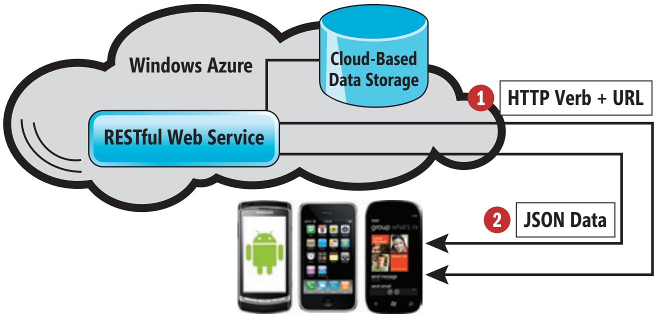


Figure 2 **A Solution Based on Open Standards**

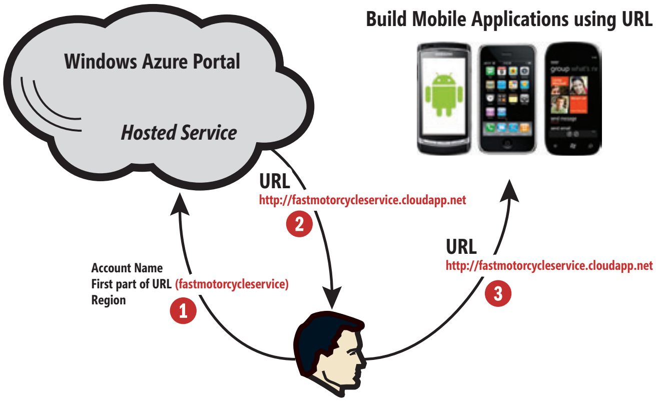

Windows Azure Portal
Hosted Service
Build Mobile Applications using URL
URL
http://fastmotorcycleservice.cloudapp.net
2
Account Name
First part of URL (fastmotorcycleservice)
Region
1
URL
http://fastmotorcycleservice.cloudapp.net
3

Figure 3 **Provisioning the Windows Azure RESTful Web Service**

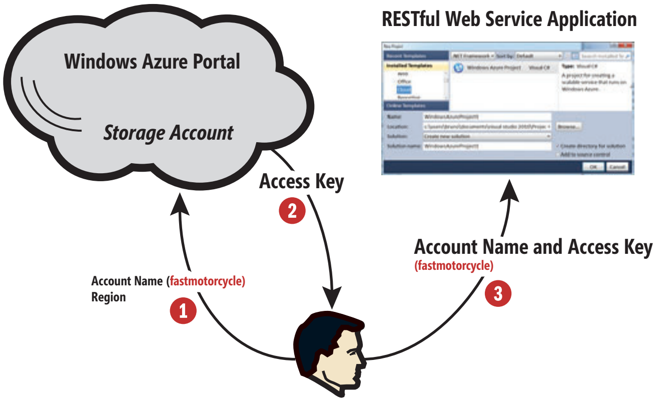

Windows Azure Portal
Storage Account
RESTful Web Service Application
Access Key
2
Account Name (fastmotorcycle)
Region
1
Account Name and Access Key
(fastmotorcycle)
3

Figure 4 **Provisioning the Windows Azure Storage Account**

Windows Azure makes it easy to increase and decrease scale on demand. By simply changing a number—the "Instance Count"—through either the Windows Azure Portal or a management API, you can scale RESTful Web services almost effortlessly to meet any level of demand.

Our implementation uses JSON (and not XML) as the data format because it's compact and widely supported. XML suffers from larger payloads.

Although many vendors offer cloud hosting solutions for RESTful Web services, Windows Azure has some advantages. For starters, you can choose from among six highly automated datacenters in Asia, Europe and North America, including support from 24 Content Delivery Networks (CDNs), making it possible to connect to users with low latency and data locality.

Windows Azure offers an array of storage and computing options in addition to powerful developer tooling. A variety of storage mechanisms are available, from Binary Large Objects (BLOBs) to relational stores. Windows Azure also provides identity management systems, secure messaging and hybrid cloud/on-premises connectivity capabilities.

Getting Started

The remainder of this article will divide the architecture and implementation into four parts:

1. Provision an account using the Windows Azure Portal.
2. Create a Windows Azure Cloud Project and write some code to define a RESTful Web service.
3. Deploy the cloud project to the account using the Windows Azure Portal.
4. Build mobile applications for: Windows Phone, Android and iOS (iPhone/iPad).

Let's go through these steps together. The first one takes place at the Windows Azure Portal, which you can access at windows.azure.com if you have a subscription. (For more information, visit azure.com.)

Part 1: Provisioning the Web Service at the Windows Azure Portal

The two key options at the Windows Azure Portal are: New Hosted Service and New Storage Account.

Figure 3 illustrates the workflow for provisioning a "hosted service." This process will result in a URL that represents the endpoint in a Microsoft datacenter where the RESTful Web service will be deployed. Developers of Windows Phone, Android and iOS apps will need this URL to communicate with the service.

The workflow to get all this working is straightforward:

1. Log in to the Windows Azure Portal.
2. Select "New Hosted Service." Specify an account name, a URL and a region (location of datacenter).
3. Store the URL the Windows Azure Portal generates; it will be used—along with the account name—when building both the RESTful Web service and the mobile clients. The account name will also be used in Part 3.

Note: The example in this article uses the account name "fastmotorcycleservice," with the URL "http://fastmotorcycleservice.cloudapp.net."

The second task at the Windows Azure Portal is creating a Storage Account. **Figure 4** illustrates this process, including the name and location of the Windows Azure tables. Once again, it's possible to choose from among the six datacenters. It makes sense to host both the Web service and the data in the same datacenter to reduce cost and improve performance.

The workflow is similar to the "hosted service" explained previously:

1. Log in to the Windows Azure Portal.
2. Create a new storage account and provide an account name and a region.
3. Store the access key the Windows Azure Portal generates and provides, as well as the account name; they'll be required when building the RESTful Web service.

Now that Part 1 is complete, the needed Windows Azure Portal information can be used to write the RESTful Web service as well as the Windows Phone, Android and iOS applications.

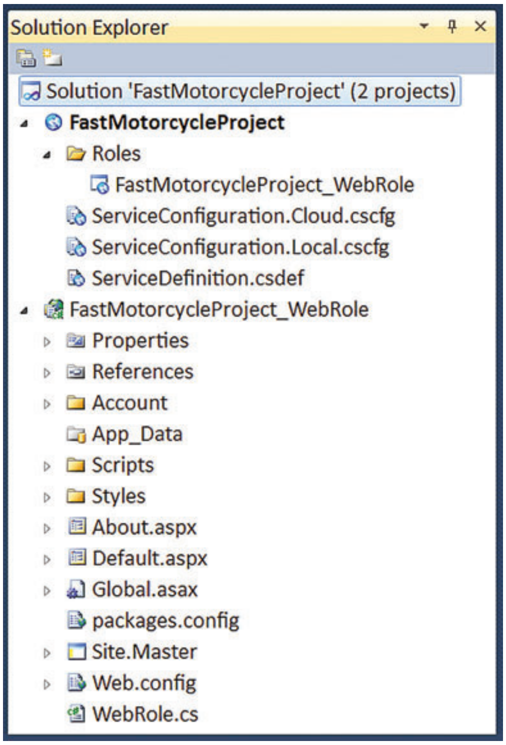
Part 2: Building the Windows Azure-Hosted RESTful Web Service

Building a RESTful Web service in Visual Studio is simple. Open Visual Studio as administrator from Start | All Programs | Microsoft Visual Studio 2010 by right-clicking the Microsoft Visual Studio 2010 shortcut and choosing "Run as administrator." From the File menu, choose New | Project.

In the New Project dialog, expand the language of preference in the Installed Templates list and select Cloud. Choose the Windows Azure Project template, set the name of the project to FastMotorcycleProject and set the location to anything convenient.

A video demonstrating these steps in detail can be found at bit.ly/VideoAzureRestfulService.

The Solution Explorer will look like **Figure 5**.



```
Solution Explorer
Solution 'FastMotorcycleProject' (2 projects)
  ◢ FastMotorcycleProject
      ◢ Roles
          FastMotorcycleProject_WebRole
        ServiceConfiguration.Cloud.cscfg
        ServiceConfiguration.Local.cscfg
        ServiceDefinition.csdef
  ◢ FastMotorcycleProject_WebRole
      ▷ Properties
      ▷ References
      ▷ Account
        App_Data
      ▷ Scripts
      ▷ Styles
      ▷ About.aspx
      ▷ Default.aspx
      ▷ Global.asax
        packages.config
      ▷ Site.Master
      ▷ Web.config
        WebRole.cs
```

Figure 5 **Creating a New Windows Azure Project**

Figure 6 shows some basic steps not covered in this article (but which are covered in the referenced video).

These steps are common to almost all Windows Azure projects. For example, it's standard practice to use a Web Role to host RESTful Web services. A DataConnectionString is needed to access the storage account defined previously at the Windows Azure Portal. Startup code is needed inside the Visual Studio project to read account names and access keys from the configuration files to use against the storage accounts.

Once the preliminary steps are complete, a RESTful Web service can be added using the WCF Service template in Visual Studio.

To add a WCF Service, right-click the FastMotorcycleProject_WebRole folder, select Add | New Item dialog and set the name of the class to FastMotorcycleService.

FastMotorcycleService.svc.cs will be generated. Replace the entire code of the class with the code shown in **Figure 7**.

The key to making this work is to know how to map different URIs and verbs to RESTful methods. For this, the WebGet and WebInvoke attributes must be added to the code in **Figure 7**.

These attributes tell the framework that the method should respond to HTTP GET requests. WebInvoke is mapped to HTTP POST by default. Also by default, the URI is determined by the name of the method (added onto the base URI of the endpoint). Some experts or REST purists might argue that our method names should not be verbs but rather nouns.

Figure 6 **Basic Steps Not Covered in This Article**

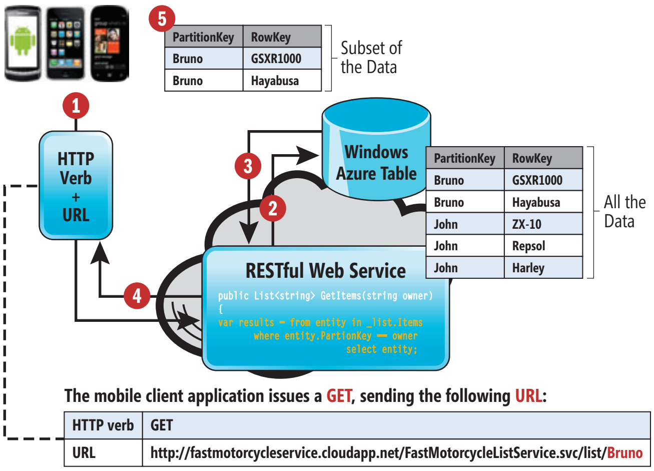
Task Covered in Video	Notes
Adding an ASP.NET Web Role	Will be used to host the RESTful Web service
Adding a DataConnectionString	Will include the account name and access key
Adding some basic startup code to initialize data	Add code to global.asax.cs to read the DataConnectionString

The WCF REST programming model shown in **Figure 8** allows customization of URIs for each method by using templates that can be set via the UriTemplate property on the WebInvoke and WebGet attributes. The model is explained in the following list, with numerals corresponding to those in **Figure 8**:

1. A mobile application uses standard HTTP to send a message request, which includes an HTTP verb plus a URL.
2. The RESTful Web service intercepts the mobile application message request (request for data) and makes a call to GetItems, passing "Bruno" as a parameter. GetItems queries for data using a LINQ query, using "Bruno" as part of the where clause.
3. Only the records in which the PartitionKey is equal to "Bruno" are returned from the Windows Azure Table Service.
4. The data is converted to JSON format (automatically) and returned to the mobile device.
5. The data is available to the mobile application. The data is used to populate a ListBox and presented to the mobile application user.

Figure 7 **FastMotorcycleListService.svc.cs**

```
[ServiceContract]
public class FastMotorcycleListService
{
  private FastMotorcycleListDataProvider _data;

  public FastMotorcycleListService()
  {
    _data = new FastMotorcycleListDataProvider();
  }

  [OperationContract]
  [WebGet(UriTemplate = "/list/{owner}", ResponseFormat =
    WebMessageFormat.Json)]
  public List<string> GetItems(string owner)
  {
    return _data.GetItems(owner);
  }

  [OperationContract]
  [WebInvoke(UriTemplate = "/list/{owner}", Method = "POST",
    RequestFormat = WebMessageFormat.Json)]
  public void AddItem(string owner, string item)
  {
    _data.AddItem(owner, item);
  }

  [OperationContract]
  [WebInvoke(UriTemplate = "/list/{owner}/{item}", Method = "DELETE")]
  public void DeleteItem(string owner, string item)
  {
    _data.DeleteItem(owner, item);
  }
}
```



PartitionKey	RowKey
Bruno	GSXR1000
Bruno	Hayabusa

PartitionKey	RowKey
Bruno	GSXR1000
Bruno	Hayabusa
John	ZX-10
John	Repsol
John	Harley

```
public List<string> GetItems(string owner)
{
var results = from entity in _list.Items
      where entity.PartionKey == owner
                     select entity;
```

The mobile client application issues a GET, sending the following URL:

HTTP verb	GET
URL	http://fastmotorcycleservice.cloudapp.net/FastMotorcycleListService.svc/list/Bruno

Figure 8 **Workflow for Mobile Application Requesting RESTful Data**

The next three classes we discuss are helper objects, which are needed to interact with the Windows Azure Table Service. FastMotorcycleListDataProvider, FastMotorcycleListItem and FastMotorcycleList are classes that abstract away storage and Windows Azure Table-specific API details from the code in **Figure 9**, allowing the application to perform Create, Read, Update and Delete (CRUD) operations with the Windows Azure Table Service.

In Visual Studio, add a new class module called FastMotorcycleListDataProvider.cs. Replace the code with the code in **Figure 9**.

Part 3: Deploying the RESTful Web Service

This is one of the areas where Windows Azure really shines. It's as simple to deploy 100 RESTful Web service instances as it is to deploy only one. Note the following list of steps:

1. In Visual Studio, right-click on FastMotorcycleProject and select Package.
2. Return back to the browser with the portal and select "Hosted Services, Storage Accounts & CDN."
3. In the top pane, select "Hosted Services."
4. In the middle pane, select the Hosted Service you previously created.
5. Right-click and select "New Production Deployment" and upload the files (FastMotorcycleProject.cspkg and ServiceConfiguration.Cloud.cscfg); these were generated in the first step.

Part 4: Consuming the RESTful Web Service from Mobile Applications

Now we'll discuss consuming the RESTful Web services from various mobile applications. This section is meant to highlight the interoperability this approach provides.

The JSONKit (github.com/johnezang/JSONKit) makes interacting with the RESTful Web service from iOS devices easier. With a few lines of code, it's possible to call the RESTful Web service, download the JSON-formatted data, convert it to a more usable format and attach the converted data to a Table View control, used by iPhone or iPad applications (see **Figure 10**).

Developing for Android involves the Java programming language, which has been around for a long time and can natively

Figure 9 **The FastMotorcycleListDataProvider, FastMotorcycleListItem and FastMotorcycleList Classes**

```
public class FastMotorcycleListDataProvider
{
  private FastMotorcycleList _list;

  public FastMotorcycleListDataProvider()
  {
    string configValue = RoleEnvironment.GetConfigurationSettingValue(
      "DataConnectionString");
    var account = CloudStorageAccount.Parse(configValue);

    _list = new FastMotorcycleList(account.TableEndpoint.ToString(),
                                   account.Credentials);
  }

  public List<string> GetItems(string owner)
  {
    var results = from entity in _list.Items
                  where entity.PartitionKey == owner
                  select entity;

    var list = new List<string>();
    foreach (var item in results)
    {
      list.Add(item.RowKey);
    }

    return list;
  }

  public void AddItem(string owner, string item)
  {
    _list.AddObject("FastBikes", new FastMotorcycleListItem(owner, item));
    _list.SaveChanges();
  }

  public void DeleteItem(string owner, string item)
  {
    var entity = (from i in _list.Items
                  where i.PartitionKey == owner
                  && i.RowKey == item
                  select i).Single();

    _list.DeleteObject(entity);
    _list.SaveChanges();
  }
}

public class FastMotorcycleListItem : TableServiceEntity
{
  public FastMotorcycleListItem()
  {
  }

  public FastMotorcycleListItem(string partitionKey, string rowKey)
    : base(partitionKey, rowKey)
  {
  }
}

public class FastMotorcycleList : TableServiceContext
{
  public FastMotorcycleList(string baseAddress,
    StorageCredentials storageCredentials)
    : base(baseAddress, storageCredentials)
  {
  }

  public DataServiceQuery<FastMotorcycleListItem> Items
  {
    get
    {
      return this.CreateQuery<FastMotorcycleListItem>("FastBikes");
    }
  }
}
```

parse JSON data. **Figure 11** shows an example. The Windows Phone SDK includes native support to call RESTful Web services and process the JSON-formatted data. The SDK makes it easy to process JSON data with DataContractJsonSerializer. **Figure 12** shows an example. Finally, if you'd like to see a more robust toolkit for developing for Android and iOS, you can visit this Microsoft-sanctioned link: github.com/microsoft-dpe.

Figure 10 **Objective-C Code That Parses JSON Data**

```
NSString *username = @"Bruno"; // Gets passed to the RESTful Web Service

NSString *serviceUri = "http://your_hosted_service_name.cloudapp.net/"+
  "FastMotorcycleListService.svc/list/";
// Build the service URI (will point to our RESTful Web service
NSString *url = [NSString stringWithFormat:@"%@%@", serviceUri, username];

// Retrieve the data in the form of a JSON array
NSData *json = [NSData dataWithContentsOfURL:[NSURL URLWithString:url]];

// Convert from JSON array to NSArray
// This allows us to populate the table view more easily
NSArray *itemArray = [json objectFromJSONData];

// Assign the array to the TableView
// fastbikes is the name of our TableView control
self.fastbikes = [[NSMutableArray alloc] initWithArray:itemArray];
```

Figure 11 **Android Code That Parses JSON Data**

```
// HttpClient used to talk to Web service
HttpClient httpclient = new DefaultHttpClient();

String url =
  "http://your_hosted_service_name.cloudapp.net/"+
  "FastMotorcycleListService.svc/list/Bruno";
// This will be the array we need to convert
// We get the data from the Web service
JSONArray listItems = null;
String jason = null;

// Set up the RESTful call to 'GET' the data
HttpGet request_http_get = new HttpGet(url);


// Read the JSON data and assign it to ListView
try
{
  // Fill a response object using a request
  HttpResponse response_http_get = httpclient.execute(request_http_get);

  // Length represents the number of data items returned
  // by RESTful Web service
  long length = response_http_get.getEntity().getContentLength();

  // "entity" ends up being the data coming back from Web server
  HttpEntity entity = response_http_get.getEntity();

  // Read the bytes, one byte at a time
  InputStream stream = entity.getContent();

  // Allocate a series of bytes
  byte[] buffer = new byte[(int) length];

  // Read bytes from RESTful Web service
  // After this loop, we end up with something like ->
  // ["busa","gxr1000","ninja250"]
  for (int i = 0; i < length; i++)
  {
    buffer[i] = (byte) stream.read();
  }
  // Create an array of strings
  jason = new String(buffer);
  // Convert to JSON array for Android ListBox
  // listItems ends up being a three-element JSON array (see "busa")
  listItems = new JSONArray(jason);
  }
  catch (Exception e)
  {
    System.out.println(e);
  }
```

Figure 12 **C# Code That Parses JSON Data**

```
private void LoadList()
{
  string uri =
    @"http://your_hosted_service_name.cloudapp.net/"+
    "FastMotorcycleListService.svc/list/Bruno";
  var webRequest = (HttpWebRequest)WebRequest.Create(uri);
  webRequest.Method = "GET";

  try
  {
    webRequest.BeginGetResponse(new AsyncCallback((result) =>
    {
      var webResponse =
        (HttpWebResponse)webRequest.EndGetResponse(result);

      if (webResponse.StatusCode == HttpStatusCode.OK)
      {
        var jsonDeserializer =
          new DataContractJsonSerializer(typeof(List<string>));
        List<string> items =
          (List<string>)jsonDeserializer.ReadObject(
          webResponse.GetResponseStream());

        shoppingListBox.Dispatcher.BeginInvoke(new Action(() =>
        {
          shoppingListBox.Items.Clear();
          foreach (var item in items)
          {
            shoppingListBox.Items.Add(item);
          }
        }));
      }

    }), null);
  }
  catch
  {
     // Ignored
  }
}
```

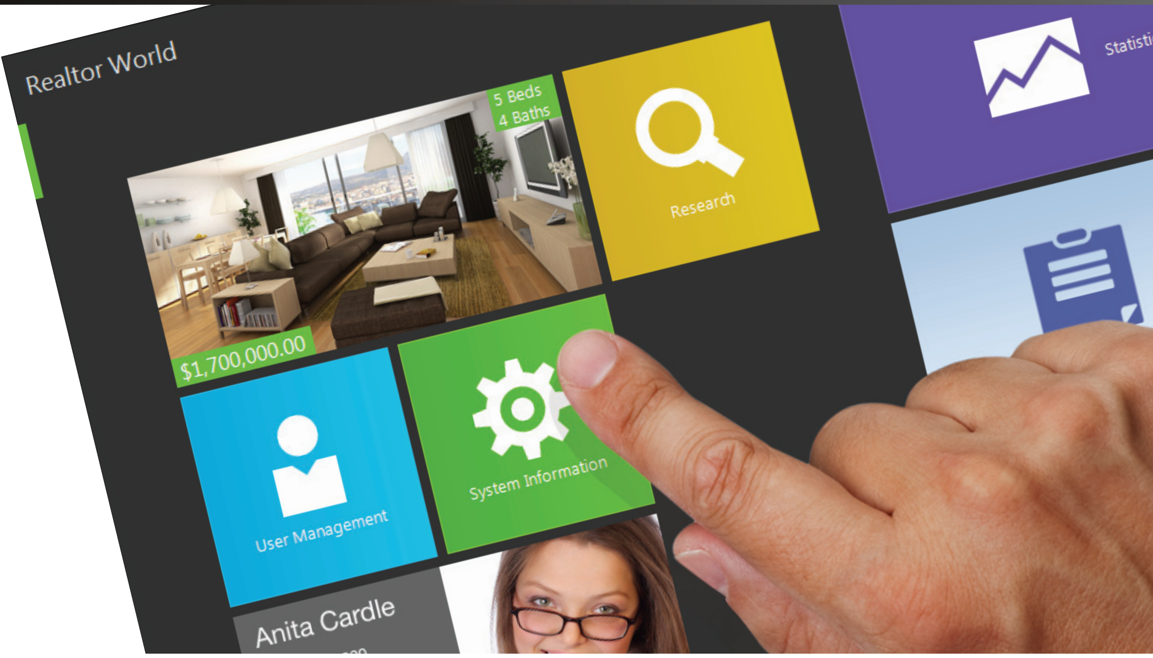
Access to Entire Spectrum of Devices

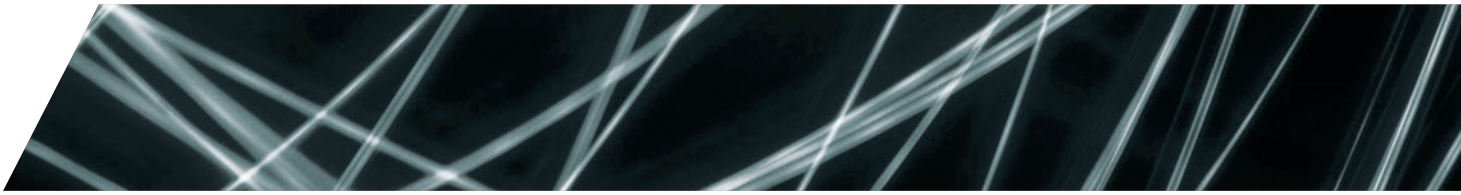
Because Windows Azure-hosted RESTful Web services are based on HTTP, any client application that supports this protocol is capable of communicating with them. This opens up a wide spectrum of devices for developers, because the majority of devices fall into this category. Although we covered mobile platforms in this article, JavaScript implementations such as jQuery are also capable of consuming RESTful Web services. Regardless of the path mobile platforms take in regard to UI diversity, it will always make sense to build on simple, open, HTTP-based Web service architectures. ■

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

Ricardo Villalobos *is a seasoned software architect with more than 15 years of experience designing and creating applications for companies in the supply chain management industry. Holding different Microsoft certifications, as well as an MBA in Supply Chain Management from the University of Dallas, he joined Microsoft in 2010 as a Windows Azure architect evangelist.*

Thanks to the following technical experts for reviewing this article: *Reza Alizadeh and Wade Wegner*


Realtor World
5 Beds
4 Baths
Research
Statisti
$1,700,000.00
User Management
System Information
Anita Cardle



The Features and Foibles of ASP.NET MVC Model Binding

Jess Chadwick

ASP.NET MVC model binding simplifies controller actions by introducing an abstraction layer that automatically populates controller action parameters, taking care of the mundane property mapping and type conversion code typically involved in working with ASP.NET request data. Though model binding seems simple, it's actually a relatively complex framework composed of a number of parts that work together to create and populate the objects that your controller actions require.

This article will take you deep into the heart of the ASP.NET MVC model binding subsystem, showing each layer of the model binding framework and the various ways you can extend the model binding logic to meet your application's needs. Along the way, you'll see a few frequently overlooked model binding techniques as well as how to avoid some of the most common model binding mistakes.

This article discusses:

- Model binding basics
- Binding to complex objects
- Examining parts of the framework
- Recursive model binding
- Model binding limitations
- Using custom attributes

Technologies discussed:

ASP.NET MVC

Model Binding Basics

To understand what model binding is, first take a look at a typical way to populate an object from request values in an ASP.NET application, shown in **Figure 1**.

Then compare the action in **Figure 1** with **Figure 2**, which leverages model binding to produce the same result.

Though the two examples both achieve the same thing—a populated Product instance—the code in **Figure 2** relies on ASP.NET MVC to convert the values from the request into strongly typed values. With model binding, controller actions can be focused on providing business value and avoid wasting time with mundane request mapping and parsing.

Binding to Complex Objects

Although model binding to even simple, primitive types can make a pretty big impact, many controller actions rely on more than just a couple of parameters. Luckily, ASP.NET MVC handles complex types just as well as primitive types.

The following code takes one more pass at the Create action, skipping the primitive values and binding directly to the Product class:

```
public ActionResult Create(Product product)
{
  // ...
}
```

Once again, this code produces the same result as the actions in **Figure 1** and **Figure 2**, only this time no code was involved at all—the complex ASP.NET MVC model binding eliminated all of the boilerplate code required to create and populate a new Product instance. This code exemplifies the true power of model binding.

Decomposing Model Binding

Now that you've seen model binding in action, it's time to break down the pieces that make up the model binding framework.

Model binding is broken down into two distinct steps: collecting values from the request and populating models with those values. These steps are accomplished by value providers and model binders, respectively.

Value Providers

ASP.NET MVC includes value provider implementations that cover the most common sources of request values such as querystring parameters, form fields and route data. At run time, ASP.NET MVC uses the value providers registered in the ValueProviderFactories class to evaluate request values that the model binders can use.

By default, the value provider collection evaluates values from the various sources in the following order:

1. Previously bound action parameters, when the action is a child action
2. Form fields (Request.Form)
3. The property values in the JSON Request body (Request.InputStream), but only when the request is an AJAX request
4. Route data (RouteData.Values)
5. Querystring parameters (Request.QueryString)
6. Posted files (Request.Files)

> It's pretty easy to create custom value providers, but be cautious in doing so.

The value providers collection, like the Request object, is really just a glorified dictionary—an abstraction layer of key/value pairs that model binders can use without needing to know where the data came from. However, the value provider framework takes this abstraction a step further than the Request dictionary, giving you complete control over how and where the model binding framework gets its data. You can even create your own custom value providers.

Custom Value Providers

The minimum requirement to create a custom value provider is pretty straightforward: Create a new class that implements the System.Web.Mvc.ValueProviderFactory interface.

For example, **Figure 3** demonstrates a custom value provider that retrieves values from the user's cookies.

Notice how simple the CookieValueProviderFactory is. Instead of building a brand-new value provider from the ground up, the CookieValueProviderFactory simply retrieves the user's cookies and leverages the NameValueCollectionValueProvider to expose those values to the model binding framework.

After you've created a custom value provider, you'll need to add it to the list of value providers via the ValueProviderFactories.Factories collection:

```
var factory = new CookieValueProviderFactory();
ValueProviderFactories.Factories.Add(factory);
```

Figure 1 **Retrieving Values Directly from the Request**

```
public ActionResult Create()
{
  var product = new Product() {
    AvailabilityDate = DateTime.Parse(Request["availabilityDate"]),
    CategoryId = Int32.Parse(Request["categoryId"]),
    Description = Request["description"],
    Kind = (ProductKind)Enum.Parse(typeof(ProductKind),
                                   Request["kind"]),
    Name = Request["name"],
    UnitPrice = Decimal.Parse(Request["unitPrice"]),
    UnitsInStock = Int32.Parse(Request["unitsInStock"]),
  };
  // ...
}
```

It's pretty easy to create custom value providers, but be cautious in doing so. The set of value providers that ASP.NET MVC ships out of the box exposes most of the data available in the HttpRequest (with the exception of cookies, perhaps) pretty well and generally provides enough data to satisfy most scenarios.

To determine whether creating a new value provider is the right thing to do for your particular scenario, ask the following question: Does the set of information provided by the existing value providers contain all the data I need (albeit perhaps not in the proper format)?

If the answer is no, then adding a custom value provider is probably the right way to address the void. However, when the answer is yes—as it usually is—consider how you can fill in the missing pieces by customizing the model binding behavior to access the data being provided by the value providers. The rest of this article shows you how to do just that.

The main component of the ASP.NET MVC model binding framework responsible for creating and populating models using values provided by value providers is called the model binder.

Default Model Binder

The ASP.NET MVC framework includes default model binder implementation named the DefaultModelBinder, which is designed to effectively bind most model types. It does this by using relatively simple and recursive logic for each property of the target model:

1. Examine the value providers to see if the property was discovered as a simple type or a complex type by checking to see if the property name is registered as a prefix. Prefixes are simply the HTML form field name "dot notation" used to represent whether a value is a property of a complex object. The

Figure 2 **Model Binding to Primitive Values**

```
public ActionResult Create(
  DateTime availabilityDate, int categoryId,
    string description, ProductKind kind, string name,
    decimal unitPrice, int unitsInStock
  )
{
  var product = new Product() {
    AvailabilityDate = availabilityDate,
    CategoryId = categoryId,
    Description = description,
    Kind = kind,
    Name = name,
    UnitPrice = unitPrice,
    UnitsInStock = unitsInStock,
  };

  // ...
}
```

prefix pattern is [ParentProperty].[Property]. For example, the form field with the name UnitPrice.Amount contains the value for the Amount field of the UnitPrice property.

2. Get the ValueProviderResult from the registered value providers for the property's name.
3. If the value is a simple type, try to convert it to the target type. The default conversion logic leverages the property's TypeConverter to convert from the source value of type string to the target type.
4. Otherwise, the property is a complex type, so perform a recursive binding.

Recursive Model Binding

Recursive model binding effectively starts the whole model binding process over again but uses the name of the target property as the new prefix. Using this approach, the DefaultModelBinder is able to traverse entire complex object graphs and populate even deeply nested property values.

To see recursive binding in action, change Product.UnitPrice from a simple decimal type to the custom type Currency. **Figure 4** shows both classes.

With this update in place, the model binder will look for the values named UnitPrice.Amount and UnitPrice.Code to populate the complex Product.UnitPrice property.

The DefaultModelBinder recursive binding logic can effectively populate even the most complex object graphs. So far, you've seen a complex object that resided only one level deep in the object hierarchy, which the DefaultModelBinder handled with ease. To demonstrate the true power of recursive model binding, add a new property named Child to Product with the same type, Product:

```
public class Product {
  public Product Child { get; set; }
  // ...
}
```

Then, add a new field to the form and—applying the dot notation to indicate each level—create as many levels as you'd like. For example:

```
<input type="text" name="Child.Child.Child.Child.Child.Child.Name"/>
```

This form field will result in six levels of Products! For each level, the DefaultModelBinder will dutifully create a new Product instance and dive right into binding its values. When the binder is all done, it will have created an object graph that looks like the code in **Figure 5**.

Even though this contrived example sets the value of just a single property, it's a great demonstration on how the DefaultModelBinder recursive model binding functionality allows it to support some very complex object graphs right out of the box. With recursive model binding, if you can create a form field name to represent the value to populate, it doesn't matter where in the object hierarchy that value lives—the model binder will find it and bind it.

Where Model Binding Seems to Fall Down

It's true: There are some models that the DefaultModelBinder simply won't be able to bind. However, there are also quite a few scenarios in which the default model binding logic may not seem to work but in fact works just fine as long as you use it appropriately.

Following are a few of the most common scenarios that developers often assume the DefaultModelBinder can't handle and how you can implement them using the DefaultModelBinder and nothing else.

Figure 3 **Custom Value Provider Factory that Inspects Cookie Values**

```
public class CookieValueProviderFactory : ValueProviderFactory
{
  public override IValueProvider GetValueProvider
  (
    ControllerContext controllerContext
  )
  {
    var cookies = controllerContext.HttpContext.Request.Cookies;

    var cookieValues = new NameValueCollection();
    foreach (var key in cookies.AllKeys)
    {
      cookieValues.Add(key, cookies[key].Value);
    }

    return new NameValueCollectionValueProvider(
      cookieValues, CultureInfo.CurrentCulture);
  }
}
```

Figure 4 **Product Class with Complex Unitprice Property**

```
public class Product
{
  public DateTime AvailabilityDate { get; set; }
  public int CategoryId { get; set; }
  public string Description { get; set; }
  public ProductKind Kind { get; set; }
  public string Name { get; set; }
  public Currency UnitPrice { get; set; }
  public int UnitsInStock { get; set; }
}

public class Currency
{
  public float Amount { get; set; }
  public string Code { get; set; }
}
```

Figure 5 **An Object Graph Created from Recursive Model Binding**

```
new Product {
  Child = new Product {
    Child = new Product {
      Child = new Product {
        Child = new Product {
          Child = new Product {
            Child = new Product {
              Name = "MADNESS!"
            }
          }
        }
      }
    }
  }
}
```

Figure 6 **Binding to an Interface**

```
public interface IProduct
{
  DateTime AvailabilityDate { get; }
  int CategoryId { get; }
  string Description { get; }
  ProductKind Kind { get; }
  string Name { get; }
  decimal UnitPrice { get; }
  int UnitsInStock { get; }
}

public ActionResult Create(IProduct product)
{
  // ...
}
```

Complex Collections The out-of-the-box ASP.NET MVC value providers treat all request field names as if they're form post values. Take, for example, a collection of primitive values in a form post, in which each value requires its own unique index (whitespace added for readability):

```
MyCollection[0]=one &
MyCollection[1]=two &
MyCollection[2]=three
```

JSON requests must adhere to the form post naming syntax.

The same approach can also be applied to collections of complex objects. To demonstrate this, update the Product class to support multiple currencies by changing the UnitPrice property to a collection of Currency objects:

```
public class Product : IProduct
{
  public IEnumerable<Currency> UnitPrice { get; set; }

  // ...
}
```

With this change, the following request parameters are required to populate the updated UnitPrice property:

```
UnitPrice[0].Code=USD &
UnitPrice[0].Amount=100.00 &

UnitPrice[1].Code=EUR &
UnitPrice[1].Amount=73.64
```

Pay close attention to the naming syntax of the request parameters required to bind collections of complex objects. Notice the indexers used to identify each unique item in the area, and that each property for each instance must contain the full, indexed reference to that instance. Just keep in mind that the model binder expects property names to follow the form post naming syntax, regardless of whether the request is a GET or a POST.

Though it's somewhat counterintuitive, JSON requests have the same requirements—they, too, must adhere to the form post naming syntax. Take, for example, the JSON payload for the previous UnitPrice collection. The pure JSON array syntax for this data would be represented as:

```
[
  { "Code": "USD", "Amount": 100.00 },
  { "Code": "EUR", "Amount": 73.64 }
]
```

However, the default value providers and model binders require the data to be represented as a JSON form post:

```
{
  "UnitPrice[0].Code": "USD",
  "UnitPrice[0].Amount": 100.00,

  "UnitPrice[1].Code": "EUR",
  "UnitPrice[1].Amount": 73.64
}
```

The complex object collection scenario is perhaps one of the most widely problematic scenarios that developers run into because the syntax isn't necessarily evident to all developers. However, once you learn the relatively simple syntax for posting complex collections, these scenarios become much easier to deal with.

Generic Custom Model Binders Though the DefaultModelBinder is powerful enough to handle almost anything you throw at it, there are times when it just doesn't do what you need. When these scenarios occur, many developers jump at the chance to take advantage of the model binding framework's extensibility model and build their own custom model binder.

For example, even though the Microsoft .NET Framework provides excellent support for object-oriented principles, the DefaultModelBinder offers no support for binding to abstract base classes and interfaces. To demonstrate this shortcoming, refactor the Product class so that it derives from an interface—named IProduct—that consists of read-only properties. Likewise, update the Create controller action so that it accepts the new IProduct interface instead of the concrete Product implementation, as shown in **Figure 6**.

The updated Create action shown in **Figure 6**—while perfectly legitimate C# code—causes the DefaultModelBinder to throw the exception: "Cannot create an instance of an interface." It's quite understandable that the model binder throws this exception, considering that DefaultModelBinder has no way of knowing what concrete type of IProduct to create.

Figure 7 **ProductModelBinder—a Tightly Coupled Custom Model Binder**

```
public class ProductModelBinder : IModelBinder
{
  public object BindModel
    (
      ControllerContext controllerContext,
      ModelBindingContext bindingContext
    )
  {
    var product = new Product() {
      Description = GetValue(bindingContext, "Description"),
      Name = GetValue(bindingContext, "Name"),
  };

    string availabilityDateValue =
      GetValue(bindingContext, "AvailabilityDate");

    if(availabilityDateValue != null)
    {
      DateTime date;
      if (DateTime.TryParse(availabilityDateValue, out date))
      product.AvailabilityDate = date;
    }

    string categoryIdValue =
      GetValue(bindingContext, "CategoryId");

    if (categoryIdValue != null)
    {
      int categoryId;
      if (Int32.TryParse(categoryIdValue, out categoryId))
      product.CategoryId = categoryId;
    }

    // Repeat custom binding code for every property
    // ...

    return product;
  }

  private string GetValue(
    ModelBindingContext bindingContext, string key)
  {
    var result = bindingContext.ValueProvider.GetValue(key);
    return (result == null) ? null : result.AttemptedValue;
  }
}
```

Working Like a Machine?

ASPOSE has the Tools you Need!

.NET, Java, SharePoint, SSRS, & JasperReports

- Create
- Edit
- Convert
- Import
- Export
- Render
- Save
- Print

Aspose.Words

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

Aspose.Cells

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

Aspose.Pdf

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

Aspose.Slides

PPT, PDF, PPS, POTX, PPTX & other formats.

and many more!

Aspose.BarCode Aspose.Tasks
Aspose.Email Aspose.Diagram
Aspose.OCR

100% Standalone - No Office Automation

Scan for an exclusive 20% coupon code.

ASPOSE
Your File Format Experts

Follow us on Facebook & Twitter

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

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

Enterprise Sales – enterprise.sales@aspose.com

The simplest way to solve this issue is to create a custom model binder that implements the IModelBinder interface. **Figure 7** shows ProductModelBinder, a custom model binder that knows how to create and bind an instance of the IProduct interface.

The downside to creating custom model binders that implement the IModelBinder interface directly is that they often duplicate much of the DefaultModelBinder just to modify a few areas of logic. It's also common for these custom binders to focus on specific model classes, creating a tight coupling between the framework and the business layer and limiting reuse to support other model types.

To avoid all these issues in your custom model binders, consider deriving from DefaultModelBinder and overriding specific behaviors to suit your needs. This approach often provides the best of both worlds.

Abstract Model Binder The only problem with trying to apply model binding to an interface with the DefaultModelBinder is that it doesn't know how to determine the concrete model type. Consider the higher-level goal: the ability to develop controller actions against a non-concrete type and dynamically determine the concrete type for each request.

By deriving from DefaultModelBinder and overriding only the logic that determines the target model type, you can not only address the specific IProduct scenario, but also actually create a general-purpose model binder that can handle most other interface hierarchies as well. **Figure 8** shows an example of a general-purpose model abstract model binder.

To support model binding to an interface, the model binder must first translate the interface into a concrete type. To accomplish this, AbstractModelBinder requests the "__type__" key from the request's value providers. Leveraging value providers for this kind of data provides flexibility as far as where the "__type__" value is defined. For example, the key could be defined as part of the route (in the route data), specified as a querystring parameter or even represented as a field in the form post data.

> To support model binding to an interface, the model binder must first translate the interface into a concrete type.

Next, the AbstractModelBinder uses the concrete type name to generate a new set of metadata that describes the details of the concrete class. AbstractModelBinder uses this new metadata to replace the existing ModelMetadata property that described the initial abstract model type, effectively causing the model binder to forget that it was ever bound to a non-concrete type to begin with.

After AbstractModelBinder replaces the model metadata with all the information needed to bind to the proper model, it simply releases control back to the base DefaultModelBinder logic to let it handle the rest of the work.

The AbstractModelBinder is an excellent example that shows how you can extend the default binding logic with your own custom logic without reinventing the wheel, by deriving directly from the IModelBinder interface.

Figure 8 **A General-Purpose Abstract Model Binder**

```
public class AbstractModelBinder : DefaultModelBinder
{
  private readonly string _typeNameKey;

  public AbstractModelBinder(string typeNameKey = null)
  {
    _typeNameKey = typeNameKey ?? "__type__";
  }

  public override object BindModel
  (
    ControllerContext controllerContext,
    ModelBindingContext bindingContext
  )
  {
    var providerResult =
    bindingContext.ValueProvider.GetValue(_typeNameKey);

    if (providerResult != null)
    {
      var modelTypeName = providerResult.AttemptedValue;

      var modelType =
        BuildManager.GetReferencedAssemblies()
          .Cast<Assembly>()
          .SelectMany(x => x.GetExportedTypes())
          .Where(type => !type.IsInterface)
          .Where(type => !type.IsAbstract)
          .Where(bindingContext.ModelType.IsAssignableFrom)
          .FirstOrDefault(type =>
            string.Equals(type.Name, modelTypeName,
              StringComparison.OrdinalIgnoreCase));

      if (modelType != null)
      {
        var metaData =
        ModelMetadataProviders.Current
        .GetMetadataForType(null, modelType);

        bindingContext.ModelMetadata = metaData;
      }
    }

    // Fall back to default model binding behavior
    return base.BindModel(controllerContext, bindingContext);
  }
}
```

Model Binder Selection

Creating custom model binders is just the first step. To apply custom model binding logic in your application, you must also register the custom model binders. Most tutorials show you two ways to register custom model binders.

The Global ModelBinders Collection The generally recommended way to override the model binder for specific types is to register a type-to-binder mapping to the ModelBinders.Binders dictionary.

The following code snippet tells the framework to use the AbstractModelBinder to bind Currency models:

```
ModelBinders.Binders.Add(typeof(Currency), new AbstractModelBinder());
```

Overriding the Default Model Binder Alternatively, to replace the global default handler, you can assign a model binder to the ModelBinders.Binders.DefaultBinder property. For example:

```
ModelBinders.Binders.DefaultBinder = new AbstractModelBinder();
```

Although these two approaches work well for many scenarios, there are two more ways that ASP.NET MVC lets you register a model binder for a type: attributes and providers.

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

BEST SELLER

FusionCharts | from $195.02

InfoSoft Global
empowering human thoughts

Interactive Flash & JavaScript (HTML5) charts for web apps.

- Liven up your web applications using animated & data-driven charts
- Create AJAX-enabled charts with drill-down capabilities in minutes
- Export charts as images/PDF and data as CSV from charts itself
- Create gauges, dashboards, financial charts and over 550 maps
- Trusted by over 19,000 customers and 400,000 users in 110 countries

BEST SELLER

Spread for .NET Professional | from $1,439.04

GrapeCity PowerTools

Includes MultiRow, Stand-Alone Chart, Formula Provider and Runtime Spread Designer.

- Programmable embedded spreadsheet platform with royalty-free licensing
- Full import/export support for Excel documents
- Built-in support for Excel functions
- Flexible printing and PDF export options
- Built-in chart with hundreds of chart styles

BEST SELLER

Janus WinForms Controls Suite V4.0 | from $889.00

Janus systems

Add powerful Outlook style interfaces to your .NET applications.

- Includes Ribbon, Grid, Calendar view, Timeline and Shortcut bar
- Now features Office 2010 visual style for all controls
- Visual Studio 2010 and .NET Framework Client Profiles support
- Janus Ribbon adds Backstage Menus and Tab functionality as in Office 2010 applications
- Now features support for multiple cell selection

NEW RELEASE

TX Text Control .NET for Windows Forms/WPF | from $1,045.59

TX TEXT CONTROL
word processing components

Word processing components for Visual Studio .NET.

- Add professional word processing to your applications
- Royalty-free Windows Forms and WPF rich text box
- True WYSIWYG, nested tables, text frames, headers and footers, images, bullets, structured numbered lists, zoom, dialog boxes, section breaks, page columns
- Load, save and edit DOCX, DOC, PDF, PDF/A, RTF, HTML, TXT and XML

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

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

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

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

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

MasterCard VISA DISCOVER

GSA Schedule
Contract GS-35F-0188R

Figure 9 **A CustomModelBinderAttribute Implementation**

```
[AttributeUsage(
  AttributeTargets.Class | AttributeTargets.Enum |
  AttributeTargets.Interface | AttributeTargets.Parameter |
  AttributeTargets.Struct | AttributeTargets.Property,
  AllowMultiple = false, Inherited = false
)]
public class AbstractModelBinderAttribute : CustomModelBinderAttribute
{
  public override IModelBinder GetBinder()
  {
    return new AbstractModelBinder();
  }
}
```

Adorning Models with Custom Attributes

In addition to adding a type mapping to the ModelBinders dictionary, the ASP.NET MVC framework also offers the abstract System.Web.Mvc.CustomModelBinderAttribute, an attribute that allows you to dynamically create a model binder for each class or property to which the attribute is applied. **Figure 9** shows a CustomModelBinderAttribute implementation that creates an AbstractModelBinder.

You can then apply the AbstractModelBinderAttribute to any model class or property, like so:

```
public class Product
{
  [AbstractModelBinder]
  public IEnumerable<CurrencyRequest> UnitPrice { get; set; }
  // ...
}
```

Now when the model binder attempts to locate the appropriate binder for Product.UnitPrice, it will discover the AbstractModelBinderAttribute and use the AbstractModelBinder to bind the Product.UnitPrice property.

> Taking the time to understand ASP.NET MVC model binding and how to use it properly can have a large impact, even on the simplest of applications.

Leveraging custom model binder attributes is a great way to achieve a more declarative approach to configuring model binders while keeping the global model binder collection simple. Also, the fact that custom model binder attributes can be applied to both entire classes and individual properties means you have fine-grain control over the model binding process.

Ask the Binders!

Model binder providers offer the ability to execute arbitrary code in real time to determine the best possible model binder for a given type. As such, they provide an excellent middle ground among explicit model binder registration for individual model types, static attribute-based registration and a set default model binder for all types.

The following code shows how to create an IModelBinderProvider that provides an AbstractModelBinder for all interfaces and abstract types:

```
public class AbstractModelBinderProvider : IModelBinderProvider
{
  public IModelBinder GetBinder(Type modelType)
  {
    if (modelType.IsAbstract || modelType.IsInterface)
      return new AbstractModelBinder();

    return null;
  }
}
```

The logic dictating whether the AbstractModelBinder applies to a given model type is relatively straightforward: Is it a nonconcrete type? If so, the AbstractModelBinder is the appropriate model binder for the type, so instantiate the model binder and return it. If the type is a concrete type, then AbstractModelBinder is not appropriate; return a null value to indicate that the model binder isn't a match for this type.

An important thing to keep in mind when implementing the .GetBinder logic is that the logic will be executed for every property that's a candidate for model binding, so be sure to keep it lightweight or you can easily introduce performance issues into your application.

In order to begin using a model binder provider, add it to the list of providers maintained in the ModelBinderProviders.BinderProviders collection. For example, register the AbstractModelBinder like so:

```
var provider = new AbstractModelBinderProvider();
ModelBinderProviders.BinderProviders.Add(provider);
```

And that easily, you've added model binding support for nonconcrete types throughout your entire application.

The model binding approach makes model binding selection much more dynamic by taking the burden of determining the proper model binder away from the framework and placing it in the most appropriate place: the model binders themselves.

Key Extensibility Points

Like any other method, the ASP.NET MVC model binding allows controller actions to accept complex object types as parameters. Model binding also encourages better separation of concerns by separating the logic of populating objects from the logic that uses the populated objects.

I've explored some key extensibility points in the model binding framework that can help you leverage it to the fullest. Taking the time to understand ASP.NET MVC model binding and how to use it properly can have a large impact, even on the simplest of applications. ■

JESS CHADWICK *is an independent software consultant specializing in Web technologies. He has more than a decade of development experience ranging from embedded devices in startups to enterprise-scale Web farms at Fortune 500 companies. He's an ASPInsider, Microsoft MVP in ASP.NET, and book and magazine author. He is actively involved in the development community, regularly speaking at user groups and conferences as well as leading the NJDOTNET Central New Jersey .NET user group.*

THANKS to the following technical expert for reviewing this article: *Phil Haack*

Practical Cross-Browser HTML5 Audio and Video

John Dyer

When the HTML5 audio and video tags were first introduced, codec and browser incompatibilities made them difficult to use and unrealistic to deploy on large-scale Web sites. The tags were great for companies writing experimental code or doing cross-browser media development, but the HTML5 media API was too unreliable for general use.

Today, things are different. Browsers and JavaScript libraries have matured to the point where you can—and should—use HTML5 media as the default for any projects that will display audio and video content. Even retrofitting existing Flash and Silverlight video content for HTML5 playback has become fairly simple. In this article, I'll explore the benefits of using HTML5 for media playback, show some sample code, address some major issues that developers face and present solutions to those problems.

This article discusses:

- Simple HTML5 playback and controls
- HTML5 media attributes
- Issues with HTML5 Media
- HTML5 video and audio JavaScript libraries

Technologies discussed:

HTML5 Video and Audio, JavaScript, MediaElement.js

Code download available at:

code.msdn.microsoft.com/mag201202HTML5

Benefits of HTML5 Media

The advantage of using HTML5 for media is that you can leverage your HTML, CSS and JavaScript skills rather than learning Flash or Silverlight. If you can create buttons in HTML and control them with JavaScript, you already know all you need to develop HTML5 media. HTML5 media has built-in support for captions and subtitles using the new track element, and proposals for additional features—such as direct byte access for video manipulation—are already being considered.

Moreover, media that uses HTML5 video and audio performs better than media played through plug-ins such as Flash or Silverlight, resulting in longer battery life and smoother playback. In addition, mobile devices running Windows Phone 7.5, Apple iOS and Android (as well as the Metro-style browser in Windows 8) support media playback only through HTML5 video and audio. Some Android devices include Flash, but Adobe has recently discontinued its mobile Flash efforts, which means that HTML5 will be the only way to play media on mobile devices in the future.

Simple HTML5 Playback and Controls

In the days of using Flash or Silverlight for video playback, the simplest possible markup to display a video (video.mp4 in this case) would have looked something like this:

```
<object width="640" height="360" classid="clsid:d27cdb6e-ae6d-11cf-
96b8-444553540000" codebase="http://fpdownload.macromedia.com/pub/
shockwave/cabs/flash/swflash.cab#version=8,0,0,0">
  <param name="src" value="player.swf?file=video.mp4">
  <embed src="player.swf?file=video.mp4" width="640"
    height="360"></embed>
</object>
```

Compare those nested object, param and embed tags with this HTML5 video tag used to play the same h.264-encoded video:

```
<video src="video.mp4" controls></video>
```

It's much simpler—just plain HTML that needs very little explanation. When the browser has downloaded enough of a video to determine its native height and width, it resizes the video accordingly. But, just as with img tags, it's always best to specify the height and width attributes so that the page doesn't need to reflow. You can also use the style attribute to specify px or % width and height values (I'll use both in the examples that follow).

The one attribute I added is *controls*. This tells the browser to display its own built-in playback control bar, which usually includes a play/pause toggle, a progress indicator and volume controls. Controls is a Boolean attribute, which means it doesn't need to have a value assigned to it. For a more XHTML-like syntax you could write *controls="controls"*, but the browser always considers controls to be false if it's not present and true if it is present or assigned a value.

Figure 1 **Controlling Video Playback**

```
<script>
// Wrap the code in a function to protect the namespace
(function() {
// Find the DOM objects
var  video = document.getElementById("video1"),
  playBtn = document.getElementById("video1-play"),
  muteBtn = document.getElementById("video1-mute"),
  current = document.getElementById("video1-current"),
duration = document.getElementById("video1-duration");

// Toggle the play/pause state
playBtn.addEventListener("click", function() {
  if (video.paused || video.ended) {
    video.play();
    playBtn.value = "Pause";
  } else {
    video.pause();
    playBtn.value = "Play";
  }
}, false);

// Toggle the mute state
muteBtn.addEventListener("click", function() {
  if (video.muted) {
    video.muted = false;
    muteBtn.value = "Mute";
  } else {
    video.muted = true;
    muteBtn.value = "Unmute";
  }
}, false);

// Show the duration when it becomes available
video.addEventListener("loadedmetadata", function() {
  duration.innerHTML = formatTime(video.duration);
}, false);

// Update the current time
video.addEventListener("timeupdate", function() {
  current.innerHTML = formatTime(video.currentTime);
}, false);

function formatTime(time) {
  var
    minutes = Math.floor(time / 60) % 60,
    seconds = Math.floor(time % 60);

  return   (minutes < 10 ? '0' + minutes : minutes) + ':' +
           (seconds < 10 ? '0' + seconds : seconds);
}

})();
</script>
```

HTML5 Media Attributes and Child Source Tags

The audio and video elements introduce several new attributes that determine how the browser will present the media content to the end user.

- **src** specifies a single media file for playback (for multiple sources with different codecs, please see the discussion below).
- **poster** is a URL to an image that will be displayed before a user presses Play (video only).
- **preload** determines how and when the browser will load the media file using three possible values: *none* means the video will not download until the user initiates playback; *metadata* tells the browser to download just enough data to determine the height, width and duration of the media; *auto* lets the browser decide how much of the video to start downloading before the user initiates playback.
- **autoplay** is a Boolean attribute used to start a video as soon as the page loads (mobile devices often ignore this to preserve bandwidth).
- **loop** is a Boolean attribute that causes a video to start over when it reaches the end.
- **muted** is a Boolean attribute specifying whether the video should start muted.
- **controls** is a Boolean attribute indicating whether the browser should display its own controls.
- **width** and **height** tell a video to display at a certain size (video only; can't be a percentage).

Timed Text Tracks

Browsers are also beginning to implement the track element, which provides subtitles, closed captions, translations and commentaries to videos. Here's a video element with a child track element:

```
<video id="video1" width="640" height="360" preload="none" controls>
  <track src="subtitles.vtt" srclang="en" kind="subtitles"
label="English subtitles">
</video>
```

In this example, I've used four of the track element's five possible attributes:

- **src** is a link to either a Web Video Timed Text (WebVTT) file or a Timed Text Markup Language (TTML) file.
- **srclang** is the language of the TTML file (such as en, es or ar).
- **kind** indicates the type of text content: subtitles, captions, descriptions, chapters or metadata.
- **label** holds the text displayed to a user choosing a track.
- **default** is a Boolean attribute that determines the startup track element.

WebVTT is a simple text-based format that begins with a single-line declaration (WEBVTT FILE) and then lists start and end times separated by the --> characters, followed by the text to display between the two times. Here's a simple WebVTT file that will display two lines of text at two different time intervals:

```
WEBVTT FILE

00:00:02.5 --> 00:00:05.1
This is the first line of text to display.

00:00:09.1 --> 00:00:12.7
This line will appear later in the video.
```

As of this writing, only Internet Explorer 10 Platform Preview and Chrome 19 support the track element, but other browsers are expected to do so soon. Some of the JavaScript libraries I discuss later add support for the track element to browsers that have not yet implemented it, but there's also a standalone track library called captionator.js (captionatorjs.com) that parses track tags, reads WebVTT and TTML (as well as SRT and YouTube SBV) files and provides a UI for browsers that don't yet have native support.

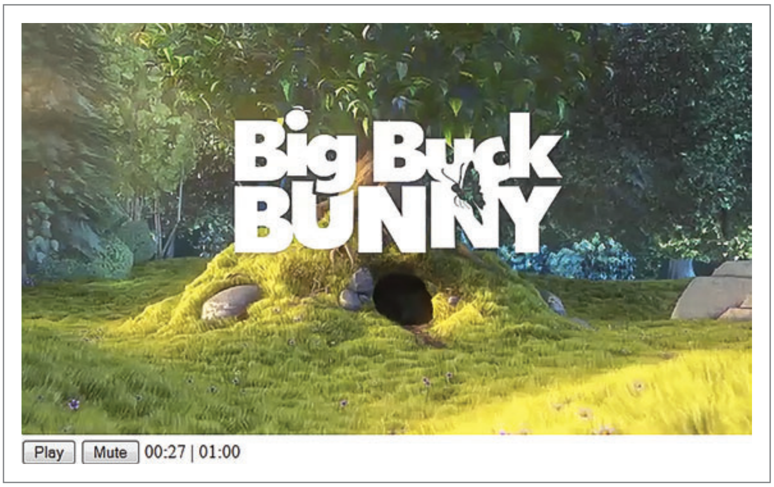


Figure 2 **A Working Video Player That Shows the Time**

Scripting HTML5 Media

Earlier, I used the controls attribute to tell the browser to display its native controls on top of the video or audio tags. The problem with this is that each browser has a different set of controls that are unlikely to match your Web site's design. To create a consistent experience, you can remove the browser's controls and then add custom buttons to the page that you control with JavaScript. You can also add event listeners to track the state of the video or audio playback. In the following example, I've removed the controls attribute and added markup underneath the video to serve as a basic control bar:

```
<video id="video1" style="width:640px; height:360px" src="video.mp4"> </video>
<div>
  <input type="button" id="video1-play" value="Play" />
  <input type="button" id="video1-mute" value="Mute" />
  <span id="video1-current">00:00</span>
  <span id="video1-duration">00:00</span>
</div>
```

The simple JavaScript in **Figure 1** will control video playback and show the current time in the video, and will create the complete working player depicted in **Figure 2** (rendered in Internet Explorer 9). (Note that in HTML5, the *type="text/javascript"* attribute is not required on the script tag.)

The code in **Figure 1** introduces the play and pause methods, the timeupdate and loadedmetadata events, and the paused, ended, currentTime and duration properties. The full HTML5 media API (w3.org/TR/html5/video.html) includes much more that can be used to build a full-fledged media player. In addition to the HTML5 media tag attributes listed earlier, HTML5 media objects have other properties accessible only via JavaScript:

- **currentSrc** describes the media file the browser is currently playing when source tags are used.
- **videoHeight** and **videoWidth** indicate the native dimensions of the video.
- **volume** specifies a value between 0 and 1 to indicate the volume. (Mobile devices ignore this in favor of hardware volume controls.)
- **currentTime** indicates the current playback position in seconds.
- **duration** is the total time in seconds of the media file.
- **buffered** is an array indicating what portions of the media file have been downloaded.
- **playbackRate** is the speed at which the video is played back (default: 1). Change this number to go faster (1.5) or slower (0.5).
- **ended** indicates whether the video has reached the end.
- **paused** is always true at startup and then false once the video has started playing.
- **seeking** indicates the browser is trying to download and move to a new position.

HTML5 media objects also include the following methods for scripting:

- **play** attempts to load and play the video.
- **pause** halts a currently playing video.
- **canPlayType(type)** detects which codecs a browser supports. If you send a type such as video/mp4, the browser will answer with *probably*, *maybe*, *no* or a blank string.
- **load** is called to load the new video if you change the src attribute.

The HTML5 media spec provides 21 events; here are some of the most common ones:

- **loadedmetadata** fires when the duration and dimensions are known.
- **loadeddata** fires when the browser can play at the current position.
- **play** starts the video when the video is no longer paused or ended.
- **playing** fires when playback has started after pausing, buffering or seeking
- **pause** halts the video.
- **ended** fires when the end of the video is reached.
- **progress** indicates more of the media file has been downloaded.
- **seeking** is true when the browser has started seeking.
- **seeked** is false when the browser has finished seeking.
- **timeupdate** fires as the media resource is playing.
- **volumechange** fires when muted or volume properties have changed.

These properties, methods and events are powerful tools for presenting users with a rich media experience, all driven by HTML, CSS and JavaScript. In the basic example in **Figure 1**, I first create variables for all of the elements on the page:

```
// Find the DOM objects
var video = document.getElementById("video1"),
  playBtn = document.getElementById("video1-play"),
  muteBtn = document.getElementById("video1-mute"),
  current = document.getElementById("video1-current"),
  duration = document.getElementById("video1-duration");
```

/n software

We didn't invent the Internet...

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

TOOLS • COMPONENTS • ENTERPRISE ADAPTERS

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

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

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

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

connectivity
powered by

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

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

Figure 3 **Codec Support in Various Browsers**

Video	IE8	IE9+	Chrome	Safari	Mobile	Firefox	Opera
MP4 (h.264/AAC)	no	yes	yes	yes	yes	no	no
WebM (VP8/Vorbis)	no	install	yes	no	no	yes	yes

Figure 4 **Audio Support in Various Browsers**

Audio	IE8	IE9+	Chrome	Safari	Mobile	Firefox	Opera
MP3	no	yes	yes	yes	yes	no	no
Ogg Theora	no	install	yes	no	no	yes	yes
WAV	no	no	maybe	yes	yes	yes	yes

Then I add a click event to my buttons to control media playback. Here I toggle the play and pause state of the video and change the label on the button:

```
// Toggle the play/pause state
playBtn.addEventListener("click", function() {
  if (video.paused || video.ended) {
      video.play();
      playBtn.value = "Pause";
  } else {
      video.pause();
      playBtn.value = "Play";
  }
}, false);
```

Finally, I add events to the media object to track its current state. Here, I listen for the timeupdate event and update the control bar to the current time of the playhead, formatting the seconds to a minutes:seconds style:

```
// Update the current time
video.addEventListener("timeupdate", function() {
  current.innerHTML = formatTime(media.currentTime);
}, false);
```

Issues with HTML5 Media

Unfortunately, getting HTML5 media to work across all browsers and devices is not quite as simple as in my example. I've already mentioned that not all browsers support the track element, and now I'll address three additional issues that you encounter when using the audio and video tags, along with solutions to overcome them. At the end of the article, I'll introduce some JavaScript libraries that wrap all of these solutions into single, easily deployable packages.

HTML5 Audio and Video Codec Support The first issue you face when developing with HTML5 media is the inconsistent support for video and audio codecs. My examples work in Internet Explorer 9 and later, Chrome and Safari, but they won't work in Firefox or Opera because although those browsers support the HTML5 video tag, they don't support the h.264 codec. Due to copyright concerns, browser vendors have split into two codec camps, and that brings us to the familiar HTML5 Media chart in **Figure 3**, showing which codecs work with which browsers.

Internet Explorer 9+, Safari, Chrome and mobile devices (iPhone, iPad, Android 2.1+ and Windows Phone 7.5+) all support the h.264 video codec, which is usually placed in an MP4 container. Firefox and Opera, in contrast, support the VP8 video codec, which is placed inside the WebM container. Chrome also supports WebM, and has pledged to remove h.264 support at some point. Internet Explorer 9+ can render WebM if the codec has been installed by the end user. Finally, Firefox, Opera and Chrome also support the Theora codec placed inside an Ogg container, but this has been largely phased out in favor of WebM (unless you need to support Firefox 3.x), so I've left it out of the chart and examples for simplicity.

For audio, the browser vendors are again split into two camps, with the first group (Internet Explorer 9, Chrome and Safari) supporting the familiar MP3 format and the second group (Firefox and Opera) supporting the Vorbis codec inside an Ogg container. Many browsers can also play the WAV file format. See **Figure 4**.

To deal with these differences, the video and audio tags support multiple child source tags, which lets browsers choose a media file they can play. Each source element has two attributes:

- **src** specifies a URL for a media file.
- **type** specifies the mimetype and optionally the specific codec of the video.

To offer both h.264 and VP8 video codecs, you'd use the following markup:

```
<video id="video1" width="640" height="360">
  <source src="video.mp4" type="video/mp4">
  <source src="video.webm" type="video/webm">
</video>
```

Note that earlier builds of iOS and Android need the MP4 file to be listed first.

This markup will work on all modern browsers. The JavaScript code will control whichever video the browser decides it can play. For audio, the markup looks like this:

```
<audio id="audio1">
  <source src="audio.mp3" type="audio/mp3">
  <source src="audio.oga" type="audio/oga">
</audio>
```

If you're hosting audio or video content on your own server, you must have the correct MIME type for each media file or many HTML5-ready browsers (such as Internet Explorer and Firefox) will not play the media. To add MIME types in IIS 7, go to the Features View, double-click MIME Types, click the Add button in the Actions pane, add the extension (mp4) and MIME type (video/mp4), and then press OK. Then do the same for the other types (webm and video/webm) you plan to use.

Supporting Older Browsers Including two media files (such as MP4 and WebM for video) makes HTML5 media work in all modern browsers. But when older browsers (such as Internet Explorer 8) encounter the video tag, they can't display the video. They will, however, render the HTML put between the opening <video> and closing </video> tags. The following example includes a message urging users to get a newer browser:

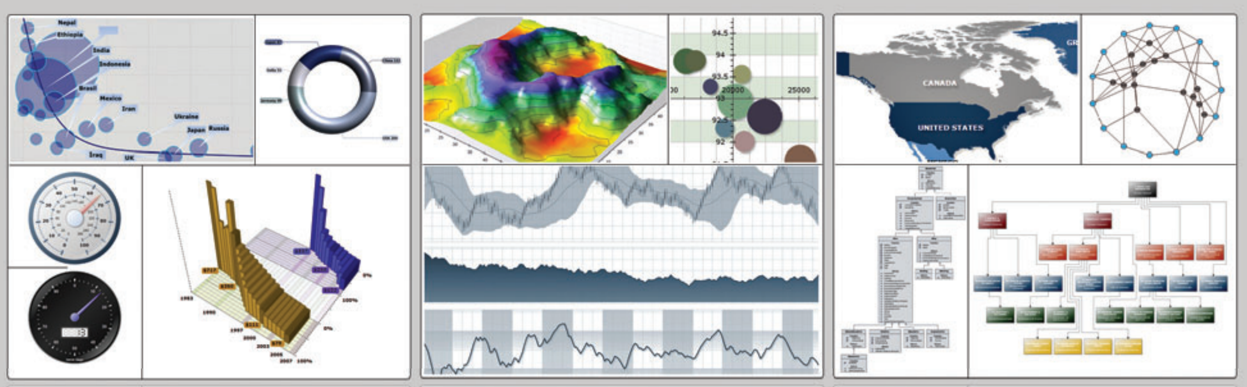
```
<video id="video1" width="640" height="360" >
  <source src="video.mp4" type="video/mp4">
  <source src="video.webm" type="video/webm">
  <p>Please update your browser</p>
</video>
```

To allow visitors with non-HTML5-ready browsers to play the video, you can provide an alternative with Flash embedded that plays the same MP4 you supply for Internet Explorer 9, Safari and Chrome, as shown in **Figure 5**.

NEVRON
visualize your success

Let us help you visualize your success

Nevron provides the essential components for the creation of advanced digital dashboards, scientific and financial applications, diagrams and MMI interfaces for a variety of .NET centric technologies.



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

DEVELOPERS

Nevron .NET Vision incorporates components that help you create enterprise grade digital dashboards, scorecards, diagrams, maps, MMI interfaces and much more.

- Chart for .NET
- Diagram for .NET
- Gauge for .NET
- Map for .NET
- User Interface for .NET

IT PROFFESIONALS

Nevron Reporting Services Vision instantly enhances your SQL Server Reporting Services 2005/2008 reports with the industry leading data visualization technology.

- Chart for SQL Server Reporting Services
- Gauge for SQL Server Reporting Services

Nevron SharePoint Vision instantly converts your SharePoint pages into advanced dashboards and reports, that unite powerful data analysis with industry leading data visualization.

- Chart for SharePoint
- Gauge for SharePoint

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

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

www.nevron.com | sales@nevron.com | 1888 201 6088

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

Figure 5 **Video Playback with Flash**

```
<video id="video1" width="640" height="360" >
  <source src="video.mp4" type="video/mp4">
  <source src="video.webm" type="video/webm">
  <object width="640" height="360" classid="clsid:
    d27cdb6e-ae6d-11cf-96b8-444553540000" codebase=
      "http://fpdownload.macromedia.com/pub/
      shockwave/cabs/flash/swflash.cab#version=8,0,0,0">
        <param name="SRC" value="player.swf?file=video.mp4">
        <embed src="player.swf?file=video.mp4" width="640"
          height="360"></embed>
        <p>Please update your browser or install Flash</p>
  </object>
</video>
```

This markup presents all browsers with some way to play back video. Browsers with neither HTML5 nor Flash will see a message urging them to upgrade. For more information on how and why this nested markup works, see Kroc Camen's "Video for Everybody" (camendesign.com/code/video_for_everybody).

This approach has some drawbacks, however. First, there's a lot of markup to maintain. Second, you have to encode and store at least two media files. And third, any HTML/JavaScript controls you add to the page will not work with the embedded Flash player. Later, I'll suggest several JavaScript libraries that can help you overcome these issues, but first, let's address one final issue.

Full-Screen Support Flash and Silverlight both include a full-screen mode that lets users watch video and other content on their entire screen. You can implement this feature by creating a simple button and tying it to an ActionScript (for Flash) or C# (for Silverlight) full-screen command.

Today's browsers have a similar full-screen mode that users can trigger with a keyboard or menu command (often F11 or Ctrl+F). But until recently, no equivalent JavaScript API allowed developers to initiate full-screen mode from a button on a page. This meant that HTML5 video could be displayed only in a "full window" that filled the browser window but not the entire screen.

In late 2011, Safari, Chrome and Firefox added support for the W3C proposed FullScreen API, which offers capabilities similar to those in Flash and Silverlight. The Opera team is currently working on implementing it, but the Internet Explorer team has, as of this writing, not yet decided whether it will implement the API. The Metro-style browser in Windows 8 will be full screen by default, but desktop Internet Explorer users will need to enter full-screen mode manually using menu options or the F11 key.

To enter full-screen mode in browsers that support it, you call the requestFullscreen method on the element to be displayed full screen. The command to exit full screen is called on the document object: document.exitFullscreen method. The W3C proposal is still a work in progress, so I won't go into more detail here, but I am tracking the state of the API on my blog: bit.ly/zlgxUA.

HTML5 Video and Audio JavaScript Libraries

A number of developers have created JavaScript libraries that make HTML5 video and audio easier by integrating all of the relevant code into a single package. Some of the best open source libraries are MediaElement.js, jPlayer, VideoJS, Projekktor, Playr and LeanBack. You'll find a complete list with feature comparison at praegnanz.de/html5video.

All you need to do is provide a video or audio tag and the library will automatically build a set of custom controls, as well as insert a Flash player for browsers that don't support HTML5 Media. The only problem is that the Flash players many libraries insert don't always look or function like the HTML5 player. This means that any HTML5 events you add won't work with the Flash player and any custom CSS you use won't be visible, either.

In my own work, I was asked to create an HTML5 video player with synchronized slides and transcripts (see online.dts.edu/player for a demo). We had an existing library of more than 3,000 h.264 video files and it was deemed unfeasible to transcode them to WebM for Firefox and Opera. We also needed to support older browsers such as Internet Explorer 8, but a separate Flash fallback wouldn't work because it wouldn't respond to events for the slides and transcripts.

To overcome these difficulties, I created one of the players mentioned previously called MediaElement.js (mediaelementjs.com). It's an open source (MIT/GLPv2) JavaScript library that includes special Flash and Silverlight players that mimic the HTML5 API. Instead of a totally separate Flash player, MediaElement.js uses Flash only to render video and then wraps the video with a JavaScript object that looks just like the HTML5 API. This effectively upgrades all browsers so they can use the video tag and additional codecs not natively supported. To start the player with a single h.264 file using jQuery, you need only the following code:

```
<video id="video1" width="640" height="360" src="video.mp4" controls></video>
<script>
jQuery(document).ready(function() {
  $("video1").mediaelementplayer();
});
</script>
```

For browsers that don't support the video tag (like Internet Explorer 8) or those that don't support h.264 (Firefox and Opera), MediaElement.js will insert the Flash (or Silverlight, depending on what the user has installed) shim to "upgrade" those browsers so they gain the HTML5 media properties and events I've covered.

For audio support, you can use a single MP3 file:

```
<audio id="audio1" src="audio.mp3" controls></audio>
```

Alternatively, you could include a single Ogg/Vorbis file:

```
<audio id="audio1" src="audio.oga" controls></audio>
```

Again, for browsers that don't support the audio tag (Internet Explorer 8) or those that don't support Ogg/Vorbis (Internet Explorer 9+ and Safari), MediaElement.js will insert a shim to "upgrade" those browsers so they all function as if they supported the codec natively. (Note: Ogg/Vorbis will not be playable on mobile devices.)

MediaElement.js also includes support for the track element, as well as native full-screen mode for browsers that have implemented the JavaScript API. You can add your own HTML5 events or track properties and it will work in every browser and mobile device.

I hope I've shown you that despite a few quirks, the HTML5 video and audio elements, especially when paired with the excellent libraries I've suggested, are easy to add to existing Web sites and should be the default for any new projects. ■

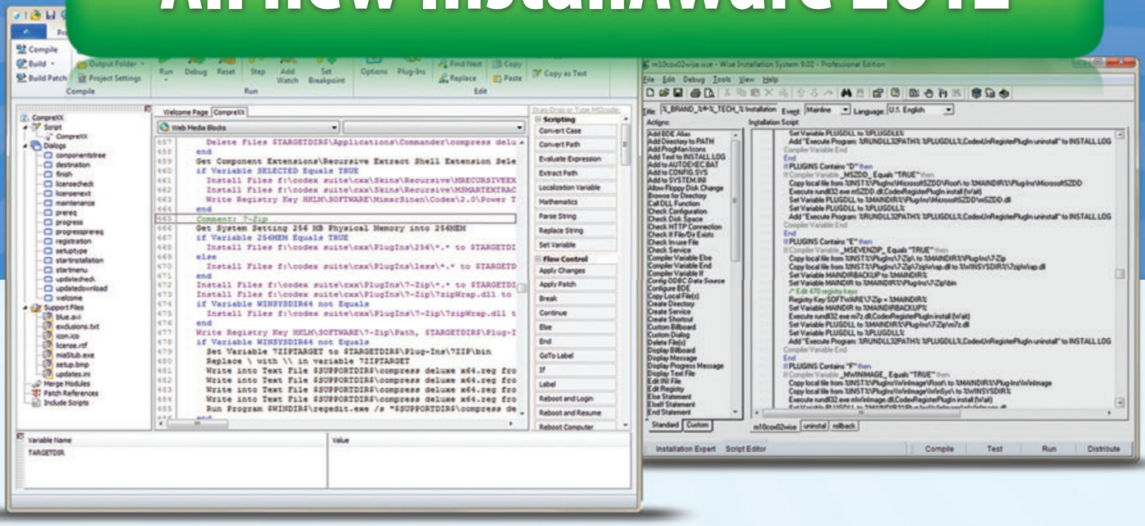
JOHN DYER *is the director of Web Development for the Dallas Theological Seminary (dts.edu). He blogs at j.hn.*

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

InstallAware

Wise people would never buy another installer if they saw InstallAware first

All new InstallAware 2012



```
    Delete Files $TARGETDIR$\Applications\Commander\compress delu
  end
  Get Component Extensions\Recursive Extract Shell Extension Sele
  if Variable SELECTED Equals TRUE
    Install Files f:\codex suite\cxx\Skins\Recursive\MRECURSIVEEX
    Install Files f:\codex suite\cxx\Skins\Recursive\MSMARTEXTRAC
    Write Registry Key HKLM\SOFTWARE\MimarSinan\Codex\2.0\Power T
  end
  Comment: 7-Zip
  Get System Setting 256 MB Physical Memory into 256MEM
  if Variable 256MEM Equals TRUE
    Install Files f:\codex suite\cxx\PlugIns\256\*.* to $TARGETDI
  else
    Install Files f:\codex suite\cxx\PlugIns\less\*.* to $TARGETD
  end
  Install Files f:\codex suite\cxx\PlugIns\7-Zip\*.* to $TARGETDI
  Install Files f:\codex suite\cxx\PlugIns\7-Zip\7zipWrap.dll to
  if Variable WINSYSDIR64 not Equals
    Install Files f:\codex suite\cxx\PlugIns\7-Zip\7zipWrap.dll t
  end
  Write Registry Key HKLM\SOFTWARE\7-Zip\Path, $TARGETDIR$\Plug-I
  if Variable WINSYSDIR64 not Equals
    Set Variable 7ZIPTARGET to $TARGETDIR$\Plug-Ins\7ZIP\bin
    Replace \ with \\ in variable 7ZIPTARGET
    Write into Text File $SUPPORTDIR$\compress deluxe x64.reg fro
    Write into Text File $SUPPORTDIR$\compress deluxe x64.reg fro
    Write into Text File $SUPPORTDIR$\compress deluxe x64.reg fro
    Write into Text File $SUPPORTDIR$\compress deluxe x64.reg fro
    Run Program $WINDIR$\regedit.exe /s "$SUPPORTDIR$\compress de
```

```
  Set Variable PLUGDLL to %PLUGDLL%
  Add "Execute Program: %RUNDLL32PATH% %PLUGDLL%,CodexUnRegisterPlugin uninstall" to INSTALL.LOG
Compiler Variable End
End
If PLUGINS Contains "D" then
If Compiler Variable _MSZDD_ Equals "TRUE" then
  Copy local file from %INST%\PlugIns\MicrosoftSZDD\Root\ to %MAINDIR%\Plug-Ins\MicrosoftSZDD
  Execute rundll32.exe mSZDD.dll,CodexRegisterPlugin install (Wait)
  Set Variable PLUGDLL to %MAINDIR%\Plug-Ins\MicrosoftSZDD\mSZDD.dll
  Set Variable PLUGDLL to %PLUGDLL%
  Add "Execute Program: %RUNDLL32PATH% %PLUGDLL%,CodexUnRegisterPlugin uninstall" to INSTALL.LOG
Compiler Variable End
End
If PLUGINS Contains "E" then
If Compiler Variable _MSEVENZIP_ Equals "TRUE" then
  Copy local file from %INST%\PlugIns\7-Zip\ to %MAINDIR%\Plug-Ins\7-Zip
  Copy local file from %INST%\PlugIns\7-Zip\7zipWrap.dll to %WINSYSDIR%\7zipWrap.dll
  Set Variable MAINDIRBACKUP to %MAINDIR%
  Set Variable MAINDIR to %MAINDIR%\Plug-Ins\7-Zip\bin
  /* Edit 470 registry keys
  Registry Key SOFTWARE\7-Zip = %MAINDIR%
  Set Variable MAINDIR to %MAINDIRBACKUP%
  Execute rundll32.exe m7z.dll,CodexRegisterPlugin install (Wait)
  Set Variable PLUGDLL to %MAINDIR%\Plug-Ins\7-Zip\m7z.dll
  Set Variable PLUGDLL to %PLUGDLL%
  Add "Execute Program: %RUNDLL32PATH% %PLUGDLL%,CodexUnRegisterPlugin uninstall" to INSTALL.LOG
Compiler Variable End
End
If PLUGINS Contains "F" then
If Compiler Variable _MWINIMAGE_ Equals "TRUE" then
  Copy local file from %INST%\PlugIns\WinImage\Root\ to %MAINDIR%\Plug-Ins\WinImage
  Copy local file from %INST%\PlugIns\WinImage\WinSys\ to %WINSYSDIR%
  Execute rundll32.exe mWinImage.dll,CodexRegisterPlugin install (Wait)
```

Migrate now from Wise and InstallShield

and receive an instant **34% discount** with an additional **10% off for MSDN readers**

Visit www.installaware.com/landing/msdn.asp for this special pricing

Get Your Windows Phone Apps into the Marketplace Faster

Cheryl Simmons

The Windows Phone SDK 7.1 includes some great tools for evaluating the adherence to certification guidelines and improving the performance of your applications that target Windows Phone 7.5, prior to submission to the marketplace. In this article, I'll walk you through using the Marketplace Test Kit and Performance Analysis tool on a sample application and show how you can use these tools to evaluate the marketplace-readiness of your application. I'll show you how to use data from the tools to make improvements that will help get it accepted into the marketplace on the first try. For more information about marketplace certification requirements, see the MSDN Library article, "Application Certification Requirements for Windows Phone" (wpdev.ms/certreq).

All of the tools used in this article are included with the Windows Phone SDK 7.1, which you can get at wpdev.ms/wpsdk71rtw.

Sample Application and Test Tools

To exercise the Marketplace Test Kit and Performance Analysis tool, I created a sample application called Examine the Stamen, a simple flower identification application. I created it with my mother in mind; she can use it to improve her flower identification skills. The application displays several small images of flowers on the start screen. A user taps a flower and the application navigates to another page, where it displays a larger picture of the selected flower. Another tap and the flower's name is displayed in a MessageBox. **Figure 1** shows the images displayed as I navigate through the application. (As a side note, I used the new screenshot tool in the Windows Phone Emulator for these screenshots. For more information, see the MSDN Library article, "How to: Create Screenshots for Windows Phone Marketplace" (wpdev.ms/rYoZKP).

This article discusses:

- Using the Marketplace Test Kit
- Running monitored tests
- Using the Windows Phone Performance Analysis tool
- Finding and fixing problems
- Rerunning tests after fixes

Technologies discussed:

Windows Phone

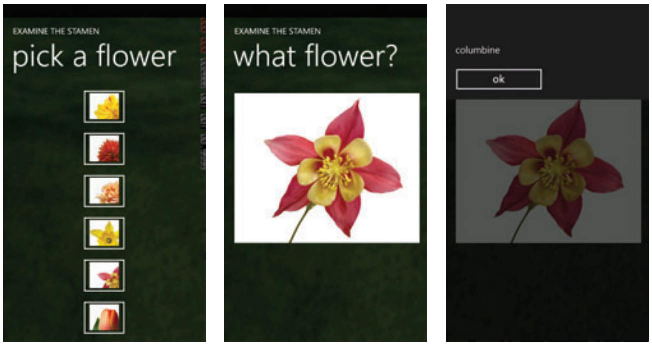
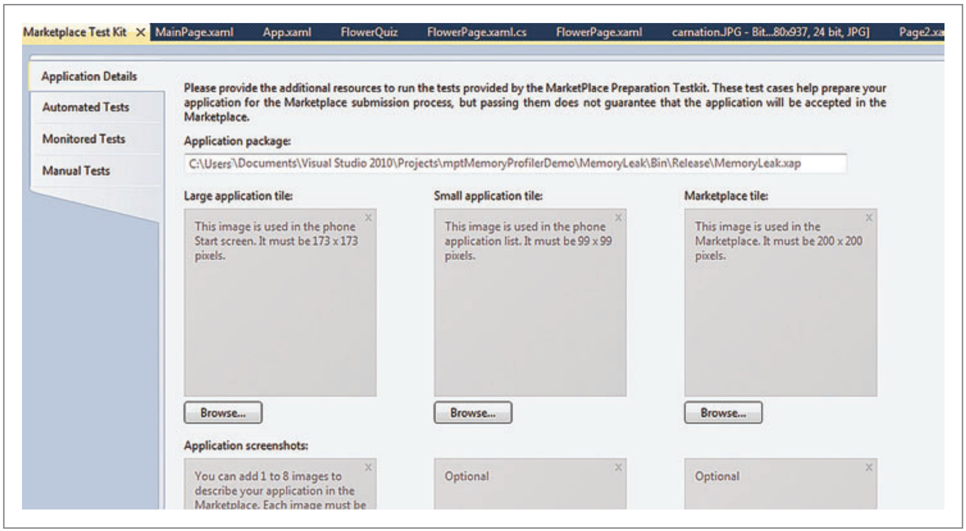


Figure 1 **Images Displayed in the Examine the Stamen Program**

Although this application isn't completely real world, it does represent a reasonable navigation model for a phone application. I'll evaluate this application using the Marketplace Test Kit in Visual Studio and then examine it further with the Windows Phone Performance Analysis tool. Once I identify any problems, I'll use documentation resources to figure out how to fix the problems, and then I'll retest with the tools.

Let's get started.



Please provide the additional resources to run the tests provided by the MarketPlace Preparation Testkit. These test cases help prepare your application for the Marketplace submission process, but passing them does not guarantee that the application will be accepted in the Marketplace.

Application package:
C:\Users\Documents\Visual Studio 2010\Projects\mptMemoryProfilerDemo\MemoryLeak\Bin\Release\MemoryLeak.xap

Large application tile:
This image is used in the phone Start screen. It must be 173 x 173 pixels.

Small application tile:
This image is used in the phone application list. It must be 99 x 99 pixels.

Marketplace tile:
This image is used in the Marketplace. It must be 200 x 200 pixels.

Application screenshots:
You can add 1 to 8 images to describe your application in the Marketplace. Each image must be

Optional

Optional

Figure 2 **The First Page of the Marketplace Test Kit in Visual Studio**

Using the Marketplace Test Kit

I created a reasonably attractive UI for Examine the Stamen, and a reasonable navigation model. I plan to add more flowers in the future, but right now I want to get my application into the marketplace. The next step for me is to use the Marketplace Test Kit to evaluate my application's marketplace readiness with a suite of automated, monitored and manual tests.

To run the tests, I open my application project in Visual Studio and select "Marketplace Test Kit" on the Project menu.

A new tab opens up within Visual Studio, displaying Marketplace Test Kit test suites. **Figure 2** shows the first page of the test kit.

The available test suites are shown in the tabs on the left. The Application Details tab lets you specify the application images for the automated tests. The automated tests evaluate the application XAP size, iconography and screenshots for compliance with certification requirements and determine the capabilities used by the application. The manual tests provide steps you walk through to exercise your application and make sure it complies with additional certification guidelines.

In this article, I'll focus on the monitored tests. For more information about all four test suites, see the "Windows Phone Marketplace Test Kit" MSDN Library page (`wpdev.ms/toHcRb`).

The monitored test suite evaluates applications for adherence to important certification guidelines such as:

- Startup time
- Peak application memory usage
- Back button handling
- Sudden application exits due to unhandled exceptions

Running the Monitored Tests

To run the monitored tests, you need to start a Release build of the application, deploy it to a device (the tests won't work on the emulator) and navigate through it. The options to configure this are on the Standard toolbar in Visual Studio. The goal when running the monitored tests is to navigate through the application as a user would, exercising all the possible navigation paths. While you do this, the test kit monitors the application and collects data about it.

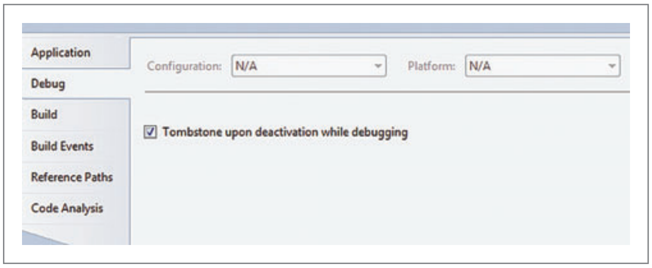
When you test your application with the monitored tests, also test how it performs when you terminate and reactivate it within a short amount of time. This termination and reactivation process is called "tombstoning." In Windows Phone 7.5, your application will go dormant automatically before it's tombstoned.

To force your application to tombstone immediately for debugging and testing purposes, select the "Tombstone upon deactivation while debugging" option on the Debug tab of the Project properties. Open the Project properties by selecting Properties on the Project menu. **Figure 3** shows this option selected. For more information about tombstoning, see the "Execution Model Overview for Windows Phone" MSDN Library page (`wpdev.ms/ExMod`).

> The goal when running the monitored tests is to navigate through the application as a user would, exercising all the possible navigation paths.

After configuring these options, I return to the Marketplace Test Kit tab. I tether my developer-registered device and click Start Application on the Monitored Tests page of the test kit.

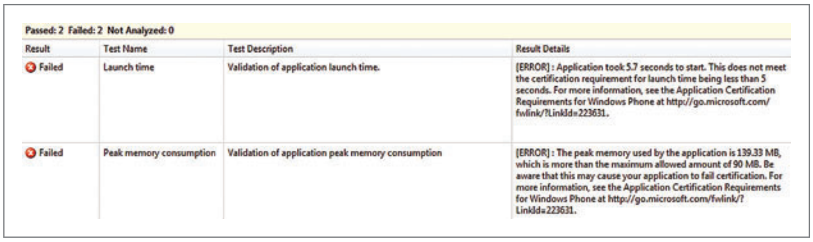
When the application starts I navigate back and forth, selecting flowers, tapping for the name and hitting the Back button to return to the start page of the application. I tap the Start button and then the Back button to force the application to tombstone and return.



Configuration: N/A   Platform: N/A

☑ Tombstone upon deactivation while debugging

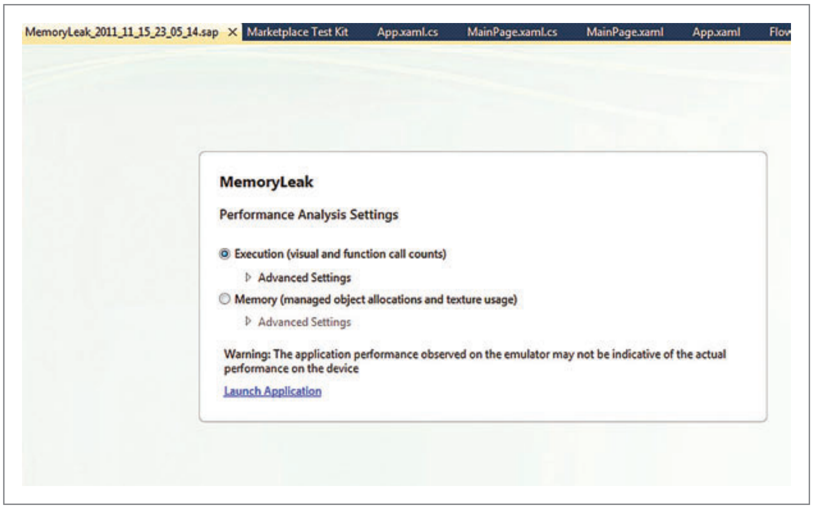
Figure 3 **Selecting the Option to Test Tombstoning in Project Properties**



Passed: 2 Failed: 2 Not Analyzed: 0

Result	Test Name	Test Description	Result Details
Failed	Launch time	Validation of application launch time.	[ERROR] : Application took 5.7 seconds to start. This does not meet the certification requirement for launch time being less than 5 seconds. For more information, see the Application Certification Requirements for Windows Phone at http://go.microsoft.com/fwlink/?LinkId=223631.
Failed	Peak memory consumption	Validation of application peak memory consumption	[ERROR] : The peak memory used by the application is 139.33 MB, which is more than the maximum allowed amount of 90 MB. Be aware that this may cause your application to fail certification. For more information, see the Application Certification Requirements for Windows Phone at http://go.microsoft.com/fwlink/?LinkId=223631.

Figure 4 **Test Results Showing Two Failures**



MemoryLeak

Performance Analysis Settings

● Execution (visual and function call counts)

▷ Advanced Settings

○ Memory (managed object allocations and texture usage)

▷ Advanced Settings

Warning: The application performance observed on the emulator may not be indicative of the actual performance on the device

Launch Application

Figure 5 **The Performance Analysis Tab Before Any Tests Have Been Run**

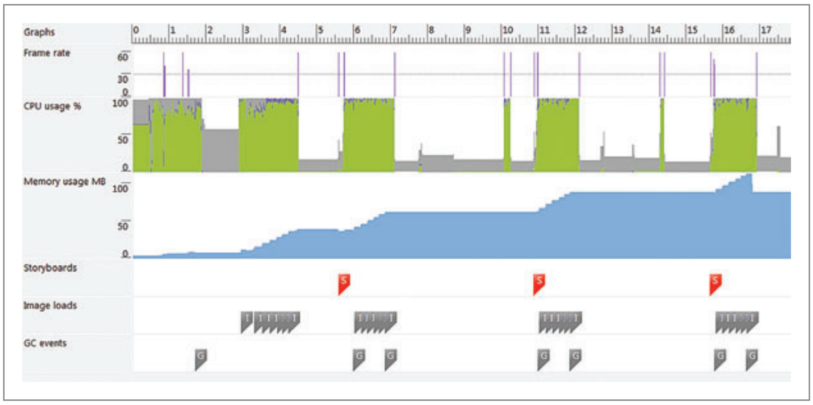


Figure 6 **Results of a Performance Analysis Test**

When I think I've navigated around the way a typical user would, and I've tombstoned and reactivated my application, I can stop the application and test session. For the best results, I exit the application by clicking the Back button from the application Start page to end the test session. I can click the Close Application button on the Monitored Tests page in the test kit, but for the most accurate test results I exit the application by using the Back button. When the application exits, the monitoring sessions ends.

After the test session ends, the test kit results status bar tells me the suite is analyzing results, and when it's finished the results table updates.

The results for my application, shown in **Figure 4**, shock me.

My application has failed two of the four tests in this test suite. The startup time is too slow and it's using too much memory. I decide to dig deeper.

Using the Performance Analysis Tool

In general, for your applications to be popular in the marketplace, they should be performant and responsive. At minimum, you should investigate and fix performance issues identified with the test kit. For either of these scenarios, you can use the Windows Phone Performance Analysis tool, also known as the profiler.

I close the test kit for now and decide to use the profiler to look at my startup time and memory issues. It's a great tool because it will show potential problems with my application and possible courses of action to correct them.

The profiler has two options:

- Execution profiling: The execution profiler will evaluate your application's frame rate, CPU usage and general memory usage. You can use it to drill in to the fill rate and see how many visuals are being created and other execution details of your application that can affect its performance.
- Memory profiling: The memory profiler shows memory usage, image loads and garbage collection events. You can use the memory profiler to look for memory usage trends, which can indicate a memory leak.

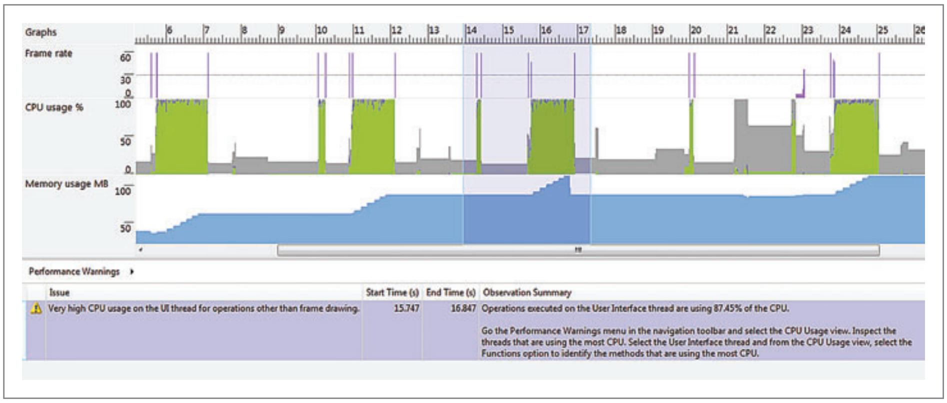
Choose the execution profiler unless you know the only issue with your application is a memory problem. I know I have a memory problem, but I'm curious about the startup time issue and I decide to look at my application with the execution profiler first.

With my application project open in Visual Studio, I go to the Debug menu and choose Start Windows Phone Performance Analysis. (Note: If you're using Visual Studio Premium or Ultimate, do not choose Start Performance Analysis, which doesn't apply to phone projects.)

When you open the Performance Analysis tool, a new tab opens in Visual Studio with the name of the current profiling session. The name includes the name of the project and the date/time of the profiling session, followed by the .sap suffix used for profiling results files. These files are always saved in the project, so you can view them multiple times. **Figure 5** shows the Performance Analysis Tab before any tests have been run.

On the profiler tab, I choose the Execution option. For best results I ensure Windows Phone Device and Release are still selected in the deployment and debug options boxes on the Visual Studio toolbar and make sure my device is tethered and unlocked. (Note: You can deploy an application to the emulator when using the profiler, but results may not be indicative of performance on a device.)

I click Launch Application to start the profiling session. Similar to the Marketplace Test Kit, I use my application the way a user would, and make sure that I tombstone and return to the application



	Issue	Start Time (s)	End Time (s)	Observation Summary
⚠	Very high CPU usage on the UI thread for operations other than frame drawing.	15.747	16.847	Operations executed on the User Interface thread are using 87.45% of the CPU. Go the Performance Warnings menu in the navigation toolbar and select the CPU Usage view. Inspect the threads that are using the most CPU. Select the User Interface thread and from the CPU Usage view, select the Functions option to identify the methods that are using the most CPU.

Figure 7 **A Performance Warning About High CPU Usage on the UI Thread**

at least once. I exit the application using the Back button, which is the preferred method for the most accurate results, although you can also end the profiling session using the Stop Profiling option in the Performance Analysis tool. The profiler spends some time analyzing the results and displays them on the page in graph format (shown in **Figure 6**). My results are very interesting.

The green portion of the graphed CPU usage indicates screen updates and touch input. I can see high CPU usage initially, which isn't surprising given the slow startup time. I also see huge spikes in CPU usage aligned with images being loaded, and that my memory usage creeps higher and higher. Viewing these results without further examination tells me that my memory use problem is probably related to how I'm handling images in my application. Although any memory used by my application is released when my application exits, based on this graph, I'm concerned that my application could crash a device if it's left running for more than the few seconds I spend testing it.

Now I rerun the performance analysis tool with the memory option selected, and this confirms my growing memory-use problem.

Finding and Fixing the Problem

To track down problems using the profiling tool, select problem areas in the graph and review the instructions in the Observation Summary section.

In the execution profiler results, I click and drag with the mouse to select a portion of the graph that shows a CPU usage spike. The Performance Warning section immediately updates with an issue to investigate (see **Figure 7**).

According to the Observation Summary, the application uses a lot of CPU to execute functions on the UI thread. This would certainly lead to slow startup time and poor performance overall, but I'm not sure it would contribute to the memory problem. The profiler is great in that it gives me some instructions to follow, so I do this. I select CPU Usage and then Functions. The results table updates and I sort the results by the Inclusive Samples (%) column. My application function calls are displayed in blue with fully qualified names that include the namespace (suspiciously, MemoryLeak in this case), class and method name. Also, the function calls are live links to the methods in my code. **Figure 8** shows these results.

I can tell by looking at these results that the methods executing when I load the second page are using a lot of CPU. This probably won't fix my start-time issue, but it definitely could be contributing to the memory issue.

I click the link to view the FlowerPage.OnNavigatedTo method. This method creates a list of Flower objects and loads a bitmap for each Flower using the LoadBitmap method. Following is a typical call I make to the LoadBitmap method:

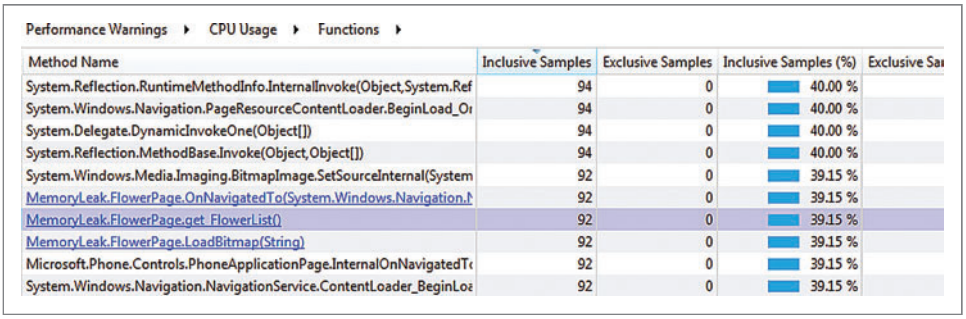
```
bitmap = LoadBitmap("/MemoryLeak;component/Images/tulip.jpg");
```

And the LoadBitmap method, which loads the resource:

```
private BitmapImage LoadBitmap(string urlString)
{
  var streaminfo = App.GetResourceStream(new Uri(urlString, UriKind.Relative));
  BitmapImage bitmap = new BitmapImage();
  bitmap.SetSource(streaminfo.Stream);
  return bitmap;
}
```

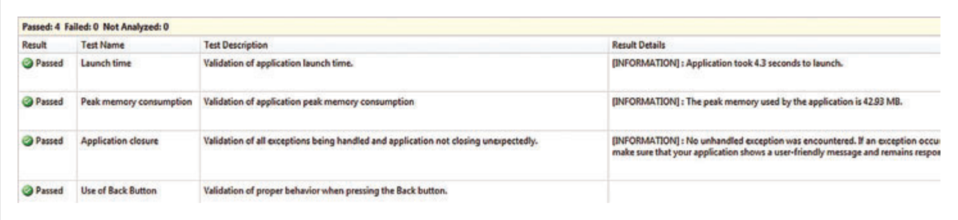
When a user navigates to the page, I extract the name of the flower that was clicked on the main page from the navigation URI and load the same flower image on the FlowerPage.

It's clear that loading the images is causing a memory problem, but it's not clear to me what I should do next.



Method Name	Inclusive Samples	Exclusive Samples	Inclusive Samples (%)	Exclusive Sa
System.Reflection.RuntimeMethodInfo.InternalInvoke(Object,System.Ref	94	0	40.00 %	
System.Windows.Navigation.PageResourceContentLoader.BeginLoad_Oı	94	0	40.00 %	
System.Delegate.DynamicInvokeOne(Object[])	94	0	40.00 %	
System.Reflection.MethodBase.Invoke(Object,Object[])	94	0	40.00 %	
System.Windows.Media.Imaging.BitmapImage.SetSourceInternal(System	92	0	39.15 %	
MemoryLeak.FlowerPage.OnNavigatedTo(System.Windows.Navigation.N	92	0	39.15 %	
MemoryLeak.FlowerPage.get_FlowerList()	92	0	39.15 %	
MemoryLeak.FlowerPage.LoadBitmap(String)	92	0	39.15 %	
Microsoft.Phone.Controls.PhoneApplicationPage.InternalOnNavigatedT	92	0	39.15 %	
System.Windows.Navigation.NavigationService.ContentLoader_BeginLoa	92	0	39.15 %	

Figure 8 **The Peformance Analysis Tool Shows Methods That Might Be Causing Problems**


Passed: 4 Failed: 0 Not Analyzed: 0

Result	Test Name	Test Description	Result Details
✓ Passed	Launch time	Validation of application launch time.	[INFORMATION] : Application took 4.3 seconds to launch.
✓ Passed	Peak memory consumption	Validation of application peak memory consumption	[INFORMATION] : The peak memory used by the application is 42.93 MB.
✓ Passed	Application closure	Validation of all exceptions being handled and application not closing unexpectedly.	[INFORMATION] : No unhandled exception was encountered. If an exception occu make sure that your application shows a user-friendly message and remains respo
✓ Passed	Use of Back Button	Validation of proper behavior when pressing the Back button.	

Figure 9 **Changing Image Handling Results in Passing All Four Marketplace Tests**

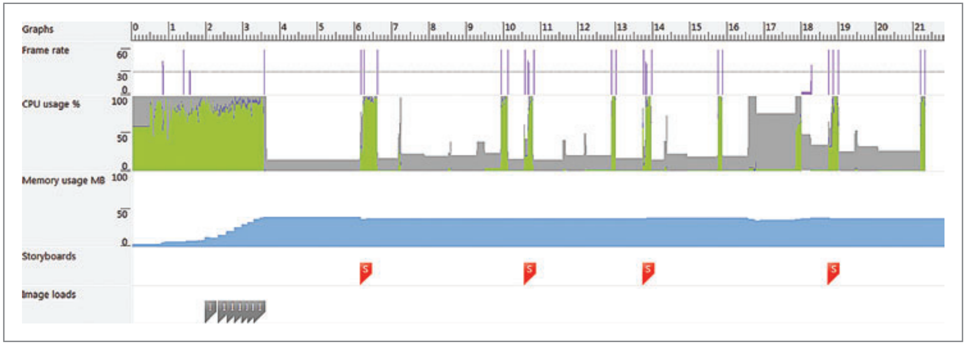


Figure 10 **Image Handling Changes Result in CPU and Memory Performance Analysis Improvements**

If the Observation Summary doesn't give you enough information to solve the performance issues in your application, you should check MSDN and the Web for performance guidance. Following are some great resources:

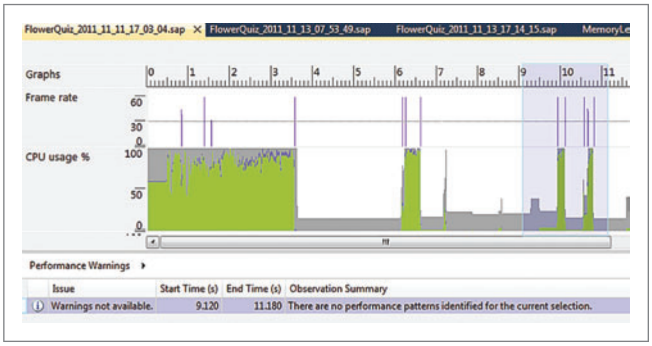
- "Performance Considerations in Applications for Windows Phone" (under the "Media" section) (wpdev.ms/utCq6h)
- "Performance Techniques for Windows Phone" (wpdev.ms/perfTech)
- Silverlight for Windows Phone Performance team blog (wpdev.ms/slmperf)
- Analyzing and Improving Windows Phone Application Performance (video from MIX11) (wpdev.ms/mixwpperf)
- Expert Lessons: Top Tips for Building a Successful Windows Phone Application (video from MIX11) (wpdev.ms/mixwptoptips)

I start researching performance and loading resources on phone applications and discover something important. According to the Media section of the "Performance Considerations in Applications for Windows Phone" MSDN Library article, I should be specifying my image files as content rather than resources, because the phone is optimized to use files. When a media file is compiled as a resource, the content is copied to a file before being used, which decreases performance.

I change the build action of my image files to Content and make a small change to my code to accommodate this.

In my LoadBitmap method, I specify the UriSource of the BitmapImage rather than calling SetSource:

```
private BitmapImage LoadBitmap(string urlString)
{
  BitmapImage bitmap = new BitmapImage();
  bitmap.UriSource = new Uri(urlString, UriKind.Relative);
  return bitmap;
}
```


FlowerQuiz_2011_11_11_17_03_04.sap × FlowerQuiz_2011_11_13_07_53_49.sap FlowerQuiz_2011_11_13_17_14_15.sap MemoryLe

Performance Warnings

	Issue	Start Time (s)	End Time (s)	Observation Summary
ⓘ	Warnings not available.	9.120	11.180	There are no performance patterns identified for the current selection.

Figure 11 **Investigating a CPU Spike Shows No Performance Warnings**

And when I make the call to LoadBitmap, I pass the relative URL to each bitmap:

```
bitmap = LoadBitmap("/Images/tulip.jpg");
```

Rerunning the Test Kit and Performance Analysis Tools

Once you think you've fixed the issues raised in the Marketplace Test Kit and the Performance Analysis tool, you can run these tools again.

I recompile my application and run the Marketplace Test Kit again, and I can't believe the difference in the results (see **Figure 9**). The app now passes all four tests. The startup time isn't great, but at least it's meeting the bar.

Finally, I run the execution profiler a final time. I see a big difference in results (see **Figure 10**).

Now instead of big CPU spikes and images being loaded over and over as the user navigates between pages, the images are loaded once, when the application starts. This graph also tells me that my application's memory use is remaining constant and relatively low compared to my previous version of the application. Next, I select some of the smaller CPU spikes and see the results shown in **Figure 11**.

I'm relieved to see that the profiler doesn't see any performance problems and I make plans to submit my application to the marketplace. I have confidence it will be accepted, and I can continue to improve my application and submit updates if I want.

Follow This Pattern

In this article I've described how to identify and fix issues in a sample Windows Phone application by using the Marketplace Test Kit and the Performance Analysis tool. These tools are integrated into Visual Studio and install as part of the Windows Phone SDK. The Marketplace Test Kit helps you determine if your application will meet certification requirements. The Performance Analysis tool will help you identify memory and CPU performance issues. Before you submit your applications to the marketplace, I recommend a pattern similar to what I've shown in this article:

1. Use the tools I've shown, including all the test suites in the Marketplace Test Kit.
2. Identify and fix any issues.
3. Retest to verify fixes.

If you follow this pattern, you'll find problems earlier and create better applications faster. Also, this will help ensure your applications are accepted into the marketplace on the first attempt. ■

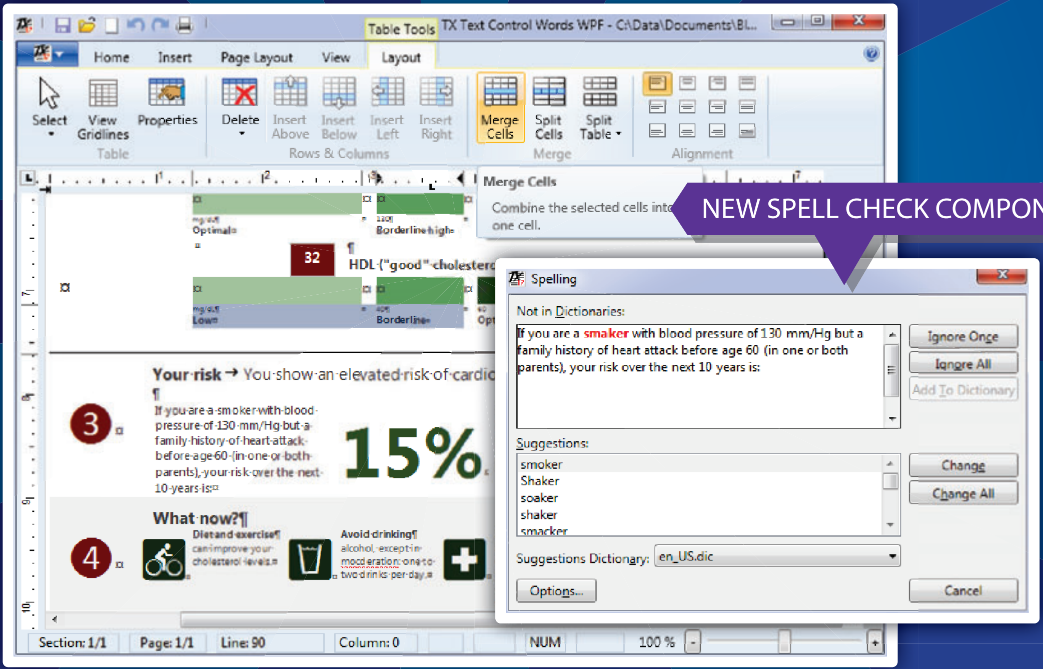
CHERYL SIMMONS *is a senior programming writer on the Windows Phone Developer Content team at Microsoft.*

THANKS to the following technical experts for reviewing this article: *Pratap Lakshman, Raghuram Lanka and Nitin Madnikar*

WINDOWS FORMS | WPF | ASP.NET | ACTIVEX

WORD PROCESSING COMPONENTS

VERSION 17.0 RELEASED



Your risk → You show an elevated risk of cardi

If you are a smoker with blood pressure of 130 mm/Hg but a family history of heart attack before age 60 (in one or both parents), your risk over the next 10 years is:

15%

What now?

Diet and exercise can improve your cholesterol levels

Avoid drinking alcohol, except in moderation: one to two drinks per day.

Not in Dictionaries:

If you are a smaker with blood pressure of 130 mm/Hg but a family history of heart attack before age 60 (in one or both parents), your risk over the next 10 years is:

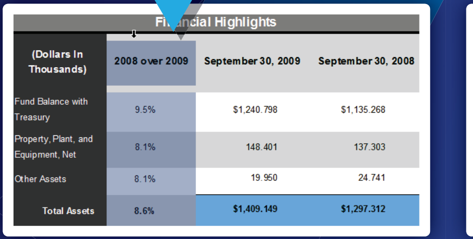
Suggestions:

- smoker
- Shaker
- soaker
- shaker
- smacker

Suggestions Dictionary: en_US.dic

NEW SPELL CHECK COMPONENT

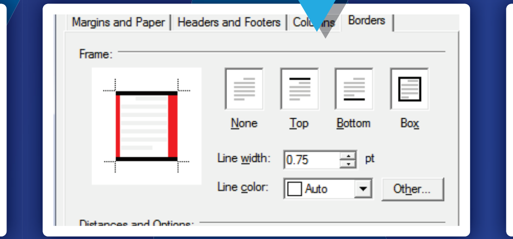
CELL MERGING, COL SELECTION



Financial Highlights

(Dollars In Thousands)	2008 over 2009	September 30, 2009	September 30, 2008
Fund Balance with Treasury	9.5%	$1,240.798	$1,135.268
Property, Plant, and Equipment, Net	8.1%	148.401	137.303
Other Assets	8.1%	19.950	24.741
Total Assets	8.6%	$1,409.149	$1,297.312

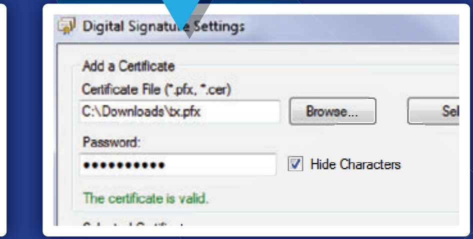
PAGE BORDERS



Line width: 0.75 pt

Line color: Auto

DIGITAL SIGNATURES IN PDF



Add a Certificate

Certificate File (*.pfx, *.cer): C:\Downloads\tx.pfx

Password: ••••••••••

☑ Hide Characters

The certificate is valid.

TX TEXTCONTROL®
word processing components

Microsoft® Visual Studio®
PARTNER

Word Processing Components
for Windows Forms, WPF & ASP.NET

WWW.TEXTCONTROL.COM

US +1 877 - 462 - 4772 (toll-free)
EU +49 421 - 4270671 - 0

What's New in Windows Workflow Foundation 4.5

Leon Welicki

At the BUILD conference last September (buildwindows.com), Microsoft unveiled the next version of Windows Workflow Foundation (WF 4.5) and made available a public preview as part of the Windows 8 Developer Preview (msdn.microsoft.com/windows/apps/br229516). In this article, I'll walk through the key new features added in WF 4.5. Given the general scope of the article and the size of the feature set, each discussion will be brief, but I hope you'll get excited enough to download the Windows 8 Developer Preview and start playing with WF 4.5.

WF: Yesterday, Today and Tomorrow

WF 4 shipped in 2010 to a great reception from the developer community. WF 4 included significant advances over its predecessor (WF 3.5): improved runtime, better overall performance, simplified activity authoring, full Windows Communication Foundation (WCF) integration, declarative authoring and dramatically simplified designer rehosting. 2011 brought a fully featured version of StateMachine in the Microsoft .NET Framework 4 Platform Update 1, available today with the same quality and support guarantees as any other component of the framework. With WF 4.5, our goal was to solve the major issues we learned about from our customers. We've also been working to take all the power of WF to the cloud; there's a great BUILD presentation you can watch to see what the team has been up to (bit.ly/rbJROw).

Windows Workflow Foundation 4.5 is currently a community technical preview. All information is subject to change.

This article discusses:

- Authoring improvements in WF 4.5
- New versioning support in WF 4.5
- Runtime enhancements in WF 4.5

Technologies discussed:

Windows Workflow Foundation, Windows 8 Developer Preview, Windows Communication Foundation

Code download available at:

code.msdn.microsoft.com/mag201202WF45

As you read through this article, I'm sure you'll have questions about compatibility between WF 4 and WF 4.5. Be assured that the versions are built on the same code base and are fully compatible. All your WF 4 investments will be preserved in WF 4.5 without needing any changes.

After we shipped WF 4, we received a lot of great feedback we used to plan the next release. In WF 4.5, we wanted to address your top concerns and add features to provide the best framework for building workflow applications. **Figure 1** shows customer requests and the features created in response to those requests.

Notice that the table is organized by themes; I'll use the same themes in the upcoming sections to present the most important features.

Authoring Improvements

C# Expressions WF 4 lets you write expressions using Visual Basic and, when used in Visual Studio, the expression editor provides all the language services you might expect, such as auto-complete, IntelliSense and so forth. But our WF 4 customers told us they'd rather write C# than Visual Basic, so we added support for C# expressions. Now, if you create a C# project you get C# expressions in your workflow. Don't worry, though. You still get Visual Basic expressions if you create a Visual Basic project. C# support also comes with all the language services you'd expect, as shown in **Figure 2**. Notice the IntelliSense box showing second C# overload for string.Format.

If you want to try C# expressions, just create a C# Workflow project. You can also open the CSharpExpressions project in the companion code for this article.

Figure 1 **Customer Requests and New Features for WF 4.5**

Theme	Request	Feature
Authoring Improvements	Expressions in the language of the project	C# Expressions
	Create workflow services based on an existing contract	Contract-First
	Add comments to activities in the designer surface	Annotations
	Use the WF designer more effectively, especially navigating large workflows	Auto-connect, Auto-insert, Pan, Tree view, Auto-surround with sequence
	Search integration in the workflow designer	Search
	Break the build when there's an error in a workflow in the designer	Build integration
	Create StateMachine workflows	StateMachine
Versioning	Basic building blocks for a versioning story	WorkflowIdentity
	Host several versions of a service side by side	WFSH versioning support
	Update running instances of a workflow to a new definition	Dynamic Update
Runtime Enhancements	Run my workflows in Partial Trust	Partial trust support
	Be able to plug my own expressions story	Expressions extensibility
	Better runtime performance	Visual Basic expressions performance enhancements

Contract-First Service Authoring WF 4 comes with great WCF integration. You can create WCF services leveraging all of the WF capabilities (long-running execution, durable state, declarative authoring and visibility into execution) without writing a single line of code (see "Visual Design of Workflows with WCF and WF 4" at msdn.microsoft.com/magazine/ff646977). When you author a workflow service in WF 4, service contracts are inferred from the workflow definition. When the host starts, it walks through the workflow definition looking for messaging activities, then exposes the corresponding contract (for example, each Receive activity translates to a service operation).

Many customers said they'd prefer to create their workflow services from an existing WCF service contract (the workflow is just one of the possible implementations of that contract), so in WF 4.5 we added support for "contract-first" service authoring. Now you can import a WCF service contract definition in a workflow project and make one or more workflow services implement that contract. This new approach to workflow service authoring doesn't replace the workflow-first approach.

Contract-first in WF is a two-step process: You add a reference to the contract (a regular WCF service contract) in your project and then implement the contract in your services.

To import the contract, right-click on the project, select the "Import Service Contract" menu item and choose the contract type. Once the contract is imported and the project is built, a set of activities representing each operation is added to the toolbox, as shown in **Figure 3**.

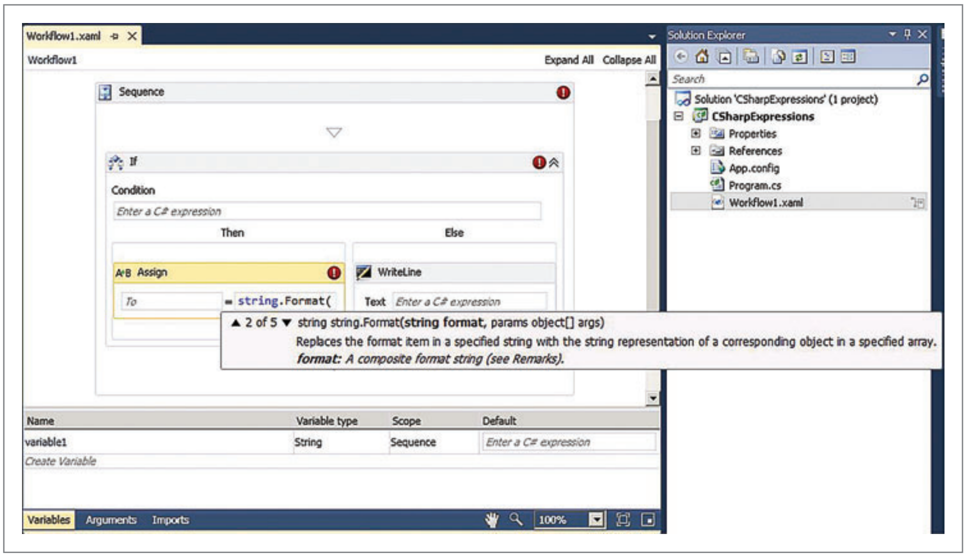
As **Figure 4** shows, you can add activities between the Receive and SendReply. These activities represent the body of the operation.

The second step is to indicate that a given service implements the contract. To do this, you open the workflow service in the designer and add the contract to the ImplementedContracts collection of the WorkflowService root.

Once the service implements the contract, the designer will run a new set of validation rules to ensure the service honors the contract. As a result, you'll see warnings and errors if the contract isn't fully implemented.

If you want to see contract-first in action, take a look to the ContractFirst project in the companion code. For a hands-on experience follow the steps described in the previous section.

Use the WF designer more effectively The WF visual designer provides a graphic representation of your workflows, making them easier to create and share. The designer is a key feature of WF and is the most used authoring tool. In WF 4.5, we added highly requested features to enable you to be more productive using it.

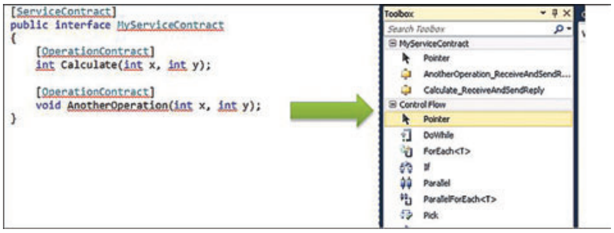


= string.Format(

▲ 2 of 5 ▼ string string.Format(**string format**, params object[] args)
Replaces the format item in a specified string with the string representation of a corresponding object in a specified array.
format: *A composite format string (see Remarks).*

Name	Variable type	Scope	Default
variable1	String	Sequence	*Enter a C# expression*
Create Variable			

Figure 2 **C# Expressions in a Workflow**



```
[ServiceContract]
public interface MyServiceContract
{
    [OperationContract]
    int Calculate(int x, int y);

    [OperationContract]
    void AnotherOperation(int x, int y);
}
```

Figure 3 **The Generated Activities Are Added to the Toolbox**

Annotations In WF 4, the only place you can add text in the designer is the DisplayName section of an activity, which is a one-line text area at the top of the activity. There's no other way to add non-structured descriptive information to your workflow. However, you wanted to be able to add comments or text in your activity designer to convey more than just the workflow definition (such as adding a description of the behavior of a process step).

In WF 4.5 we added *annotations*, which enable you to add textual comments and descriptions to activities in the designer. **Figure 5** shows a workflow with and without annotations. Notice how you can now add descriptive information to any activity. With annotations, a printout of the designer can fully convey what the workflow is doing.

You can display annotations as always visible or as a sticky note in the DisplayName bar of an activity.

Annotations are an opt-in feature and have no execution-time impact. They're added to the workflow XAML file as attached properties, as shown in the following code snippet:

```
<Activity mc:Ignorable="sap2010"
  x:Class="DesignerImprovements.Annotations"
  xmlns="http://schemas.microsoft.com/netfx/2009/xaml/activities"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  xmlns:sap2010="http://schemas.microsoft.com/netfx/2010/xaml/
activities/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <Sequence sap2010:Annotation.AnnotationText="This is an example annotation">
    <WriteLine Text="Hello Dev11"/>
  </Sequence>
</Activity>
```

To see annotations in action, take a look at the attached Annotations project in the companion code. For a hands-on experience, open a WF 4.5 workflow project, add an activity to the designer and select "Add annotation" in the activity's context menu.

Enhancing Flowchart and StateMachine Designers You asked us to improve aspects of the design experience to make you more productive, particularly with regard to adding new activities to a Flowchart or StateMachine (available through .NET Framework Platform Update 1). We listened.

We added *auto-connect*, which enables automatically connecting a new activity to an existing activity in the designer. And we added *auto-insert*, which lets you insert an activity between two existing activities by dropping the new activity on the connecting line. **Figure 6** shows these features.

To try this, just create a WF project and add a Flowchart or StateMachine. When adding a new activity, you'll notice dropping targets for auto-connect. After you've connected two activities, you can add a third by dropping it on the connecting line.

Large-Workflow Navigation In WF 4, when a workflow is larger than the screen, you move around using the vertical and horizontal scrollbars. You told us that moving around a large workflow this way is painful and limits your productivity.

In WF 4.5 we added panning, which allows moving through the designer canvas using the mouse. You can enable Pan mode by clicking the Pan button in the designer. Panning is available as a general designer feature, so you can use it with any existing activity.

To try panning, open any workflow in the designer, click the Pan button and enjoy panning around!

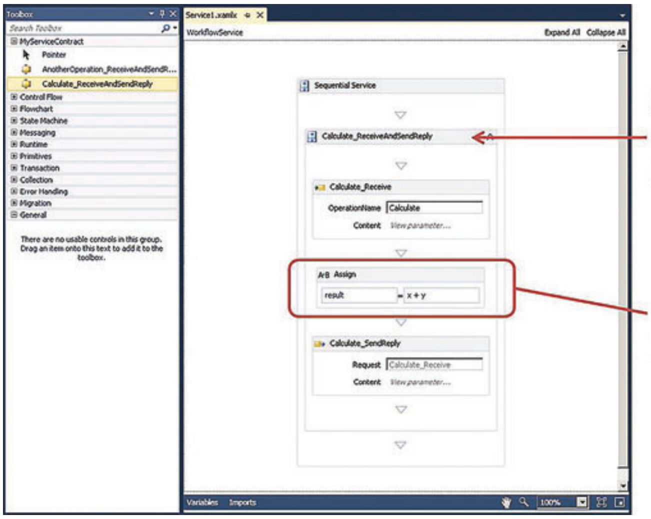
WF 4.5 also provides a tree view that allows you to visualize the document outline of your workflow. This gives you a very succinct and structured view of your workflow, enabling you to navigate large workflows more easily. When you click an activity in this view, it's selected in the workflow definition, as shown in **Figure 7**.

To try this feature, open a workflow in the designer, go to the View menu, select Other Windows and choose Document Outline.

Search Integration in the Designer In WF 4 you can use the Visual Studio search when using the WF designer, but search results aren't integrated with the WF designer. Clicking a search result doesn't take you to a particular location in a workflow. You made it clear how this affects your productivity and how important it was for us to provide this capability.

In WF 4.5 we integrated the WF designer with Visual Studio search. Now clicking on a search result takes you to an activity in a workflow, even if it's nested several levels away.

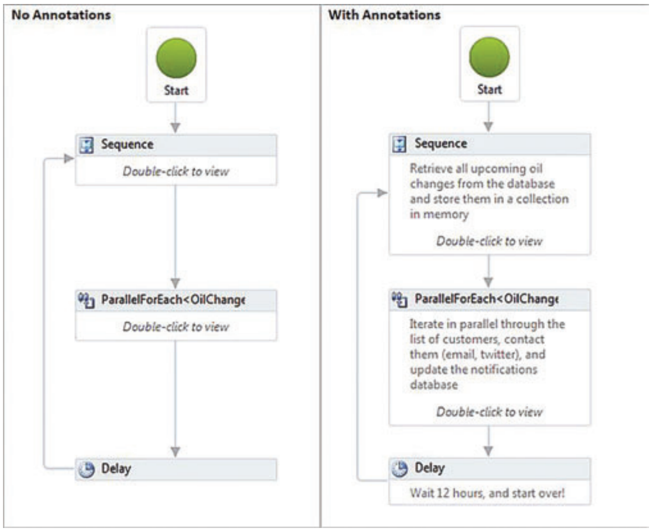
To try this out, create a workflow project and start editing a workflow in the designer. Add some activities



Calculate_ReceiveAndSendReply is the template created for the Calculate operation in the contract (it is ReceiveAndSendReply activity configured to honor the contract).

Represents the operation implementation. In this case it is a simple computation but it can be any activity (including Flowchart or StateMachine).

Figure 4 **A Simple Service Method That Computes a Value**



Retrieve all upcoming oil changes from the database and store them in a collection in memory

Iterate in parallel through the list of customers, contact them (email, twitter), and update the notifications database

Wait 12 hours, and start over!

Figure 5 **A Workflow with and Without Annotations**

and configure their properties. Now, search for a keyword using Visual Studio search. The search results should take you to the activity that matches the keyword.

Auto-Surround with Sequence The WF 4 activity model supports composition at its very core; activities can be composed without restrictions. For example, the Body property of a While activity can be a Sequence, Parallel, Flowchart, StateMachine, WriteLine, Delay or any other existing activity. Often, composite activities have a Body property that accepts just one activity (instead of a set), which can be any other leaf or composite activity.

One of the more common activities for Body is Sequence, especially with control flow activities like While, ForEach and so forth. With these, if you add a single child and then change your mind in favor of a sequence of elements, you need to cut the child in the Body, then add a Sequence to the Body and paste the child. This is tedious and error prone.

In WF 4.5, we added an "auto-surround with Sequence" feature. For example, if you have a While and its Body is a WriteLine, when you drop another activity, a Sequence containing both the existing activity and the new one is automatically created, as illustrated in **Figure 8**.

To try this, follow the steps in **Figure 8**.

Build Integration In WF 4 you can create a new activity declaratively in the designer. When you do so, you're actually defining a new type that will be placed in the assembly generated when the project is compiled. However, if you build a WF 4 project and there are errors in the XAML, the build still succeeds though your workflow is not included in the resulting assembly.

In WF 4.5 we fixed this problem. If you build a workflow that has an error, the build will break, as you'd expect.

To implement this solution, we added extensibility to XamlBuildTask so it can be accessed during compilation of XAML activities, breaking the build when a XAML workflow file has errors.

To try this feature, create a new WF application and add an incorrectly configured activity to a workflow. You'll get a build error. You can also try the ErrorBreaksTheBuild project in the companion code.

State Machine out of the Box StateMachine is a very important control flow activity that didn't make it to WF 4. That brought tons of feedback about the importance of this activity, so now StateMachine is available in WF 4 after you install the .NET Framework Product Update 1. In WF 4.5, you won't need to install any update to take advantage of StateMachine—it's included right out of the box.

To try this feature, create a workflow project and add a StateMachine activity or just open the StateMachine sample project in the companion code for this article.

Versioning Support

WF 4 doesn't include support for workflow versioning. If you wanted versioning, you had to write everything on your own, often hitting issues that are hard to work around. WF 4.5 includes new features that enable versioning.

WorkflowIdentity In WF 4, the host is responsible for associations between definitions and instances. Once an instance has been persisted and unloaded from memory, the host must provide the right definition to continue the execution of the instance. One big challenge is that there is no information in the persisted instance state that helps the host determine what definition has been used to create the instance. Furthermore, if the host is configured with the wrong definition when loading an instance, the user will get an awkward exception because the error is a side effect of not being able to match the instance state with the definition, rather than a real version mismatch.

WF 4.5 introduces WorkflowIdentity, a new type that refers to a fully configured workflow definition and is part of the instance runtime state. A WorkflowIdentity contains a name (string), a version (System.Version), and a package (string). Name and version are self-explanatory; package is an optional string used for disambiguation. It refers to the container of the workflow definition (assembly name, service URI, or any string of your choice). WorkflowIdentity is the cornerstone for all WF versioning features.

One of the most useful traits of WorkflowIdentity is that it's a part of the workflow instance state and lives through the entire lifecycle of an activity: It's saved during persistence, can be queried from the instance store and is emitted with the tracking information for a workflow instance.

Using WorkflowIdentity is easy. The following snippet shows how to use it with WorkflowApplication, our single-instance,

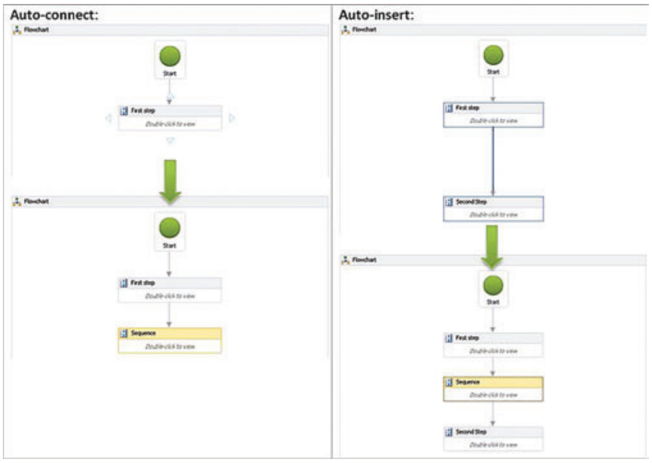


Figure 6 **Auto-Connect and Auto-Insert**

single-definition, in-process host (you just have to pass an instance of WorkflowIdentity to the constructor of WorkflowApplication):

```
WorkflowIdentity identity = new WorkflowIdentity("Sample", new Version(1, 0, 0, 0),
  "MyPackage");
WorkflowApplication application = new WorkflowApplication(new
MyActivity(), identityV1);
```

We just configured the WorkflowApplication with a WorkflowIdentity, but we haven't yet done anything useful. The next code sample shows how to use the identity to detect a version mismatch and to provide an actionable error message. In the event of a version mismatch when trying to load an instance, you'll get a VersionMismatchException that states the cause of the problem and contains the supplied and expected identities. This information can be used for logging or recovering from the error:

```
try
{
  WorkflowIdentity wrongIdentity = new WorkflowIdentity("Sample", new
Version(2, 0, 0, 0),
    "MyPackage");

  WorkflowApplication application = new WorkflowApplication(new WrongActivity(),
    identityV2);

  application.Load(instanceId);
}
catch (VersionMismatchException ex)
{
  Console.WriteLine("Version Mismatch! {0}", ex.Message);
  Console.WriteLine("Expected: {0}; Provided: {1}", ex.ExpectedVersion,
ex.ActualVersion);

}
```

Finally, you can learn the identity of a given workflow instance before loading it from the instance store. To query the identity, you need to get a WorkflowApplicationInstance, a new type introduced in WF 4.5 that represents an instance that has not been associated with a definition. It's used to retrieve metadata about the instance—in this case, the identity. See bit.ly/ssAYDn for more information.

Note that WorkflowIdentity works not only with WorkflowApplication, but also with WorkflowServiceHost.

If you want to try this feature, open the WorkflowIdentity project in the companion code.

WorkflowServiceHost WorkflowServiceHost (WFSH) is the out-of-box, single-definition, multiple-instances host for workflows provided in WF 4. However, an unfortunate WF 4 limitation is that changes to the workflow definition result in an exception if you try to load a previously persisted instance into the WFSH. This is because the host is unable to run these instances using the new definition (this is the problem we noted in the WorkflowIdentity section). Some customers worked around the lack of built-in versioning support in WF 4 using multiple WFSHs and a WCF routing service as an intermediary between the client app and the workflows. Clients send messages to the router, which routes the messages to the corresponding WFSH configured with the right version of the definition. The downside is that this requires the client application to be version-aware to successfully send messages to a service instance.

In WF 4.5, WFSH has become a multiversion host; you can deploy multiple versions of a workflow service within the same WFSH and it will deliver incoming messages to the correct version.

The semantics are very simple: New instances start with the latest version of the service, and running instances continue executing with the version used to start them. There are some restrictions on what you can change from one version to the next; you can't remove service operations (Receives), but you can add new ones (the rules are similar to creating a derived interface in the CLR).

The key enabler for this feature is WorkflowIdentity. To determine the version of a service definition you need to configure its identity. Old versions of the service must be placed in a "supported versions" folder, which is a folder in App_Code with the same name as the workflow service (see **Figure 9**). Alternatively, old versions can also be loaded explicitly into the WorkflowServiceHost by adding to the SupportedVersions collection prior to opening the host.

With this new feature, the router is no longer needed, and client apps don't have to be version-aware. They just send a message; the usual correlation mechanism resolves it to the right instance and the host uses the corresponding definition (because the persisted instance state contains the identity of the definition needed to load it). XCopy deployment semantics are preserved, so you don't need to write a single line of code to use it. This feature is also available in self-hosting scenarios (hosting WFSH in your own app).

To try this feature, open the WFSH_SxS project in the companion code.

Dynamic Update In WF 4, once a workflow instance has begun, there's no supported way to change the definition of the workflow. This is often a problem when programs need to be updated due to bug fixes or changing requirements.

Our enterprise customers were emphatic about the importance of this capability, because they often

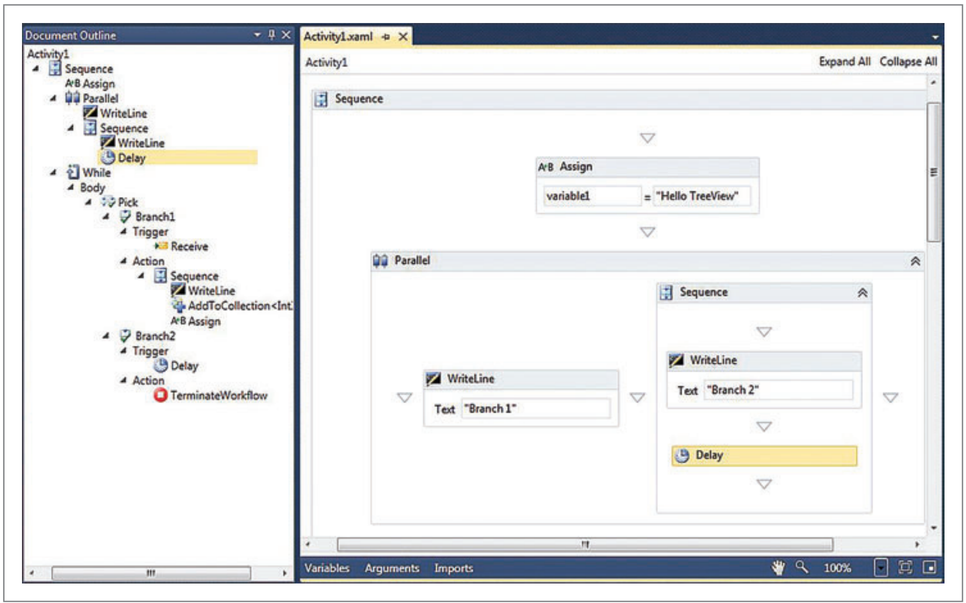
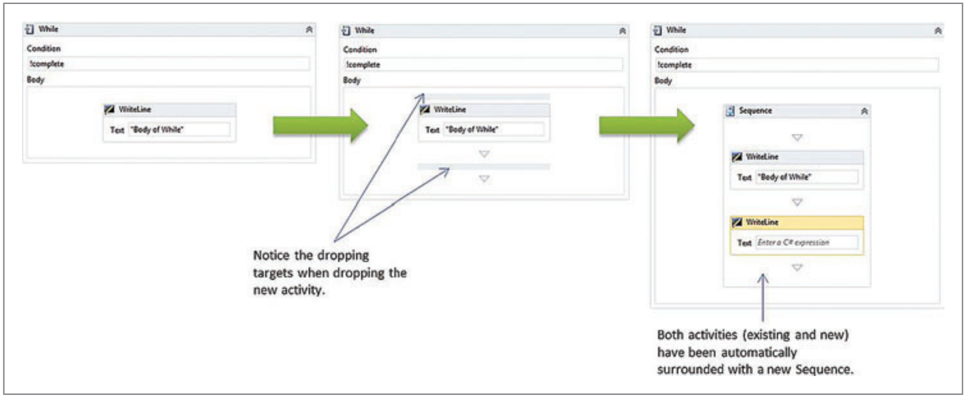


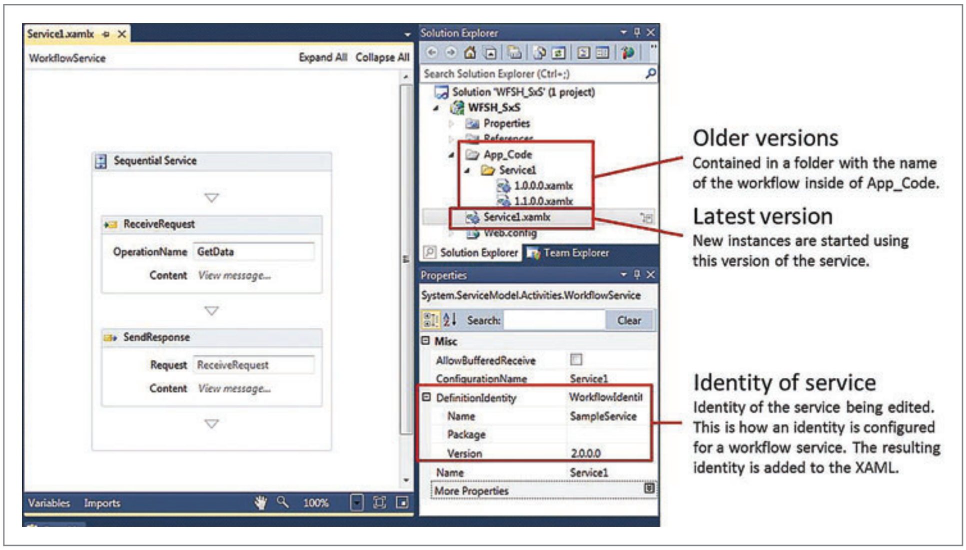
Figure 7 **Document Outline View of a Workflow**



Notice the dropping targets when dropping the new activity.

Both activities (existing and new) have been automatically surrounded with a new Sequence.

Figure 8 **Auto-Surround with Sequence**



OperationName GetData
Content View message...
Request ReceiveRequest
Content View message...

AllowBufferedReceive ☐
ConfigurationName Service1
DefinitionIdentity WorkflowIdentit
Name SampleService
Package
Version 2.0.0.0
Name Service1

Older versions
Contained in a folder with the name of the workflow inside of App_Code.

Latest version
New instances are started using this version of the service.

Identity of service
Identity of the service being edited. This is how an identity is configured for a workflow service. The resulting identity is added to the XAML.

Figure 9 **WorkflowServiceHost Side-by-Side Versioning**

need to change a particular workflow instance in long-running workflows. For example, suppose a workflow that models an interview process has four interviewers but, given a new company policy, now needs to be changed to add a fifth interviewer. You can't do that with WF 4.

You can in WF 4.5. Dynamic Update allows you to make changes to a running instance of a workflow to accommodate a new workflow definition. Such changes might be motivated by omissions at design time, bugs in the workflow, or new requirements. Dynamic Update isn't intended for situations that require wholesale changes, resulting in a workflow that differs significantly from its original design. In that sort of case, you should design a new workflow instead of making changes to a running instance.

Dynamic Update is generally a two-step process: When changing the workflow definition, you also need to create an update map: a structure that contains information about the changes. When the new version is deployed, the map can be used to update running instances to the new definition. Running instances can be updated only when they're idle and persisted.

Dynamic Update, which is available in both WorkflowApplication and WorkflowServiceHost, is an advanced, complex feature, and it has far more capabilities than I've mentioned. It supports updating activities, providing custom update semantics, emitting tracking information and more. It gives you a rich API you can use in your applications to provide update capabilities for your running instances.

Runtime Enhancements

I want to briefly describe some runtime enhancements in WF 4.5. Due to space constraints, I won't be delving into them with the same level of detail as the other features; these are more scoped and advanced.

Partial Trust WF 4 requires full-trust to execute. WF 4.5 can run in partially trusted AppDomains.

Expressions Extensibility We modified ExpressionTextBox to bind to an object instead of a string. As a result, you can provide your own expression editing experience and you're not limited to a textual representation. We also exposed fast-path capabilities that can be used with code activities for authoring expression activities with better performance.

Visual Basic Performance Enhancements We significantly improved VisualBasicValue/VisualBasicReference performance by changing their internal implementation.

WF 3 Obsoleted WF 3 types have been marked as obsolete in WF 4.5 assemblies.

Wrapping Up

We took WF 4 and, based on your feedback, made it better. We added much-requested features and fixed some key issues to provide you with the best workflow framework for your .NET applications.

WF 4.5 is an in-place replacement for WF 4 and is fully compatible with WF 4. We have thoroughly tested that WF 4.5 doesn't break any WF 4 scenarios; all your WF 4 investments will be fully preserved in WF 4.5 without need of any changes.

If you're writing WF 4 applications, keep doing so! When you're ready to move to WF 4.5, your code will just work, and you and your customers will be able to leverage and enjoy all the improvements described in this article. ■

LEON WELICKI *is a senior program manager on the WF team focusing on the WF programming model and runtime.*

THANKS to the following technical experts for reviewing this article:
Joe Clancy, Dave Cliffe, Dan Glick, Hani Khoshdel Nikkhoo and Isaac Yuen

Creating a NuGet Gallery

Clark Sell and Mark Nichols

This is the third and last article in our series on NuGet, a new package management ecosystem for developers. NuGet, a project of the Outercurve Foundation, gives .NET developers a way to consume, author and publish packages. So far, the series showed you how to manage project libraries with NuGet (msdn.microsoft.com/magazine/hh547106) and how to become a NuGet author (msdn.microsoft.com/magazine/hh708753). In this article, we'll take a look at hosting your own NuGet gallery and creating a build process to help manage your packages.

Why Host the Gallery?

NuGet.org already exists as a public repository for NuGet packages, so you might question the point of hosting a gallery. That's understandable, but what about leveraging that ecosystem infrastructure within the walls of your own dev environment? Why not set up your own private gallery not only to facilitate your product's development ecosystem but, better yet, to tie it directly to your build and deployment processes? Moreover, anything you see at NuGet.org is available to you free.

In this article, we'll explore the steps necessary to create and consume your own NuGet gallery. As we've said before, incorporating NuGet into your development lifecycle isn't complicated and will be well worth the effort. Once you've done it, your package consumption, creation and distribution problems will become no more than a distant memory.

This article relies on beta versions of the NuGet Gallery and NuGetFeed.org.

This article discusses:

- Why you should host your own NuGet gallery
- A fast and easy way to host a NuGet gallery
- Getting the NuGet Gallery source
- Integrating your NuGet gallery with your existing build processes

Technologies discussed:

NuGet, Visual Studio, Windows PowerShell, ASP.NET MVC, TFS Nugetter

Before diving in, let's explore a use case that most developers have encountered: versioning. Your product, depending on its overall size and complexity, is very likely the result of many teams, differing release schedules and a variety of assemblies. Of course the released product is nicely bundled for your customers, but its journey to completion included hundreds if not thousands of builds, churning out many different versions of product assets.

Over the past decade, development practices—including automated build and deployment systems—helped reduce the pain, but they always fell short in terms of distribution and choice. Package management systems provide the solution for this.

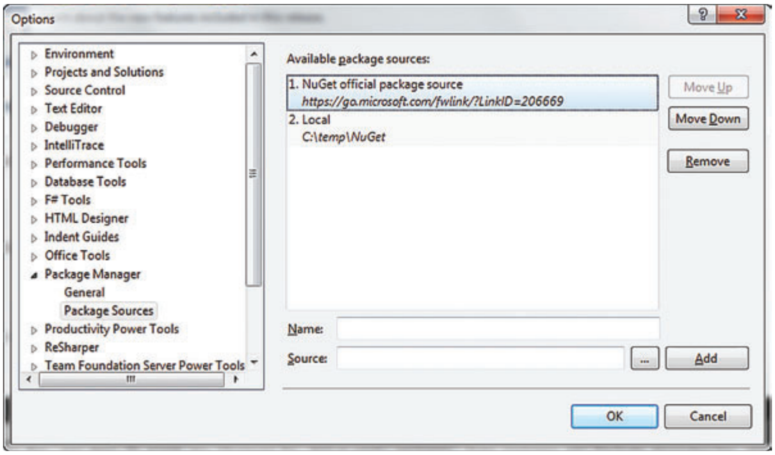
Think of your own development ecosystem, in which different teams work independently to publish the versions they choose for consumption. Each team, with its own build and release schedule, wants control and an easy way to publish its products. With a package management system, each team gets to decide what the product is and how it should get installed, and the customers get to decide when they should consume it.

To some degree, this is very similar to how open source communities operate, using proven practices that easily scale. Although the packages you find on NuGet.org are fairly "large grained," you can follow those same practices for your building your own applications. A good, concrete example of the type of software this would be useful for might be a helper library your company builds and maintains for its public products.

Maybe your company restricts only the libraries and versions its developers may use. This is another great reason for you to host your own proprietary gallery with the approved packages and package versions.

Fast and Simple

The easiest way to host your own NuGet gallery is to expose a file share. This might sound a bit rudimentary, but if you need something fast and simple, this method works extremely well. The share can be structured as you see fit, placing your packages—that is, your *.nupkg files—anywhere on the share. You can refer to the share as a package source.



Available package sources:
1. NuGet official package source
https://go.microsoft.com/fwlink/?LinkID=206669
2. Local
C:\temp\NuGet

Name:
Source:

Figure 1 **Adding a Package Source in Visual Studio**

With the package source created, let's add it to the development environment. You can have as many package sources as desired and easily move between each. To edit your package sources in Visual Studio, navigate to Tools | Options | Package Manager | Package Sources (see **Figure 1**). Note that as of this writing, version 1.5 was current. You should expect slight differences across NuGet versions.

Here you can add, remove and change the default package source. If you prefer, you can do the same in WebMatrix, a free development environment. Just click on the NuGet Gallery icon on the main menu and select Add Source, as shown in **Figure 2**.

As you can see, both figures list two package sources. You can choose to use just one or all at any given time.

Introducing NuGet Gallery

While the file system package source is very easy to set up and start using, it breaks down pretty quickly when you start scaling out. Luckily, NuGet.org was built with exactly this in mind, to provide an open platform for people to take, extend and build on for their own needs. You can find the NuGet Gallery project at github.com/NuGet/NuGetGallery.

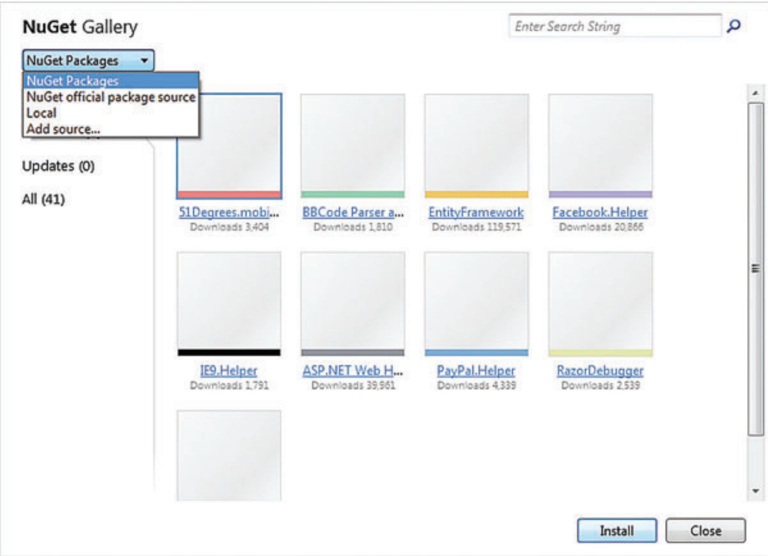


Figure 2 **Adding a Package Source in WebMatrix**

The NuGet Gallery is an ASP.NET MVC 3 project built on top of Razor, OData, SQL Server and Windows Azure.

The best way to understand the features and functionality the NuGet Gallery has to offer your development ecosystem is to simply explore NuGet.org.

Before you can create and publish a package, you need to be a registered user. Once you've registered, you have full rights to upload and manage your packages and edit your profile. You can also generate the unique API key that lets you automate publishing packages to the Gallery (see **Figure 3**). For more information about publishing a package, please see the previous article in this series, "Becoming a NuGet Author."

When you upload a package to the NuGet Gallery, it automatically receives its own place in the Gallery, as shown in **Figure 4**.

This is where you can view all of the important information about a package. For each package, the NuGet Gallery tracks the version downloads, total downloads, dates, licenses and so on. Of course, a package on the NuGet.org site will most likely yield considerably larger download numbers given its ecosystem, but these statistics are vital to any package author, regardless of project size.

The package feed is clearly the most important feature in the overall stack. It's an OData feed you'll find at http://YourGallery/api/v2. This is the same URL you'll use as your package source. Without this feed, all of the power of NuGet is lost.

Getting Started

Before getting started, verify you have the following installed:

- Visual Studio 2010: microsoft.com/visualstudio
- Windows PowerShell 2.0: microsoft.com/powershell
- NuGet: docs.nuget.org/docs/start-here/installing-nuget
- Windows Azure SDK: microsoft.com/windowsazure/sdk

At the time of this writing, there are no installers for the Gallery and the project is under very active development. That means it's always best to go to the project's homepage for the latest information. You'll need to download the source and build it, which is simple.

To get the source, you can either just download a zipped copy of the latest master branch or clone the source base with git, like so:

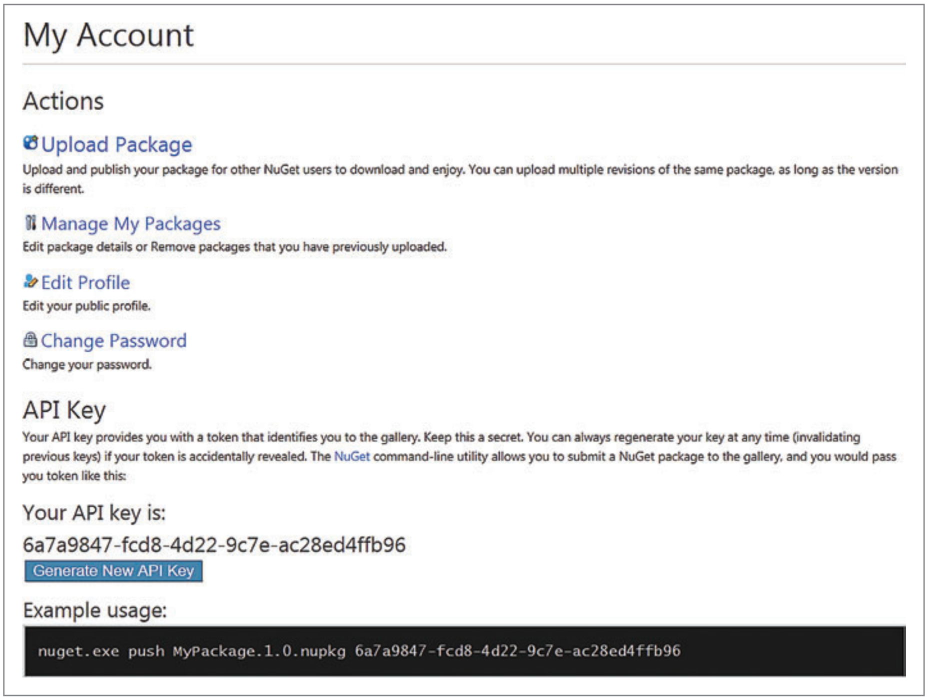
```
$ git clone https://github.com/NuGet/NuGetGallery.git
```

When you've done this, you'll have the entire source base locally. At its root you'll find a Windows PowerShell script called Build-Solution.ps1. Run this script to set up your machine and build the source.

If you run the source, you'll notice it looks and feels exactly like NuGet.org. You might be thinking of tweaking the UI, perhaps modifying the homepage. That's understandable, but be cautious, especially if your intentions are to stay in sync with the master source base.

Build Process Meets NuGet Gallery

The next logical step is integrating your NuGet gallery with your existing build processes. If you don't have a



My Account

Actions

Upload Package

Upload and publish your package for other NuGet users to download and enjoy. You can upload multiple revisions of the same package, as long as the version is different.

Manage My Packages

Edit package details or Remove packages that you have previously uploaded.

Edit Profile

Edit your public profile.

Change Password

Change your password.

API Key

Your API key provides you with a token that identifies you to the gallery. Keep this a secret. You can always regenerate your key at any time (invalidating previous keys) if your token is accidentally revealed. The NuGet command-line utility allows you to submit a NuGet package to the gallery, and you would pass you token like this:

Your API key is:

6a7a9847-fcd8-4d22-9c7e-ac28ed4ffb96

Example usage:

```
nuget.exe push MyPackage.1.0.nupkg 6a7a9847-fcd8-4d22-9c7e-ac28ed4ffb96
```

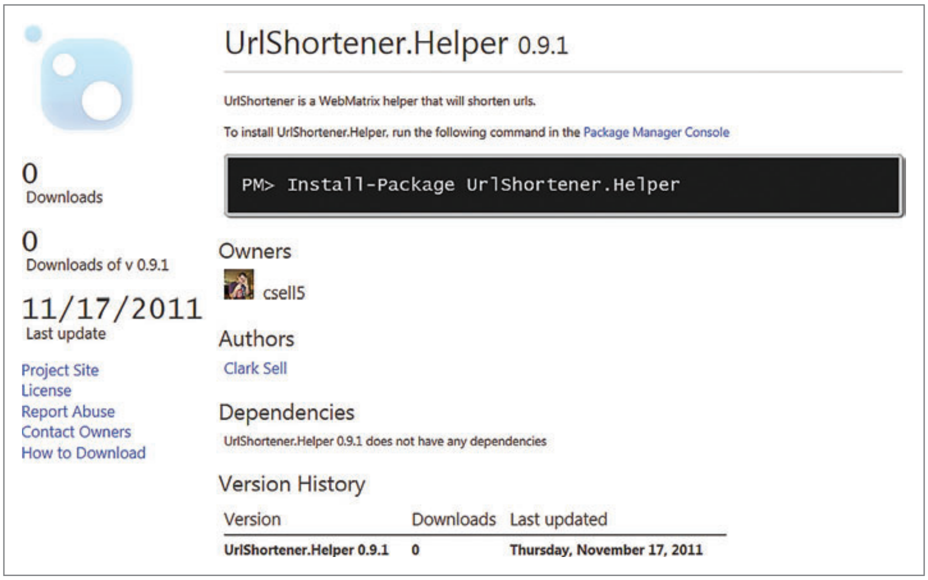
Figure 3 **A Registered User Account in NuGet**

build process and you're looking for a little "getting started" guidance with Team Foundation Server (TFS) build processes, check out the Microsoft Visual Studio ALM Rangers Build Customization Guide at rabcg.codeplex.com.

All right: You have a library you want to make available on a gallery, which could be the official NuGet Gallery, a proprietary gallery or even a local file share. No matter which, you'll have to perform certain tasks to create the NuGet package, and they're generally the same tasks no matter where the gallery resides.

Through the course of developing your product, you'll repetitively go through the process of coding, compiling, packaging, pushing and publishing. You'll also, we hope, include a couple of healthy doses of testing; results from that testing might make you deviate from publishing to the gallery if you find any issues (see **Figure 5**).

Keeping consumers happy with working software is a good thing, and you'll have an easier time doing so using a managed process. Remember, though, the managed process can't be entirely automated; there's no replacement for human verification. Don't assume you can publish just because your software passes all the automated tests. You could easily publish a package that isn't ready and make your consumers angry. Moreover, they won't be happy if you publish constantly and they can't keep up with the ever-changing versions. Make publishing a conscious decision.



UrlShortener.Helper 0.9.1

UrlShortener is a WebMatrix helper that will shorten urls.

To install UrlShortener.Helper, run the following command in the Package Manager Console

```
PM> Install-Package UrlShortener.Helper
```

0
Downloads

0
Downloads of v 0.9.1

11/17/2011
Last update

Owners

csell5

Authors

Clark Sell

Dependencies

UrlShortener.Helper 0.9.1 does not have any dependencies

Version History

Version	Downloads	Last updated
UrlShortener.Helper 0.9.1	**0**	**Thursday, November 17, 2011**

Figure 4 **A Package Uploaded to the NuGet Gallery**

Use the build process to build, package and push your NuGet library to a gallery where you perform final tests and do that human verification. That way, you can choose continuous integration or a scheduled build to automatically push the package and make it ready for you to test. How might you do that? Here's an example using TFS and its workflow-based automated build capability.

NuGet.exe is a stand-alone, parameter-driven console application, but it can easily be called by another process, such as TFS Team Build. Team Build executes a workflow; in TFS terms, this is a set of custom activities controlled through a build template and initiated by a build definition. One packaged solution for automating the NuGet development process is the NuGetter project at nugetter.codeplex.com. This project provides an automated shell around the NuGet.exe application. We'll use this package to show what can be done and things to think about as you work toward automating your own NuGet packages.

As **Figure 6** shows, the build definition in TFS is where you provide all the necessary information to do the packaging, push and publish. Building a flexible and repeatable build process for NuGet requires several pieces of data and the switches that turn portions of the process on or off. The data is organized into categories that are titled to give you a sense of the order in which the steps are executed.

The PrePackaging section is used for complicated or multiframework NuGet packaging that requires some initial preparation. NuGetter can call a Windows PowerShell script (if you desire) to organize the

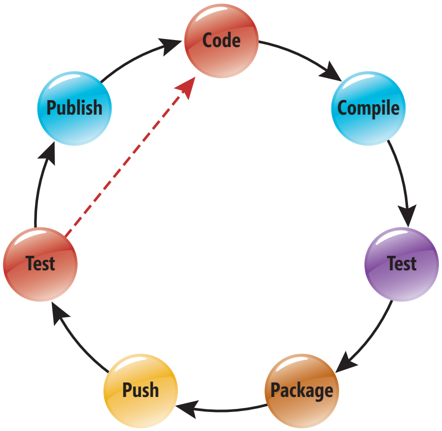


Figure 5 **The NuGet Development Process**

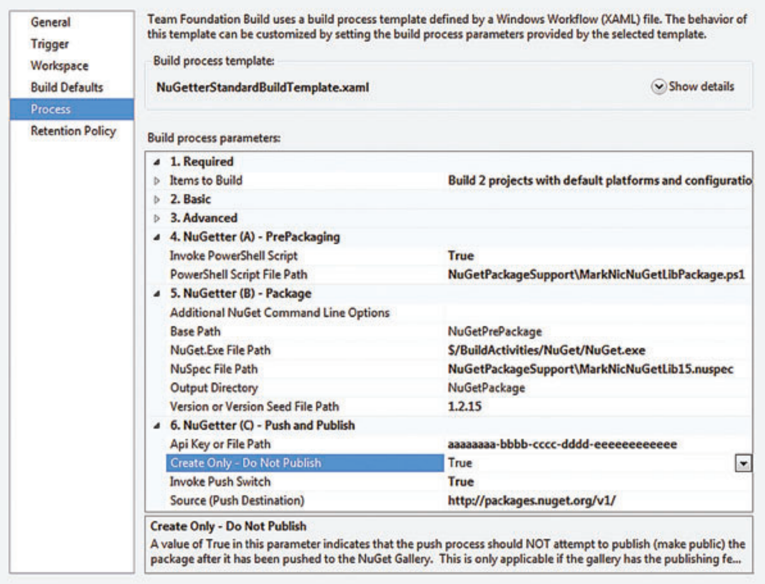
library files to make packaging easier. This step isn't required, so you can use the flag parameter "Invoke PowerShell Script" to tell the process whether to execute the script.

You can use a .nuspec file directly to define how your package will be created or you can do so indirectly through a .csproj file. It's up to you; use the approach that better suits your needs.

As you can see, all of the parameters required for packaging are included. Version, base path (source folder for packaging), API Key and gallery location (called Source) are all provided as part of the build definition. However, the security of this information might be of great importance to you. For example, the API Key is what allows you and only you to push NuGet packages to the gallery. If that key becomes public, anyone could push bogus packages in your name. Because of this, NuGetter allows you to provide either the actual API Key or a path to a file that contains the key. If you use a file path, the build process reads the file, extracts the key and uses it without listing the key in the log, and security is maintained. Obviously, the build service ID must have read access to the file for this to work.

Another potential issue for you is managing the version of your package. In **Figure 6**, the version number is provided in the build definition. But, do you really want to keep changing that number in the build definition? What happens when you have several builds (continuous integration [CI], daily, other scheduled and various manual builds)? NuGetter gives you the choice to enter the version directly or, as with the API Key, you can provide a file path that multiple build definitions can use. They will all use the same version number and you can manage it in just one place.

The Push Destination in the example is the NuGet Gallery, but this is where you provide the build destination even if you're pushing your package to an internal gallery or to a local file store. This is another case where the build process might need to intervene. NuGet.exe expects a URL as the destination and if you're pushing to a local store—which won't be a URL—the process needs to interpret the destination format. In this case, instead of using NuGet.exe to do the pushing, you have the build process do it.



General	Team Foundation Build uses a build process template defined by a Windows Workflow (XAML) file. The behavior of this template can be customized by setting the build process parameters provided by the selected template.	
Trigger	Build process template:	
Workspace	NuGetterStandardBuildTemplate.xaml	Show details
Build Defaults		
Process	Build process parameters:	
Retention Policy		

1. Required	
Items to Build	Build 2 projects with default platforms and configuratio
2. Basic	
3. Advanced	
4. NuGetter (A) - PrePackaging	
Invoke PowerShell Script	True
PowerShell Script File Path	NuGetPackageSupport\MarkNicNuGetLibPackage.ps1
5. NuGetter (B) - Package	
Additional NuGet Command Line Options	
Base Path	NuGetPrePackage
NuGet.Exe File Path	$/BuildActivities/NuGet/NuGet.exe
NuSpec File Path	NuGetPackageSupport\MarkNicNuGetLib15.nuspec
Output Directory	NuGetPackage
Version or Version Seed File Path	1.2.15
6. NuGetter (C) - Push and Publish	
Api Key or File Path	aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee
Create Only - Do Not Publish	True
Invoke Push Switch	True
Source (Push Destination)	http://packages.nuget.org/v1/

Create Only - Do Not Publish
A value of True in this parameter indicates that the push process should NOT attempt to publish (make public) the package after it has been pushed to the NuGet Gallery. This is only applicable if the gallery has the publishing fe...

Figure 6 **Nugetter Build Template**

For reasons of completeness and feature parity with NuGet.exe, NuGetter does provide for automated publishing. Just keep in mind our earlier caveats before deciding to use this capability.

By the way, you don't have to worry that your consumers might miss any package updates. The community has this covered with NuGetFeed (github.com/NuGetFeed/NuGetFeed). NuGetFeed lets you build a custom RSS feed based on the packages you select. Your consumers can add a custom feed to their favorite RSS reader and be informed of any updates.

Another option for creating a NuGet feed is MyGet at MyGet.org. MyGet might be perfect for those of you who want a private NuGet feed for your list of approved packages but don't want to spend the time, money, and effort creating and maintaining your own gallery infrastructure. MyGet is a hosted solution that allows you to create galleries very quickly that are specific to your needs. Refer to the MyGet site for more detailed information.

Finally, as we mentioned earlier, NuGet and NuGet Gallery are open source projects. This means you're empowered to contribute anything from documentation to features, or smash a few bugs along the way. On the github homepage, the project team has detailed the exact steps you can take to contribute.

Wrapping Up

NuGet has no doubt changed the way we think about our .NET package management ecosystem, both public and private. Both NuGet Gallery and NuGetFeed are essential components to completing that ecosystem. Bringing these tools into your daily routine opens new possibilities. As noted, NuGet Gallery is in active development, so make sure to visit github.com/NuGet/NuGetGallery for the most up-to-date information.

You can find all of the links used in this article and more at on.csell.net/HostingNuGet. ■

CLARK SELL *is as a senior Web evangelist for Microsoft outside of Chicago. He podcasts at DeveloperSmackdown.com, blogs at csell.net and can be found on Twitter at twitter.com/csell5.*

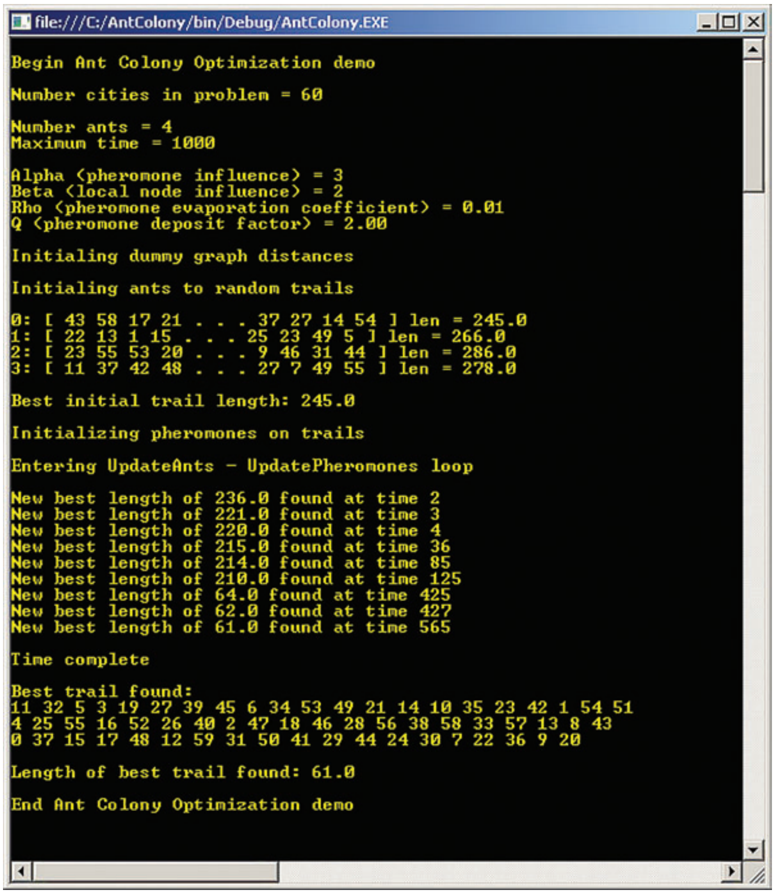
MARK NICHOLS *is as a senior software consultant for Microsoft outside of Chicago. He podcasts at DeveloperSmackdown.com, blogs at logs at marnick.net and can be found on Twitter at twitter.com/mark_nic.*

THANKS to the following technical experts for reviewing this article: *David Ebbo, Phil Haack and Brandon Satrom*

TEST RUN JAMES MCCAFFREY

Ant Colony Optimization

In this month's column I present C# code that implements an Ant Colony Optimization (ACO) algorithm to solve the Traveling Salesman Problem (TSP). An ACO algorithm is an artificial intelligence technique based on the pheromone-laying behavior of ants; it can be used to find solutions to exceedingly complex problems that seek the optimal path through a graph. The best way to see where I'm headed is to take a look at the screenshot in **Figure 1**. In this case, the demo is solving an instance of the TSP with the goal of finding the shortest path that visits each of 60 cities exactly once. The demo program uses four ants; each ant represents a potential solution. ACO requires the specification of several parameters such as the pheromone influence factor (alpha) and the pheromone evaporation coefficient (rho), which I'll explain later. The four ants are initialized to random trails through the 60 cities; after initialization, the best ant has a shortest trail length of 245.0 units. The key idea of ACO is the use of simulated pheromones, which attract ants to better trails through the graph. The main processing loop alternates between updating the ant trails based on the current pheromone values and updating the pheromones based on the new ant trails. After the maximum number of times (1,000) through the main processing loop, the program displays the best trail found and its corresponding length (61.0 units).



```
Begin Ant Colony Optimization demo

Number cities in problem = 60

Number ants = 4
Maximum time = 1000

Alpha <pheromone influence> = 3
Beta <local node influence> = 2
Rho <pheromone evaporation coefficient> = 0.01
Q <pheromone deposit factor> = 2.00

Initialing dummy graph distances

Initialing ants to random trails

0: [ 43 58 17 21 . . . 37 27 14 54 ] len = 245.0
1: [ 22 13 1 15 . . . 25 23 49 5 ] len = 266.0
2: [ 23 55 53 20 . . . 9 46 31 44 ] len = 286.0
3: [ 11 37 42 48 . . . 27 7 49 55 ] len = 278.0

Best initial trail length: 245.0

Initializing pheromones on trails

Entering UpdateAnts - UpdatePheromones loop

New best length of 236.0 found at time 2
New best length of 221.0 found at time 3
New best length of 220.0 found at time 4
New best length of 215.0 found at time 36
New best length of 214.0 found at time 85
New best length of 210.0 found at time 125
New best length of 64.0 found at time 425
New best length of 62.0 found at time 427
New best length of 61.0 found at time 565

Time complete

Best trail found:
11 32 5 3 19 27 39 45 6 34 53 49 21 14 10 35 23 42 1 54 51
4 25 55 16 52 26 40 2 47 18 46 28 56 38 58 33 57 13 8 43
0 37 15 17 48 12 59 31 50 41 29 44 24 30 7 22 36 9 20

Length of best trail found: 61.0

End Ant Colony Optimization demo
```

Figure 1 **Ant Colony Optimization in Action**

The 60-city graph is artificially constructed so that every city is connected to every other city, and the distance between any two cities is a random value between 1.0 and 8.0 arbitrary units (miles, km and so forth). There's no easy way to solve the TSP. With 60 cities, assuming you can start at any city and go either forward or backward, and that all cities are connected, there are a total of (60 - 1)! / 2 = 69,341,559,272,844,917, 868,969,509,860,194,703,172,951,438,386,343,716,270,410,647,470, 080,000,000,000,000 possible solutions. Even if you could evaluate 1 billion possible solutions per second, it would take about 2.2 * 10^{63} years to check them all, which is many times longer than the estimated age of the universe.

ACO is a meta-heuristic, meaning that it's a general framework that can be used to create a specific algorithm to solve a specific graph path problem. Although ACO was proposed in a 1991 doctoral thesis by M. Dorigo, the first detailed description of the algorithm is generally attributed to a 1996 follow-up paper by M. Dorigo, V. Maniezzo and A. Colorni. Since then, ACO has been widely studied and modified, but, somewhat curiously, very few complete and correct implementations are available online.

This column assumes you have intermediate-to-advanced programming skills. I implement the ACO program using C#, but you shouldn't have too much trouble refactoring my code to a different language, such as JavaScript. I decided to avoid all use of object-oriented programming (OOP) to keep the ideas of the algorithm as clear as possible. Because of space limitations, I can't present all of the code shown running in **Figure 1**. I'll go over the trickiest parts and you can download the complete code from code.msdn.microsoft.com/mag201202TestRun. Although you might never use ACO code directly, many of its techniques, such as roulette wheel selection, can be interesting and useful additions to your technical skill set.

Code download available at code.msdn.microsoft.com/mag201202TestRun.

Figure 2 **Ant Colony Optimization Program Structure**

```
using System;

namespace AntColony
{
  class AntColonyProgram
  {
    static Random random = new Random(0);
    static int alpha = 3;
    static int beta = 2;
    static double rho = 0.01;
    static double Q = 2.0;

    static void Main(string[] args)
    {
      try
      {
        Console.WriteLine("\nBegin Ant Colony Optimization demo\n");
        int numCities = 60;
        int numAnts = 4;
        int maxTime = 1000;

        int[][] dists = MakeGraphDistances(numCities);
        int[][] ants = InitAnts(numAnts, numCities);

        int[] bestTrail = BestTrail(ants, dists);
        double bestLength = Length(bestTrail, dists);

        double[][] pheromones = InitPheromones(numCities);

        int time = 0;
        while (time < maxTime)
        {
          UpdateAnts(ants, pheromones, dists);
          UpdatePheromones(pheromones, ants, dists);

          int[] currBestTrail = BestTrail(ants, dists);
          double currBestLength = Length(currBestTrail, dists);
          if (currBestLength < bestLength)
          {
            bestLength = currBestLength;
            bestTrail = currBestTrail;
          }
          ++time;
        }

        Console.WriteLine("\nBest trail found:");
        Display(bestTrail);
        Console.WriteLine("\nLength of best trail found: " +
          bestLength.ToString("F1"));
```

```
        Console.WriteLine("\nEnd Ant Colony Optimization demo\n");
      }
      catch (Exception ex)
      {
        Console.WriteLine(ex.Message);
      }
    } // Main

    static int[][] InitAnts(int numAnts, int numCities) { . . }

    static int[] RandomTrail(int start, int numCities) { . . }

    static int IndexOfTarget(int[] trail, int target) { . . }

    static double Length(int[] trail, int[][] dists) { . . }

    static int[] BestTrail(int[][] ants, int[][] dists) { . . }

    static double[][] InitPheromones(int numCities) { . . }

    static void UpdateAnts(int[][] ants, double[][] pheromones,
      int[][] dists) { . . }

    static int[] BuildTrail(int k, int start, double[][] pheromones,
      int[][] dists) { . . }

    static int NextCity(int k, int cityX, bool[] visited, double[][] pheromones,
      int[][] dists) { . . }

    static double[] MoveProbs(int k, int cityX, bool[] visited,
      double[][] pheromones, int[][] dists) { . . }

    static void UpdatePheromones(double[][] pheromones, int[][] ants,
      int[][] dists) { . . }

    static bool EdgeInTrail(int nodeX, int nodeY, int[] trail) { . . }

    static int[][] MakeGraphDistances(int numCities) { . . }

    static double Distance(int cityX, int cityY, int[][] dists) { . . }

    static void Display(int[] trail) { . . }

    static void ShowAnts(int[][] ants, int[][] dists) { . . }

    static void Display(double[][] pheromones) { . . }
  }
}
```

Program Structure

I implemented the ACO demo program as a single C# console application. The overall structure of the program, with most WriteLine statements removed, is shown in **Figure 2**. Although some parts are tricky, the program isn't as complicated as **Figure 2** suggests because many of the methods are short helper methods.

I used Visual Studio to create a console application program named AntColony. In the Solution Explorer window I renamed the default Program.cs file to AntColonyProgram.cs, which automatically renamed the single class in the project. I deleted all the template-generated using statements except for the System namespace—ACO typically doesn't need much library support. The two key methods are UpdateAnts and UpdatePheromones. Method UpdateAnts calls helper BuildTrail, which calls NextCity, which calls MoveProbs. Method UpdatePheromones calls helper EdgeInTrail, which calls IndexOfTarget.

I declared a class-scope Random object named random. ACO algorithms are probabilistic as you'll see shortly. The class-scope variables alpha, beta, rho and Q control the behavior of the ACO algorithm. I use these variable names because they were used in the original description of ACO.

Setting up the Problem

I used method MakeGraphDistances to set up an artificial graph:

```
static int[][] MakeGraphDistances(int numCities)
{
  int[][] dists = new int[numCities][];
  for (int i = 0; i < dists.Length; ++i)
    dists[i] = new int[numCities];
  for (int i = 0; i < numCities; ++i)
    for (int j = i + 1; j < numCities; ++j) {
      int d = random.Next(1, 9); // [1,8]
      dists[i][j] = d; dists[j][i] = d;
    }
  return dists;
}
```

For a real-world graph problem, you'd probably read graph-adjacency and distance-between-node data from a text file into some sort of data structure. Here I simulated a graph by creating an array of arrays where the row index i represents the from-city and the column index j represents the to-city. Notice that all cities are connected, distances are symmetric and the distance from a city to itself is 0.

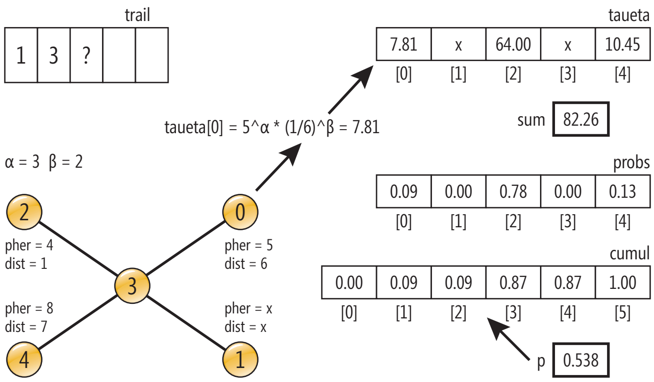


Figure 3 **Updating Ant Information**

Once I have a distances data structure I can use it for a Distance method:

```
static double Distance(int cityX, int cityY, int[][] dists)
{
  return dists[cityX][cityY];
}
```

To minimize the amount of code presented, I've omitted normal error checking, such as making sure that the cityX and cityY parameters are valid.

Initiating the Ants and the Pheromones

In this non-OOP implementation, an ant is simply an array of int values that represent the trail, or path, from an initial city through all other cities. A collection of ants is an array of arrays in which the first index indicates the ant:

```
static int[][] InitAnts(int numAnts, int numCities)
{
  int[][] ants = new int[numAnts][];
  for (int k = 0; k < numAnts; ++k) {
    int start = random.Next(0, numCities);
    ants[k] = RandomTrail(start, numCities);
  }
  return ants;
}
```

The initialization method allocates a row for the trail for each ant, picks a random start city and then calls a helper method RandomTrail:

```
static int[] RandomTrail(int start, int numCities)
{
  int[] trail = new int[numCities];
  for (int i = 0; i < numCities; ++i) { trail[i] = i; }

  for (int i = 0; i < numCities; ++i) {
    int r = random.Next(i, numCities);
    int tmp = trail[r]; trail[r] = trail[i]; trail[i] = tmp;
  }

  int idx = IndexOfTarget(trail, start);
  int temp = trail[0]; trail[0] = trail[idx]; trail[idx] = temp;

  return trail;
}
```

The RandomTrail helper allocates a trail and initializes it to 0, 1, 2, ... numCities-1. Next, the method uses the Fisher-Yates shuffle algorithm to randomize the order of the cities in the trail. The specified start city is then located and swapped into position trail[0].

Pheromones are chemicals ants place on their trails; they attract other ants. More ants will travel on a shorter trail to a food source—and deposit more pheromones—than on longer trails. The pheromones slowly evaporate over time. Here's method InitPheromones:

```
static double[][] InitPheromones(int numCities)
{
  double[][] pheromones = new double[numCities][];
  for (int i = 0; i < numCities; ++i)
    pheromones[i] = new double[numCities];
  for (int i = 0; i < pheromones.Length; ++i)
    for (int j = 0; j < pheromones[i].Length; ++j)
      pheromones[i][j] = 0.01;
  return pheromones;
}
```

Pheromone information is stored in an array-of-arrays-style symmetric matrix where the row index i is the from-city and the column index j is the to-city. All values are initially set to an arbitrary small value (0.01) to jump start the UpdateAnts-UpdatePheromones cycle.

Updating the Ants

The key to the ACO algorithm is the process that updates each ant and trail by constructing a new—we hope better—trail based on the pheromone and distance information. Take a look at **Figure 3**. Suppose we have a small graph with just five cities. In **Figure 3** the new trail for an ant is under construction. The trail starts at city 1, then goes to city 3, and the update algorithm is determining the next city. Now suppose the pheromone and distance information is as shown in the image. The first step in determining the next city is constructing an array I've called "taueta" (because the original research paper used the Greek letters tau and eta). The taueta value is the value of the pheromone on the edge raised to the alpha power, times one over the distance value raised to the beta power. Recall that alpha and beta are global constants that must be specified. Here I'll assume that alpha is 3 and beta is 2. The taueta values for city 1 and city 3 aren't computed because they're already in the current trail. Notice that larger values of the pheromone increase taueta, but larger distances decrease taueta.

Figure 4 **The BuildTrail Method**

```
static int[] BuildTrail(int k, int start, double[][] pheromones,
  int[][] dists)
{
  int numCities = pheromones.Length;
  int[] trail = new int[numCities];
  bool[] visited = new bool[numCities];
  trail[0] = start;
  visited[start] = true;
  for (int i = 0; i < numCities - 1; ++i) {
    int cityX = trail[i];
    int next = NextCity(k, cityX, visited, pheromones, dists);
    trail[i + 1] = next;
    visited[next] = true;
  }
  return trail;
}
```

Figure 5 **The NextCity Method**

```
static int NextCity(int k, int cityX, bool[] visited,
  double[][] pheromones, int[][] dists)
{
  double[] probs = MoveProbs(k, cityX, visited, pheromones, dists);

  double[] cumul = new double[probs.Length + 1];
  for (int i = 0; i < probs.Length; ++i)
    cumul[i + 1] = cumul[i] + probs[i];

  double p = random.NextDouble();

  for (int i = 0; i < cumul.Length - 1; ++i)
    if (p >= cumul[i] && p < cumul[i + 1])
      return i;
  throw new Exception("Failure to return valid city in NextCity");
}
```

After all the taueta values have been computed, the next step is to convert those values to probabilities and place them in an array I've labeled probs. The algorithm sums the taueta values, getting 82.26 in this example, and then divides each taueta value by the sum. At this point, city 0 has a probability of 0.09 of being selected and so on. Next, the algorithm needs to select the next city based on the computed probabilities. As I mentioned earlier, the ACO algorithm I'm presenting in this article uses a neat technique called roulette wheel selection. I constructed an augmented array called cumul, which holds cumulative probabilities. The size of the augmented array is one greater than the probs array, and cell [0] is seeded with 0.0. Each cell in cumul is the cumulative sum of the probabilities. After the cumul array has been constructed, a random number p between 0.0 and 1.0 is generated. Suppose p = 0.538 as shown. That p value falls between the values at [2] and [3] in the cumul array, which means that [2], or city 2, is selected as the next city.

The top-level method for updating is named UpdateAnts:

```
static void UpdateAnts(int[][] ants, double[][] pheromones, int[][]
dists)
{
  int numCities = pheromones.Length;
  for (int k = 0; k < ants.Length; ++k) {
    int start = random.Next(0, numCities);
    int[] newTrail = BuildTrail(k, start, pheromones, dists);
    ants[k] = newTrail;
  }
}
```

Notice that each ant is assigned a new, random starting city rather than preserving the old start city. Most of the actual work is performed by helper method BuildTrail, as shown in **Figure 4**.

BuildTrail maintains an array of Boolean visited, so that the trail created doesn't contain duplicate cities. The value at trail[0] is seeded with a start city, then each city is added in turn by helper method NextCity, shown in **Figure 5**.

The NextCity method implements the roulette wheel selection algorithm.Note that the algorithm will fail if the last value in the cumul array is larger than 1.00 (possibly due to rounding errors), and so you might want to add logic to always set cumul[cumul.Length-1] to 1.00. NextCity requires an array of probabilities produced by helper method MoveProbs, shown in **Figure 6**.

Figure 6 **The MoveProbs Method**

```
static double[] MoveProbs(int k, int cityX, bool[] visited,
  double[][] pheromones, int[][] dists)
{
  int numCities = pheromones.Length;
  double[] taueta = new double[numCities];
  double sum = 0.0;
  for (int i = 0; i < taueta.Length; ++i) {
    if (i == cityX)
      taueta[i] = 0.0; // Prob of moving to self is zero
    else if (visited[i] == true)
      taueta[i] = 0.0; // Prob of moving to a visited node is zero
    else {
      taueta[i] = Math.Pow(pheromones[cityX][i], alpha) *
        Math.Pow((1.0 / Distance(cityX, i, dists)), beta);
      if (taueta[i] < 0.0001)
        taueta[i] = 0.0001;
      else if (taueta[i] > (double.MaxValue / (numCities * 100)))
        taueta[i] = double.MaxValue / (numCities * 100);
    }
    sum += taueta[i];
  }

  double[] probs = new double[numCities];
  for (int i = 0; i < probs.Length; ++i)
    probs[i] = taueta[i] / sum;
  return probs;
}
```

The taueta values can be very small (if the distance value is very large) or very large (if the pheromone value is large), which can produce difficulties for the algorithm. To deal with this, I check the taueta values and impose arbitrary min and max values.

Updating the Pheromones

Updating pheromone information is much easier than updating the ant trail information. The key lines of code in method UpdatePhermones are:

```
double length = Length(ants[k], dists);
double decrease = (1.0 - rho) * pheromones[i][j];
double increase = 0.0;
if (EdgeInTrail(i, j, ants[k]) == true)
  increase = (Q / length);
pheromones[i][j] = decrease + increase;
```

Each pheromone value is decreased, simulating evaporation, and increased, simulating the deposit of pheromones by ants on the trail. The decrease effect is produced by multiplying the current pheromone value by a factor less than 1.0 that depends on global parameter rho. The larger rho is, the greater the decrease in pheromone value. The increase effect is produced by adding a proportion of the current ant's total trail length, where the proportion is determined by global parameter Q. Larger values of Q increase the amount of pheromone added. Method UpdatePheromones calls helper EdgeInTrail, which determines if a segment between two cities is on the ant's current trail. You can check out the code download for this article for the details (code.msdn.microsoft.com/mag201202TestRun).

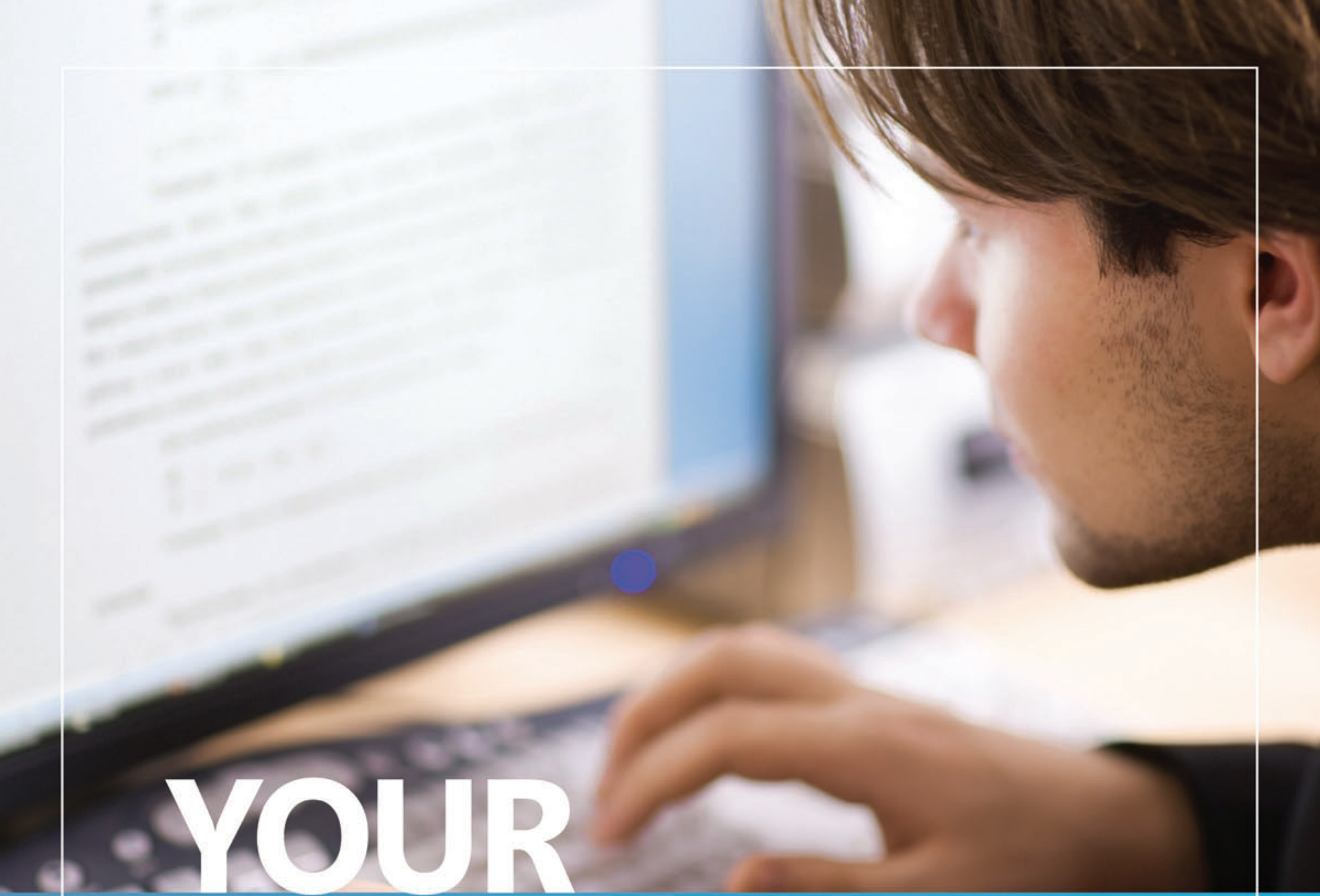
Wrapping Up

Let me emphasize that there are many variations of ACO; the version I've presented here is just one of many possible approaches. ACO advocates have applied the algorithm to a wide range of combinatorial optimization problems. Other combinatorial optimization algorithms based on the behavior of natural systems include Simulated Annealing (SA), which I covered last month (msdn.microsoft.com/magazine/hh708758), and Simulated Bee Colony (SBC), which I covered in my April 2011 column (msdn.microsoft.com/magazine/gg983491). Each approach has strengths and weaknesses. In my opinion, ACO is best-suited for problems that closely resemble the Traveling Salesman Problem, while SA and SBC are better for more general combinatorial optimization problems, such as scheduling.

ACO, in common with other meta-heuristics based on natural systems, is quite sensitive to your choice of free global parameters—alpha, beta and so on. Although there has been quite a bit of research on ACO parameters, the general consensus is that you must experiment a bit with free parameters to get the best combination of performance and solution quality. ■

DR. JAMES MCCAFFREY *works for Volt Information Sciences Inc., where he manages technical training for software engineers working at the Microsoft Redmond, Wash., campus. He's worked on several Microsoft products including Internet Explorer and MSN Search. McCaffrey is the author of ".NET Test Automation Recipes" (Apress, 2006) and can be reached at jmccaffrey@volt.com or jammc@microsoft.com.*

THANKS to the following Microsoft technical experts for reviewing this article: *Dan Liebling and Anne Loomis Thompson*



Talk to Me: Voice and SMS in the Cloud

This past October found me doing some charity work with GiveCamp in Seattle. (Don't know what GiveCamp is? Take a second and have a look: givecamp.org.) While there, I ran across a group that was interested in doing some SMS messaging as part of an application the members wanted to build. We got to talking, and the subject of interactive-voice applications came up. They were interested in doing something like that (specifically, setting up some automated thank-you calls to donors), but figured you had to run your own call center and install your own PBX software and hardware to make it work.

Au contraire, mes amis.

As luck would have it, at the Philadelphia Emerging Technology Event (also known as Philly ETE) early last year, I met some guys from Voxeo, an enterprise-class PBX system, and they introduced me to Tropo, their cloud-hosted voice-and-SMS (among other things) solution. And for anything telephony-related, folks, it's worth a look.

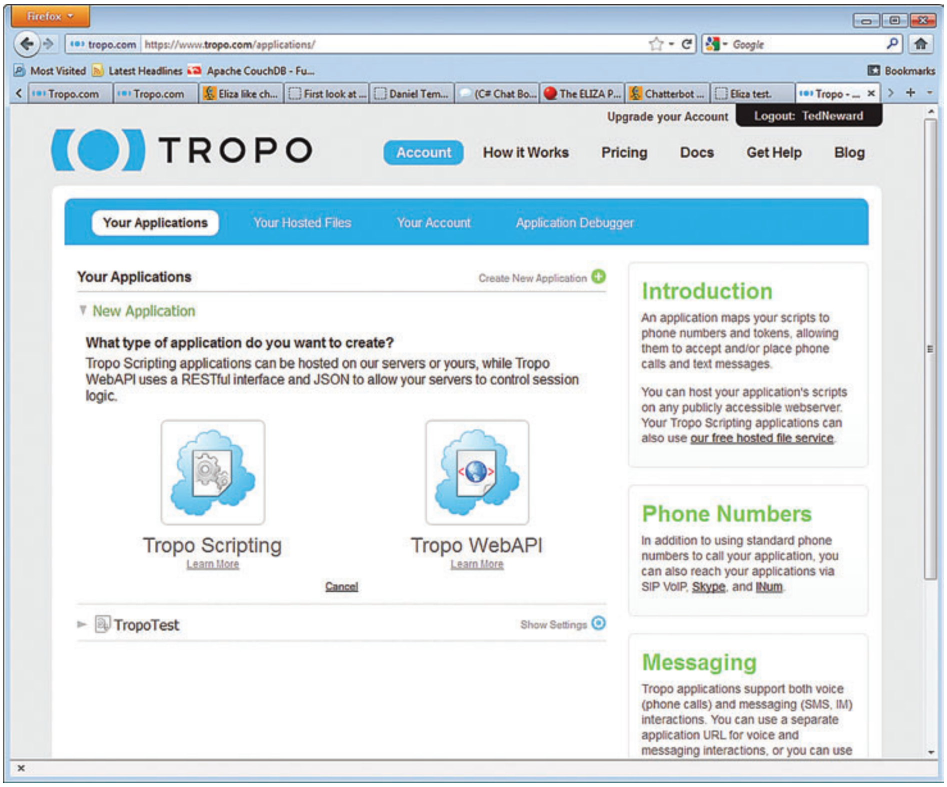
Tropo: A Testamonial

Fundamentally, Tropo isn't all that different from some of the other voice/SMS services available, but it has one distinct advantage over the others I've looked at: during your development cycle, no money changes hands. Building an app that uses this service during development is trivial—just wander on over to the Tropo Web site (tropo.com) and sign up, and the full suite of services is available and at your fingertips for as long as you want.

Naturally, there are other voice/SMS services, and it's always worth examining the alternatives. But for this article (and its successor), I'm going to use Tropo's services. *Caveat emptor*.

Getting Started

As with any other of the cloud-based services, getting started with Tropo requires creating an account on its servers and responding to the e-mail verification. Once that's done, logging back in to Tropo will show the account dashboard, and it's here that the fun begins.



Your Applications

New Application

What type of application do you want to create?

Tropo Scripting applications can be hosted on our servers or yours, while Tropo WebAPI uses a RESTful interface and JSON to allow your servers to control session logic.

Introduction

An application maps your scripts to phone numbers and tokens, allowing them to accept and/or place phone calls and text messages.

You can host your application's scripts on any publicly accessible webserver. Your Tropo Scripting applications can also use our free hosted file service.

Phone Numbers

In addition to using standard phone numbers to call your application, you can also reach your applications via SIP VoIP, Skype, and INum.

Messaging

Tropo applications support both voice (phone calls) and messaging (SMS, IM) interactions. You can use a separate application URL for voice and messaging interactions, or you can use

Figure 1 **Creating an Application in Tropo**

Just to show you how much fun doing this can be, try it now. Pick up the phone and dial 530-206-0504.

Hello, World!

As most readers of this magazine well know, tradition and the Gods of Computer Science both demand that the first application in any new language or platform be the "Hello, world!" application. Far be it from me to buck tradition (at least, when it serves my purpose to stay with it, anyway), so the first step here is to create a new Tropo application and create a simple voice greeting with it. But before we get too far with that, let's make sure we're clear on the architecture here.

Like most cloud-hosted services, Tropo owns and maintains the servers on which the telephony hardware runs, and as with most cloud-hosted applications, that means the application developers don't have any hardware—both a blessing and a curse in most scenarios. In this case, however, most of the "curse" end of the cloud falls away, because we're not going to ask Tropo to host any data for us. In fact, Tropo doesn't even have to host the scripts that drive the application. It can and quite happily will, but if that's of concern, then the script can be pulled from any arbitrary URL and executed on Tropo's servers. For this piece, we're going to use Tropo-hosted files, just because that seems easier to start.

Once the e-mail has arrived and verification is taken care of, logging back in to Tropo should reveal a dashboard.

Click "Create an application," which will take you to **Figure 1**.

> Tropo contains some speech-to-text translators and will attempt to parse the spoken response from the caller.

Choose "Tropo Scripting" and give it a name; I used "HelloMSDN" for this example. Finally, click on the "Hosted File" link and choose "Create a new hosted file for this application"; a simple text editor will pop up at that point. As you might guess, you're building a small script file (in your choice of scripting languages—JavaScript, PHP, Ruby, Groovy or Python) that will be fired when Tropo is told to "execute" this script, which in this case will be when somebody dials a phone number that Tropo will give you. (More on this later.)

Call the file "HelloMSDNScript.js" (the .js extension being important, to tell Tropo that this is a JavaScript script), and in the body of the file, put the following:

```
say("Greetings, MSDN fans!")
```

When that's done, it should look like **Figure 2**.

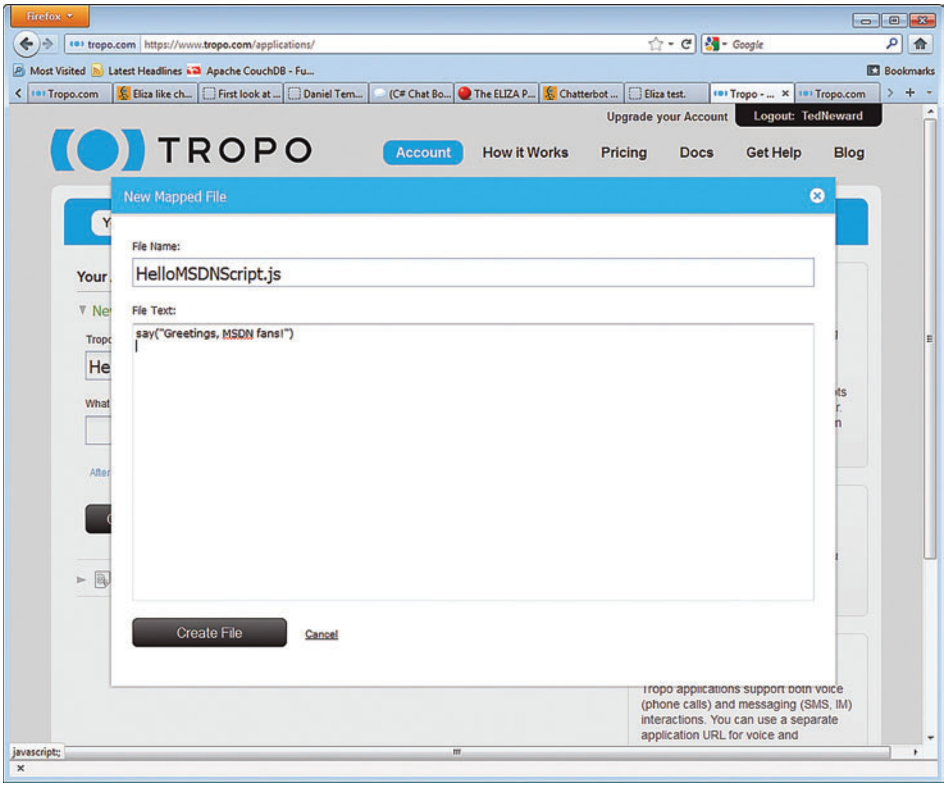
If all is well, click "Create File," and then "Create Application." Once that's done, Tropo will take you back to the application dashboard, which will look a little different now, as you can see in **Figure 3**.

This dashboard is particularly important, because this is where you're going to have some control over the channels to which Tropo is listening. In many respects, the one that generates the most visceral reaction from non-technical people is the phone demo, so let's get Tropo to assign a phone number to the application. This is done by clicking "Add a phone number" and selecting an area code from which the number will be generated. Naturally, U.S. toll-free (1-800) numbers are supported as well, but because of the costs involved, that requires setting up a billing plan. Once you've selected an area code, Tropo needs a few minutes to provision the number, and then you can dial the number and be greeted in fine synthesized voice fashion. (Yes, do it now.)

Hi, Honey! Do You Still Love Me?

But that's not nearly enough. Being as how the Valentine's season is approaching, and the holiday season is just past, and you probably played too much Xbox 360 over Christmas, and that got your significant other all annoyed with you for not paying attention to him/her (yes, my wife still brings this up over dinner), you might want to make sure your loved one (girlfriend/boyfriend/spouse/whatever) is still in love with you. So let's flip the code around for a bit and make sure. Editing the running application is pretty easy: going back to the application dashboard (which should still be up, assuming you've closed the text editor window; if you haven't, don't worry, just stay there), simply click on the "Hosted File" link again and choose the "Edit this hosted file" option to bring the text editor back up. This time, replace the say code with the following (with, of course, your loved one's name in place of my wife's):

```
say("I love you, Charlotte!");
var results = ask("Do you love me too? Yes or no?", {
  choices: "yes, no"
});
log("results.value: " + results.value)
if (results.value == "yes") {
  say("Yay! That makes me happy.");
}
else {
  say("Oh. Now I'm a sad panda.");
}
```



New Mapped File

File Name: HelloMSDNScript.js

File Text:

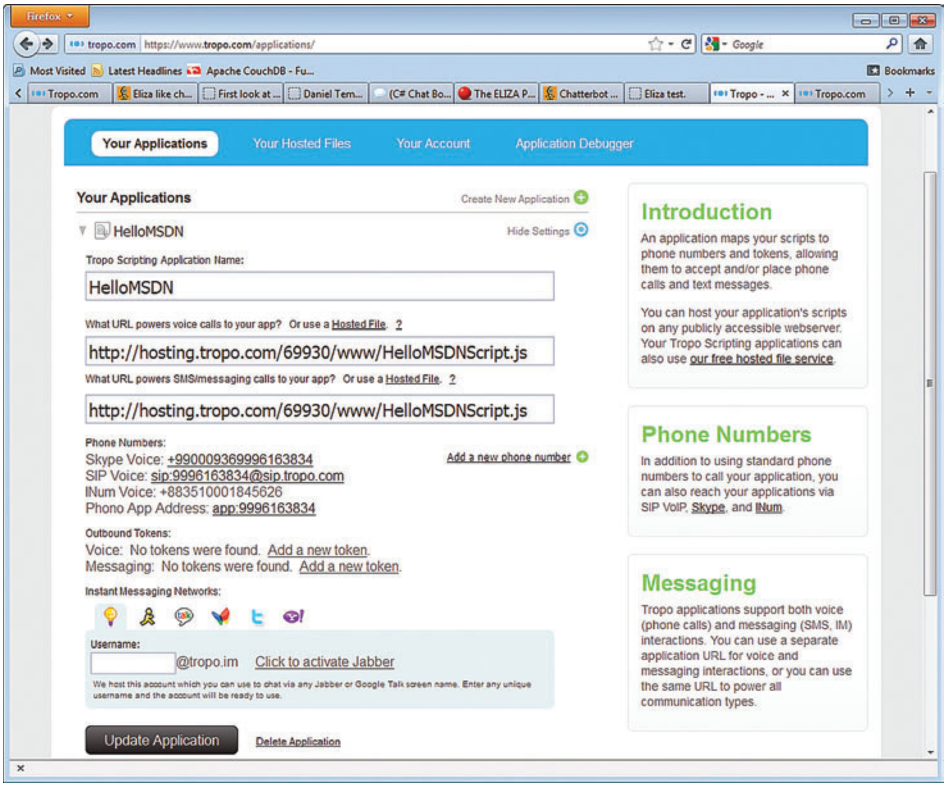
```
say("Greetings, MSDN fans!")
```

Create File   Cancel

Tropo applications support both voice (phone calls) and messaging (SMS, IM) interactions. You can use a separate application URL for voice and

Figure 2 **Creating a Script in Tropo**

The *ask* routine is a blocking call, playing the prompt, and then waiting (up to a configurable timeout number of seconds) for a voice response. This being a JavaScript API, we can pass in a number of optional parameters in the JSON struct at the end of the ask call, which in this case contains the "choices" string, which is a comma-delimited list of acceptable voice responses. Tropo contains some speech-to-text translators and will attempt to parse the spoken response from the caller—as best it can, anyway. (When Tropo does its parsing, it does so with a "confidence" factor, indicating how strongly it thinks it parsed correctly, and the level of confidence you demand in your spoken responses can be configured. By default it's .3, which is pretty loosey-goosey, but usually sufficient for spoken responses when the acceptable results are prompted, as in the previous "Yes or no" prompt.)



Your Applications

HelloMSDN

Tropo Scripting Application Name:

HelloMSDN

What URL powers voice calls to your app? Or use a Hosted File. ?

http://hosting.tropo.com/69930/www/HelloMSDNScript.js

What URL powers SMS/messaging calls to your app? Or use a Hosted File. ?

http://hosting.tropo.com/69930/www/HelloMSDNScript.js

Phone Numbers:
Skype Voice: +990009369996163834
SIP Voice: sip:9996163834@sip.tropo.com
INum Voice: +883510001845626
Phono App Address: app:9996163834

Outbound Tokens:
Voice: No tokens were found. Add a new token.
Messaging: No tokens were found. Add a new token.

Instant Messaging Networks:

Username:
@tropo.im Click to activate Jabber

We host this account which you can use to chat via any Jabber or Google Talk screen name. Enter any unique username and the account will be ready to use.

Introduction

An application maps your scripts to phone numbers and tokens, allowing them to accept and/or place phone calls and text messages.

You can host your application's scripts on any publicly accessible webserver. Your Tropo Scripting applications can also use our free hosted file service.

Phone Numbers

In addition to using standard phone numbers to call your application, you can also reach your applications via SIP VoIP, Skype, and INum.

Messaging

Tropo applications support both voice (phone calls) and messaging (SMS, IM) interactions. You can use a separate application URL for voice and messaging interactions, or you can use the same URL to power all communication types.

Figure 3 **The Tropo Application Dashboard**

But wait! The default voice is a female voice, and that could sound a little weird when sending it to my wife. So let's change the voice over to something more like my own. This is done by passing in the optional "voice" field in the JSON arguments for both say and ask:

```
say("I love you, Charlotte!", { voice:"victor"});
var results = ask("Do you love me too? Yes or no?", {
  voice: "victor",
  choices: "yes, no"
});
log("results.value: " + results.value)
if (results.value == "yes") {
  say("Yay! That makes me happy.", { voice:"victor"});
}
else {
  say("Oh. Now I'm a sad panda.", { voice:"victor"});
}
```

When Tropo does its parsing, it does so with a "confidence" factor, indicating how strongly it thinks it parsed correctly.

Once done editing, save the file; Tropo will update the file in place, and the next phone call made will play with the new voice. Note that "victor" is just one of a number of possible voices, including a variety of different accents. Make sure you don't pick one that sounds sexier than your own natural voice, though, or your chosen loved one may prefer the phone over you, and that would probably be bad.

Of course, it would be best if it were your own voice, and with a little preparation, you can make it so. Both say and ask support playing an MP3 or WAV file instead of doing the text-to-speech option currently being used, so grab your trusty computer microphone, record the prompts and the responses, and upload them to your favorite Web server. Then, instead of offering up the text to parse and synthesize, provide the URLs for the pre-recorded files (which you'll have to record and store on an HTTP-accessible server) to play; the code will read as such (where the URL is your recorded file):

```
say('http://www.tedneward.com/howdy.wav');
```

Next: Artificial Intelligence

I've only scratched the surface of what you can do with Tropo—in fact, I have much more to explore with Tropo before I'm done with the subject—but to fully understand where I want to go with this particular example, we'll have to take a side trip into the wonderful world of artificial intelligence, then revisit Tropo again. ■

TED NEWARD *is an architectural consultant with Neudesic LLC. He's written more than 100 articles, is a C# MVP and INETA speaker and has authored and coauthored a dozen books, including the recently released "Professional F# 2.0" (Wrox). He consults and mentors regularly. Reach him at ted@tedneward.com if you're interested in having him come work with your team, or read his blog at blogs.tedneward.com.*

THANKS to the following technical expert for reviewing this article: *Adam Kalsey*

Does your Team do more than just track bugs?

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

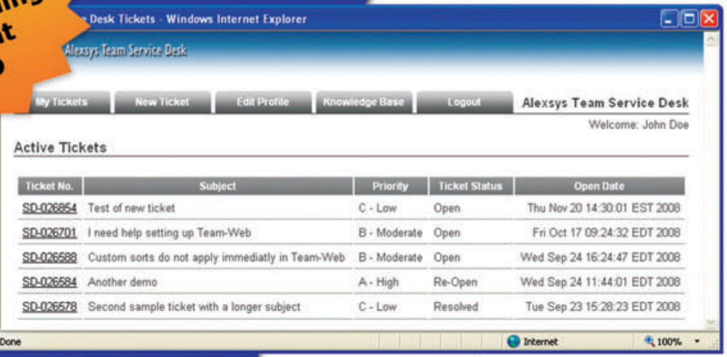
Alexsys Team® does! Alexsys Team 2 is a multi-user Team management system that provides a powerful yet easy way to manage all the members of your team and their tasks - including defect tracking. Use Team right out of the box or tailor it to your needs.

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

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

Native Smart Card Login Support including Government and DOD

Alexsys Team



Alexsys Team Service Desk

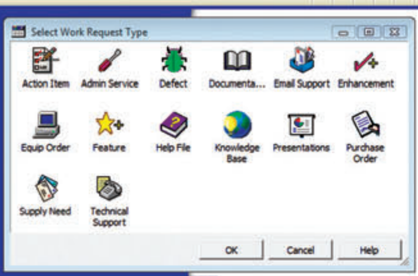
Welcome: John Doe

Active Tickets

Ticket No.	Subject	Priority	Ticket Status	Open Date
SD-026854	Test of new ticket	C - Low	Open	Thu Nov 20 14:30:01 EST 2008
SD-026701	I need help setting up Team-Web	B - Moderate	Open	Fri Oct 17 09:24:32 EDT 2008
SD-026588	Custom sorts do not apply immediatly in Team-Web	B - Moderate	Open	Wed Sep 24 16:24:47 EDT 2008
SD-026584	Another demo	A - High	Re-Open	Wed Sep 24 11:44:01 EDT 2008
SD-026578	Second sample ticket with a longer subject	C - Low	Resolved	Tue Sep 23 15:28:23 EDT 2008

New in Team 2.11

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



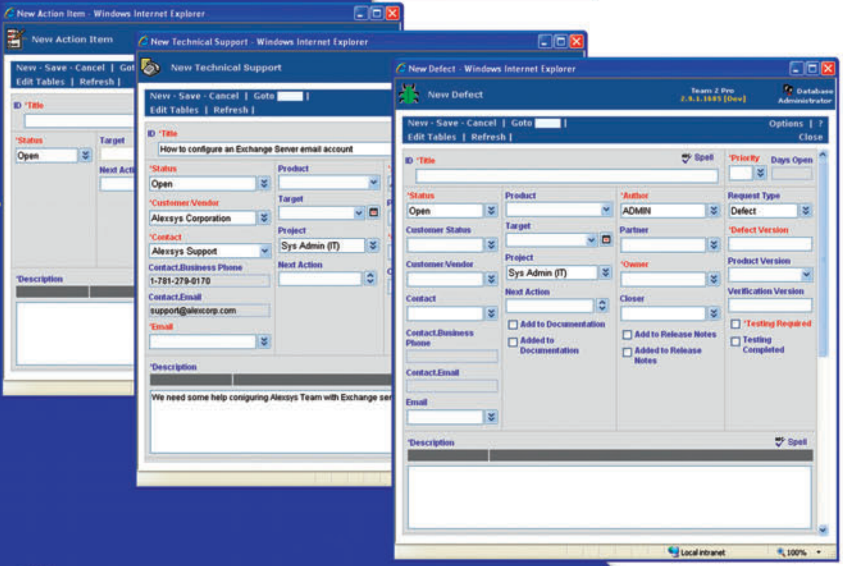
Service Desk Features

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

Team 2 Features

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

ZDNet Editors' Pick



New Action Item — *Status: Open

New Technical Support — ID *Title: How to configure an Exchange Server email account; *Status: Open; *Customer/Vendor: Alexsys Corporation; *Contact: Alexsys Support; Contact.Business Phone: 1-781-279-0170; Contact.Email: support@alexcorp.com; *Email; Project: Sys Admin (IT); *Description: We need some help coniguring Alexsys Team with Exchange ser

New Defect — ID *Title; *Priority; Days Open; *Status: Open; Customer Status; Customer/Vendor; Contact; Contact.Business Phone; Contact.Email; Email; Product; Target; Project: Sys Admin (IT); Next Action; □ Add to Documentation; □ Added to Documentation; *Author: ADMIN; Partner; *Owner; Closer; □ Add to Release Notes; □ Added to Release Notes; Request Type: Defect; *Defect Version; Product Version; Verification Version; □ *Testing Required; □ Testing Completed; *Description

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

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

CLIENT INSIGHT

JOHN PAPA

Getting Started with Knockout

Data binding is one of the most popular features in development today, and the Knockout JavaScript library brings those features to HTML and JavaScript development. The simplicity of the declarative binding syntax and seamless integration with separation patterns such as Model-View-ViewModel (MVVM) make common push-and-pull plumbing tasks much simpler while making code easier to maintain and enhance. In this inaugural Client Insight column I'll discuss the scenarios for which Knockout is ideal, explain how to get started with it and demonstrate how to use its fundamental features. The code examples, which you can download from code.msdn.microsoft.com/mag201202ClientInsight, demonstrate how to use declarative binding, create different types of binding objects and write data-centric JavaScript code that follows good separation patterns such as MVVM.

Getting Started

Knockout, developed by Steve Sanderson, is a small, open source JavaScript library with an MIT license. Knockoutjs.com maintains an updated list of the browsers that Knockout supports (currently it supports all major browsers including Internet Explorer 6+, Firefox 2+, Chrome, Opera and Safari). You need a few important resources to get started developing with Knockout. Start by getting the most recent version of Knockout (currently 2.0.0) from bit.ly/scmtAi and reference it in your project. If you're using Visual Studio 2010, however, I highly recommend that you install and use the NuGet Package Manager Visual Studio extension to download Knockout (and any other libraries you might use) because it will manage versions and alert you when a new one is available. NuGet will download Knockout and put two JavaScript files in the scripts folder of your project. The minified file is recommended for production and follows the naming convention knockout-x.y.z.js where x.y.z is the major, minor and revision number. There's also a knockout-x.y.x-debug.js file, which contains the Knockout source code in human-readable form. I recommend referencing this file when learning Knockout and when debugging.

Once you have the files, open your HTML page (or Razor file if you're using ASP.NET MVC), create a script and reference the knockout library:

```
<script src="../scripts/knockout-2.0.0.js" type="text/javascript"></script>
```

Code download available at code.msdn.microsoft.com/mag201202ClientInsight.

No Bindings

Knockout is best explained by first examining how you would write code to push data from a source object to HTML elements without using Knockout (the relevant source code can be found in sample page 01-without-knockout.html in the downloadable code sample). I'll then show how to accomplish the same thing using Knockout. I'll start with some HTML target elements and push some values from a source object into these HTML elements:

```
<h2>Without Knockout</h2>
<span>Item number:</span><span id="guitarItemNumber"></span>
<br/>
<span>Guitar model:</span><input id="guitarModel"/>
<span>Sales price:</span><input  id="guitarSalesPrice"/>
```

Knockout, developed by Steve Sanderson, is a small, open source JavaScript library with an MIT license.

If you have an object from which you want to push data into standard HTML elements, you could use jQuery:

```
$(document).ready(function () {
  var product = {
    itemNumber: "T314CE",
    model: "Taylor 314ce",
    salePrice: 1199.95
    };
    $("#guitarItemNumber").text(product.itemNumber);
    $("#guitarModel").val(product.model);
    $("#guitarSalesPrice").val(product.salePrice);
});
```

This code sample uses jQuery to locate the HTML elements with the corresponding IDs and sets their values to each appropriate object property.

There are three main points to notice in this code. First, the values are pushed from the source object into the HTML elements, thus requiring a line of code for each mapping from source value to target element. If there were many more properties (or if there were arrays and object graphs), the code could easily get unwieldy. Second, if the values in the source object change, the HTML elements won't reflect that change unless the code to push the values was called again. Third, if the values change in the HTML elements (the target), the changes won't be reflected in the underlying source

Figure 1 **With and Without Observables**

```
$(document).ready(function () {
  var data = {
    product1: {
      id: 1002,
      itemNumber: "T110",
      model: "Taylor 110",
      salePrice: 699.75
    },
    product2: {
      id: ko.observable(1001),
      itemNumber: ko.observable("T314CE"),
      model: ko.observable("Taylor 314ce"),
      salePrice: ko.observable(1199.95)
    }
  };

  ko.applyBindings(data);
});
```

object. Of course, all of this could be resolved with a lot of code, but I'll try to resolve it using data binding.

The same HTML could be rewritten using Knockout:

```
<h2>With Knockout</h2>
<span Item number</span><span data-bind="text: itemNumber"></span>
<br/>
<span>Guitar model:</span><input data-bind="value: model"/>
<span>Sales price:</span><input data-bind="value: salePrice"/>
```

Notice the id attributes have been replaced with data-bind attributes. Once you call the applyBindings function, Knockout binds the given object ("product" in this example) to the page. This sets the data context for the page to the product object, which means that the target elements can then identify the property of that data context to which they want to bind:

```
ko.applyBindings(product);
```

The values in the source object will be pushed to the target elements in this page. (All of Knockout's functions exist within its own namespace: *ko*.) The linkage between the target elements and the source object is defined by the data-bind attribute. In the preceding example, Knockout sees the data-bind attribute for the first span tag is identifying that its text value should be bound to the itemNumber property. Knockout then pushes the value for the product.itemNumber property to the element.

You can see how using Knockout could easily reduce code.

You can see how using Knockout could easily reduce code. As the number of properties and elements increase, the only JavaScript code required is the applyBindings function. However, this doesn't yet solve all of the problems. This example doesn't yet update the HTML targets when the source changes, nor does the source object update when the HTML targets change. For this, we need observables.

Observables

Knockout adds dependency tracking through observables, which are objects that can notify listeners when underlying values have changed (this is similar to the concept of the INotifyPropertyChanged interface in XAML technology). Knockout implements observable properties by wrapping object properties with a custom function named observable. For example, instead of setting a property in an object like so:

```
var product = {
  model: "Taylor 314ce"
}
```

you can define the property to be an observable property using Knockout:

```
var product = {
  model: ko.observable("Taylor 314ce")
}
```

Once the properties are defined as observables, the data binding really takes shape. The JavaScript code shown in **Figure 1** demonstrates two objects that Knockout binds to HTML elements. The first object (data.product1) defines its properties using a simple object literal while the second object (data.product2) defines the properties as Knockout observables.

The HTML for this sample, shown in **Figure 2**, shows four sets of element bindings. The first and second div tags contain HTML elements bound to non-observable properties. When the values in the first div are changed, notice that nothing else changes. The third and fourth div tags contain the HTML elements bound to observable properties. Notice that when the values are changed in the third div, the elements in the fourth div are updated. You can try this demo out using example 02-observable.html.

(Note: Knockout doesn't require you to use observable properties. If you want DOM elements to receive values once but then not be updated when the values in the source object change, simple objects will suffice. However, if you want your source object and target DOM elements to stay in sync—two-way binding—then you'll want to consider using observable properties.)

Built-in Bindings

The examples thus far have demonstrated how to bind to the Knockout's text and value bindings. Knockout has many built-in

Figure 2 **Binding to Observable and Non-Observables**

```
<div>
  <h2>Object Literal</h2>
  <span>Item number</span><span data-bind="text: product1.itemNumber"></span>
  <br/>
  <span>Guitar model:</span><input data-bind="value: product1.model"/>
  <span>Sales price:</span><input data-bind="value: product1.salePrice"/>
</div>
<div>
  <h2>Underlying Source Object for Object Literal</h2>
  <span>Item number</span><span data-bind="text: product1.itemNumber"></span>
  <br/>
  <span>Guitar model:</span><span data-bind="text: product1.model"></span>
  <span>Sales price:</span><span data-bind="text: product1.salePrice"></span>
</div>
<div>
  <h2>Observables</h2>
    <span>Item number</span><span data-bind="text: product2.itemNumber"></span>
  <br/>
    <span>Guitar model:</span><input data-bind="value: product2.model"/>
    <span>Sales price:</span><input data-bind="value: product2.salePrice"/>
</div>
<div>
  <h2>Underlying Source Object for Observable Object</h2>
  <span>Item number</span><span data-bind="text: product2.itemNumber"></span>
  <br/>
    <span>Guitar model:</span><span data-bind="text: product2.model"></span>
    <span>Sales price:</span><span data-bind="text: product2.salePrice"></span>
</div>
```

bindings that make it easier to bind object properties to target DOM elements. For example, when Knockout sees a text binding, it will set the innerText property (using Internet Explorer) or the equivalent property in other browsers. When the text binding is used, any previous text will be overwritten. While there are many built-in bindings, some of the most common for displaying can be found in **Figure 3**. The Knockout documentation online contains a complete list in the left navigation pane (bit.ly/ajRyPj).

ObservableArrays

Now that the previous examples got your feet wet with Knockout, it's time to move on to a more practical yet still fundamental example with hierarchical data. Knockout supports many types of bindings, including binding to simple properties (as seen in the previous examples), binding to JavaScript arrays, computed bindings and custom bindings (which I'll look at in a future article on Knockout). The next example demonstrates how to bind an array of product objects using Knockout to a list (shown in **Figure 4**).

When dealing with object graphs and data binding, it's helpful to encapsulate all of the data and functions the page requires into a single object. This is often referred to as a ViewModel from the MVVM pattern. In this example the View is the HTML page and its DOM elements. The Model is the array of products. The ViewModel glues the Model to the View; the glue it uses is Knockout.

The array of products is set up using the observableArray function. This is similar to the ObservableCollection in XAML technologies. Because the products property is an observableArray, every time an item is added to or removed from the array, the target elements will be notified and the item will be added or removed from the DOM, as shown here:

```
var showroomViewModel = {
  products: ko.observableArray()
};
```

The showroomViewModel is the root object that will be data bound to the target elements. It contains a list of products, which come in from a data service as JSON. The function that loads the product list is the showroomViewModel.load function, which appears in **Figure 5** along with the rest of the JavaScript that sets up the showroomViewModel object (you'll find the complete source and sample data for this example in 03-observableArrays.html). The load function loops through the sample product data and uses the Product function to create the new product objects before pushing them into the observableArray.

Although a product's properties are all defined using observables, they don't necessarily need to be observable. For example, these could be plain properties if they're read-only, and if—when the source changes—either the target isn't expected to be updated or the entire container is expected to be refreshed. However, if the properties need to be refreshed when the source changes or is edited in the DOM, then observable is the right choice.

When dealing with object graphs and data binding, it's helpful to encapsulate all of the data and functions required by the page into a single object.

The Product function defines all of its properties as Knockout observables (except photoUrl). When Knockout binds the data, the products array property can be accessed, making it easy to use standard functions such as length to show how many items are currently data bound:

```
<span data-bind="text: products().length"></span>
```

Control-of-Flow Bindings

The array of products can then be data bound to a DOM element, where it can be used as an anonymous template for displaying the list. The following HTML shows that the ul tag uses the foreach control-of-flow binding to bind to the products:

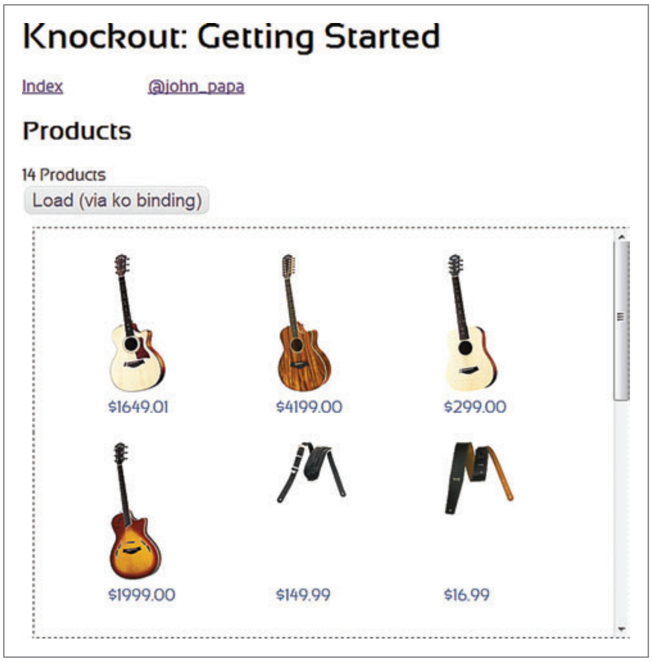
```
<ul data-bind="foreach:products">
  <li class="guitarListCompact">
    <div class="photoContainer">
      <img data-bind="visible: photoUrl, attr: { src: photoUrl }"
        class="photoThumbnail"></img>
    </div>
    <div data-bind="text: salePrice"></div>
  </li>
</ul>
```

The elements inside the ul tag will be used to template each product. Inside the foreach, the data context also changes from the root object showroomViewModel to each individual product. This is why the inner DOM elements can bind to the photoUrl and salePrice properties of a product directly.

There are four main control-of-flow bindings: *foreach*, *if*, *ifnot* and *with*. These control bindings allow you to declaratively define the control of flow logic without creating a named template.

Figure 3 **Common Knockout Bindings**

Example	Scenario
text: model	Binds the property (model) to the text value for the target element. Often used for read-only elements such as spans.
visible: isInStock	Binds the value property (isInStock) to the visibility of the target element. The property value will evaluate to true or false.
value: price	Binds the value of the property (price) to the target element. Often used with input, select and textarea elements.
css: className	Binds the value of the property (className) to the target element. Often used to set or toggle css class names for DOM elements.
checked: isInCart	Binds the value of the property (isInCart) to the target element. Used for checkbox elements.
click: saveData	Adds an event handler for the bound JavaScript function (saveData) when the user clicks the DOM element. Works on any DOM element but is often used for button, input and *a* elements.
attr: {src: photoUrl, alt: name}	Binds any specified attribute for the DOM element to the source object. Often used when the other built-in bindings don't cover the scenario, such as with the src attribute of an img tag.


Knockout: Getting Started

Index @john_papa

Products

14 Products

Load (via ko binding)

$1649.01 $4199.00 $299.00

$1999.00 $149.99 $16.99

Figure 4 **Binding to an Observable Array**

When the *if* control-of-flow binding is followed by a condition that is truthy or the *ifnot* binding is followed by a condition that evaluates to false, the contents inside of its block are bound and displayed, as shown here:

```
<div data-bind="if:onSale">
  <span data-bind="text: salePrice"></span>
</div>
```

The *with* binding changes the data context to whatever object you specify. This is especially useful when diving into object graphs with multiple parent/child relationships or distinct ViewModels within a page. For example, if there's a sale ViewModel object that's bound to the page and it has child objects customer and salesPerson, the *with* binding could be used to make the bindings more readable and easier to maintain, as shown here:

```
<div data-bind="with:customer">
  <span data-bind="text: name"></span><br/>
  <span data-bind="text: orderTotal"></span>
</div>
<div data-bind="with:salesPerson">
  <span data-bind="text: employeeNum"></span><br/>
  <span data-bind="text: name"></span>
</div>
```

Figure 5 **Defining the Data to Bind**

```
var photoPath = "/images/";
function Product () {
  this.id = ko.observable();
  this.salePrice = ko.observable();
  this.listPrice = ko.observable();
  this.rating = ko.observable();
  this.photo = ko.observable();
  this.itemNumber = ko.observable();
  this.description = ko.observable();
  this.photoUrl = ko.computed(function () {
    return photoPath + this.photo();
  }, this);
};
var showroomViewModel = {
  products: ko.observableArray()
};
showroomViewModel.load = function () {
  this.products([]); // reset
  $.each(data.Products, function (i, p) {
    showroomViewModel.products.push(new Product()
      .id(p.Id)
      .salePrice(p.SalePrice)
      .listPrice(p.ListPrice)
      .rating(p.Rating)
      .photo(p.Photo)
      .itemNumber(p.ItemNumber)
      .description(p.Description)
      );
  });
};
ko.applyBindings(showroomViewModel);
```

Computed Observables

You might have noticed that the Product function defined the photoUrl as a special type of computed property. ko.computed defines a binding function that evaluates the value for the databinding operation. The computed property automatically updates when any of the observables it depends on for its evaluation change. This is especially useful when the value isn't clearly represented in a concrete value in the source object. Another common example besides creating a URL is creating a fullName property out of firstName and lastName properties.

(Note: Previous versions of Knockout referred to computed properties as dependentObservable. Both still work in Knockout 2.0.0, but I recommend using the newer computed function.)

A computed property accepts a function for evaluating the value and the object that will represent the object to which you're attaching the binding. The second parameter is important because JavaScript object literals don't have a way of referring to themselves. In **Figure 5** the *this* keyword (which represents the showroomViewModel) is passed in so the dependent observable's function can use it to get the photo property. Without this being passed in, the photo function would be undefined and the evaluation would fail to produce the expected URL:

```
this.photoUrl = ko.computed(function () {
  return photoPath +  photo();  // photo() will be undefined
});
```

Understanding Fundamental Binding Properties

This column got you started with data binding using the Knockout JavaScript library. The most important aspect of Knockout is to understand the fundamental binding properties: observable, observableArray and computed. With these observables you can create robust HTML apps using solid separation patterns. I also covered the more common built-in bindings types and demonstrated the control-of-flow bindings. However, Knockout supports much more functionality. Next time, I'll explore built-in bindings in more detail. ■

John Papa *is a former evangelist for Microsoft on the Silverlight and Windows 8 teams, where he hosted the popular Silverlight TV show. He has presented globally at keynotes and sessions for the BUILD, MIX, PDC, Tech•Ed, Visual Studio Live! and DevConnections events. Papa is also a columnist for* Visual Studio Magazine *(Papa's Perspective) and author of training videos with Pluralsight. Follow him on Twitter at twitter.com/john_papa.*

Thanks to the following technical expert for reviewing this article: *Steve Sanderson*

TOUCH AND GO CHARLES PETZOLD

Background Audio on Windows Phone 7.5

Back in the old days of MS-DOS, programmers could implement a crude form of task-switching with a technique called Terminate and Stay Resident (also known as TSR). TSR programs installed hooks into keyboard interrupts or other OS mechanisms and then terminated, leaving the program in memory ready to kick into action when the user pressed a particular key combination or something else of interest happened.

MS-DOS wasn't equipped to handle even this level of rudimentary task-switching, so these TSRs created some serious problems. They would conflict with each other and frequently crash the whole OS. This was one of the primary reasons some of us welcomed Windows to supplement and later replace MS-DOS.

I mention this ancient history because I'm going to show you how to play music files in the background from a Windows Phone 7.5 application, and I know that developers have a tendency to think outside the box. Generally this is a good thing, but you shouldn't use this technique for any purpose other than playing music files. Doing so might cause your application to be rejected by the Windows Phone Marketplace.

The technique I'll show you is only for playing sound or music files from a Web location or isolated storage. If your application needs to play songs from the phone's normal music library, you can do that using the MediaLibrary and MediaPlayer classes I discussed in the previous issue (msdn.microsoft.com/magazine/hh708760).

Application Plus DLL

In Windows Phone 7.5, an object that runs in the background to assist an application is known as an *agent*. To play music files in the background, you use a class that derives from AudioPlayerAgent. This class must be in a dynamic link library that's referenced by the program. Program code does not itself run in the background; what runs in the background is this class that derives from AudioPlayerAgent.

Included with the downloadable code for this article is a Visual Studio solution named SimpleBackgroundAudio. For purposes of clarity, this program contains just about the minimum amount of code necessary to get background audio to work. I created the solution from the New Project dialog box by specifying Windows Phone Application, and then Windows Phone 7.1. (The 7.1 designation is used internally within the Windows Phone OS and Windows Phone applications, but it means the same thing as the more common 7.5 designation.)

Code download available at code.msdn.microsoft.com/mag201202TouchNGo.

To the MainPage class of the SimpleBackgroundAudio project I added a Button and a Click handler for that button:

```
void OnPlayButtonClick(object sender, RoutedEventArgs args)
{
  AudioTrack audioTrack =
    new AudioTrack(new Uri("http://www.archive.org/.../Iv.Presto.mp3"),
                   "Symphony No. 9: 4th Movement",
                   "Felix Weingartner",
                   "Beethoven: Symphony No. 9",
                   null,
                   null,
                   EnabledPlayerControls.Pause);
  BackgroundAudioPlayer.Instance.Track = audioTrack;
}
```

The AudioTrack and BackgroundAudioPlayer classes are in the Microsoft.Phone.BackgroundAudio namespace. The URL (which I've shortened here to accommodate the magazine's column width) references an MP3 file on the Internet Archive (archive.org) containing the last movement of Beethoven's Symphony No. 9 as conducted by Felix Weingartner in a 1935 recording. (You can alternatively specify a URL that addresses a file in isolated storage; use the UriKind.Relative argument to the Uri constructor if that's the case.) The next three arguments of the AudioTrack constructor provide album, artist and track information.

The BackgroundAudioPlayer class is somewhat similar to the MediaElement or MediaPlayer classes that play audio files in the foreground. BackgroundAudioPlayer has no constructor; instead you obtain the only instance with the static Instance property. In this code, the Track property is set to the AudioTrack object just created.

You can compile and run this program, but it won't do anything. To make BackgroundAudioPlayer sing, you need a DLL containing a class that derives from AudioPlayerAgent. Visual Studio can create this class and library project for you. For my program, I right-clicked the SimpleBackgroundAudio solution name in Visual Studio, selected Add New Project from the menu and then chose Windows Phone Audio Playback Agent from the template list. I named this new project SimpleAudioPlaybackAgent.

Visual Studio creates a library project with a class named AudioPlayer that derives from AudioPlayerAgent, initialized with several skeleton methods. This is the class that runs in the background.

Very important: Create a reference from the application to this DLL! To do this, I right-clicked the References section under the SimpleBackgroundAudio project, selected Add Reference, and then in the dialog box I selected the Projects tab and then the SimpleAudioPlaybackAgent project.

In the AudioPlayerAgent derivative, you'll want to modify at least two methods: OnPlayStateChanged and OnUserAction. The complete AudioPlayer class from this project (minus using directives and comments) is shown in **Figure 1**.

Figure 1 **The AudioPlayer Class in SimpleBackgroundAudio**

```
namespace SimpleAudioPlaybackAgent
{
  public class AudioPlayer : AudioPlayerAgent
  {
    protected override void OnPlayStateChanged(BackgroundAudioPlayer player,
                                  AudioTrack track, PlayState playState)
    {
      switch (playState)
      {
        case PlayState.TrackReady:
          player.Play();
          break;

        case PlayState.TrackEnded:
          player.Track = null;
          break;
      }
      NotifyComplete();
    }
    protected override void OnUserAction(BackgroundAudioPlayer player,
                                  AudioTrack track, UserAction action,
                                  object param)
    {
      switch (action)
      {
        case UserAction.Pause:
          player.Pause();
          break;

        case UserAction.Play:
          player.Play();
          break;
      }
      NotifyComplete();
    }

    protected override void OnError(BackgroundAudioPlayer player,
                                  AudioTrack track, Exception error,
                                  bool isFatal)
    {
        base.OnError(player, track, error, isFatal);
        NotifyComplete();
    }
    protected override void OnCancel()
    {
        base.OnCancel();
        NotifyComplete();
    }
  }
}
```

Look at the OnPlayStateChanged override first. This method is called when the PlayState property of the BackgroundAudioPlayer changes. The first argument is the same BackgroundAudioPlayer referenced in the program code; the last argument is a member of the PlayState enumeration.

When the program sets the Track property of the BackgroundAudioPlayer in the Click event handler, the BackgroundAudioPlayer accesses the music file over the Internet; the SimpleAudioPlaybackAgent DLL is loaded and the OnPlayStateChanged method eventually gets a call with the playState argument set to PlayState.TrackReady. It is the responsibility of OnPlayStateChanged to call the Play method of the BackgroundAudioPlayer object to start playing that track.

If the phone's regular music player happens to be playing something at the time, it will be stopped. At this point, you can navigate away from the program or even terminate it by pressing the phone's Back button, and the music will continue playing.

You can run this program in the Windows Phone emulator, but you'll want to try it on an actual phone so you can press the phone's volume control. This invokes a little drop-down known as the Universal Volume Control (UVC), which is an important part of background audio. The UVC displays the name of the track being played and the artist, based on the arguments that the application supplied to the AudioTrack constructor. The UVC also displays buttons for Previous Track, Pause and Next Track. In this program, only the Pause button is enabled because that's what I specified in the last argument to the AudioTrack constructor. When you press the Pause button, it toggles between Pause and Play.

The AudioPlayer class handles these Pause and Play commands in the OnUserAction override shown in **Figure 1**. The UserAction argument indicates the particular button pressed by the user. OnUserAction responds by calling the appropriate method in the BackgroundAudioPlayer object.

When the track finishes playing, OnPlayStateChanged gets a call with PlayState.TrackEnded. The method responds by setting the Track property of the BackgroundAudioPlayer to null, which removes the item from the UVC. If you want, you can go back into the application and start the music playing again.

Notice that both OnPlayStateChanged and OnUserAction conclude with calls to NotifyComplete: This is required. Also notice that neither method includes a call to the base class method. These base class calls are part of the template that Visual Studio creates for you, but I had a problem with the OnUserAction override when the base method was called. The background audio sample code from Microsoft (available online from the Windows Phone 7.5 documentation area) also doesn't include calls to the base methods.

A Very Strange DLL

When experimenting with background audio, you'll want to keep a watch on the Output window in Visual Studio. When you run SimpleBackgroundAudio from the debugger, the Output window lists all the system libraries that are loaded on the phone to run the program, as well as the program itself, which is SimpleBackgroundAudio.dll. The lines in the Output window that list these libraries begin with the words "UI Task," indicating the program.

When you tap the button in SimpleBackgroundAudio, you'll see many of the same DLLs being loaded—but now with each line prefaced by the words "Background Task." These DLLs include the application DLL as well as SimpleAudioPlaybackAgent.dll. The loading of these DLLs is one reason it takes a little bit of time for the music to begin playing after you tap the button.

As you experiment with programs that play background audio, you'll probably want to insert Debug.WriteLine statements in all method overrides in the AudioPlayer class, and then study their real-time behavior in the Output window.

You also might want to create a constructor for the AudioPlayer class with another Debug.WriteLine statement. You'll discover that a new instance of AudioPlayer is instantiated whenever a call needs to be made to OnPlayStateChanged or OnUserAction. Every call gets a new instance!

This simple fact has a profound implication: If you need this AudioPlayer class to maintain information between calls to OnPlayStateChanged and OnUserAction, you must keep that information in static fields or properties.

What if you need to pass information from a class in the SimpleBackgroundAudio program to the AudioPlayer class in the Simple-

Figure 2 **The AudioPlayer Overrides in PlaylistPlayer**

```
static List<AudioTrack> playlist = new List<AudioTrack>();
static int currentTrack = 0;

...

protected override void OnPlayStateChanged(BackgroundAudioPlayer player,
  AudioTrack track, PlayState playState)
{
  switch (playState)
  {
    case PlayState.TrackReady:
      player.Play();
      break;

    case PlayState.TrackEnded:
      if (currentTrack < playlist.Count - 1)
      {
        currentTrack += 1;
        player.Track = playlist[currentTrack];
      }
      else
      {
        player.Track = null;
      }
      break;
  }
  NotifyComplete();
}

protected override void OnUserAction(BackgroundAudioPlayer player,
  AudioTrack track, UserAction action, object param)
{
  switch (action)
  {
    case UserAction.Play:
      if (player.Track == null)
      {
        currentTrack = 0;
        player.Track = playlist[currentTrack];
      }
      else
      {
        player.Play();
      }
      break;

    case UserAction.Pause:
      player.Pause();
      break;

    case UserAction.SkipNext:
      if (currentTrack < playlist.Count - 1)
      {
        currentTrack += 1;
        player.Track = playlist[currentTrack];
      }
      else
      {
        player.Track = null;
      }
      break;

    case UserAction.SkipPrevious:
      if (currentTrack > 0)
      {
        currentTrack -= 1;
        player.Track = playlist[currentTrack];
      }
      else
      {
        player.Track = null;
      }
      break;

    case UserAction.Seek:
      player.Position = (TimeSpan)param;
      break;
  }
  NotifyComplete();
}
```

AudioPlaybackAgent library? It seems reasonable to define a public static method in the AudioPlayer class, and then call that method from MainPage or another class in the program. You can indeed do this, but it won't do what you want: Anything saved from this method will not be available during calls to OnPlayStateChanged and OnUserAction.

Why won't it be available? Recall the UI Task and Background Task designations in the Visual Studio Output window. These are two separate processes. The instance of the DLL referenced by your program is not the same as the instance that runs in the background, and therefore not even static data in a class in this DLL will be shared between the UI Task and the Background Task.

When testing a background audio application from the debugger in Visual Studio, you'll experience some additional oddities. When you exit a program that has initiated some background audio, the audio keeps playing and Visual Studio indicates that code is still running. To resume editing your program, you'll want to stop debugging directly from Visual Studio, and even then the music will often keep on playing. During development of a background audio program, you'll probably find yourself frequently uninstalling the program from the phone.

Enhancing the Application

The SimpleBackgroundAudio program has a big problem. Although it has a button to start the music playing, it has no way to pause it or shut it off. It doesn't even know when the music concludes or if anything else is happening. Yes, a user can always invoke the UVC to control background audio, but any program that starts music playing should also include its own controls to shut it off.

These enhancements are included in the project named PlaylistPlayer. As the name implies, this program plays a series of consecutive tracks—in this case, the 12 little piano pieces from Claude Debussy's Preludes, Book 1, as performed by Alfred Cortot in a 1949 recording available from the Internet Archive.

I originally wanted to create all the AudioTrack objects within the program and then hand them off to the background audio DLL. That seemed the more generalized program structure, but I discovered it didn't work because the application is accessing a different instance of the DLL than the one running in the background. Instead, I wrote the AudioPlayer class to create all the AudioTrack objects itself and store them in a generic List called playlist.

To make the program somewhat more challenging, I decided that the playlist wouldn't be circular: I didn't want to skip from the last track to the first track, or from the first to the last. For that reason, the first of the AudioTrack constructors has a last argument that combines the EnabledPlayerControls.Pause and EnabledPlayerControls.SkipNext flags; the last AudioTrack combines the Pause and SkipPrevious flags. All the others have all three flags. This is how the three buttons are enabled in the UVC for the various tracks.

Figure 2 shows the OnPlayStateChanged and OnUserAction overrides. In OnPlayStateChanged, when one track ends, the next track is set to the Track property of the BackgroundAudioPlayer. The OnUserAction override handles the Previous Track and Next Track commands from the UVC by setting the Track property to the previous or next AudioTrack in the playlist.

The program's MainPage class has a series of four standard application bar buttons to perform the same functions as the UVC. It also sets an event handler for the PlayStateChanged event of BackgroundAudioPlayer to update the screen with current track information, and a CompositionTarget.Rendering handler to update a Slider with current track progress, as shown in **Figure 3**.

The logic to enable and disable the application bar buttons is fairly simple: the Previous Track and Next Track buttons are enabled based on the PlayerControls property of the current AudioTrack; thus they should be consistent with the UVC. The Pause button is enabled if the player is playing; the Play button is enabled if the player is paused. If the current track is null, Play is enabled and all the other buttons are disabled.

The Click handlers of the four application bar buttons make calls to the BackgroundAudioPlayer methods SkipPrevious, Play, Pause and SkipNext, respectively. It's important to understand that these method calls don't directly control the operation of music playback. Instead, these method calls from the program trigger OnUserAction calls in the AudioPlayer class, and it's the code in AudioPlayer that actually starts and stops the music.

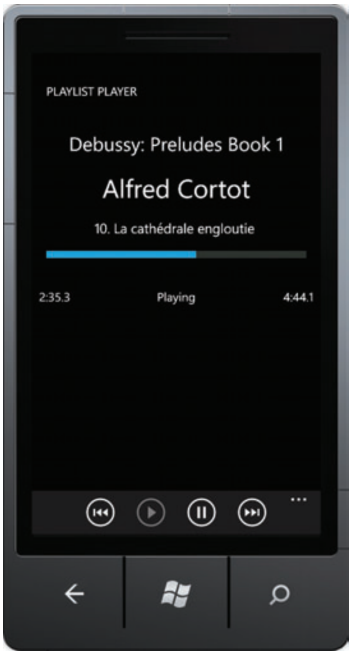


Figure 3 **The PlaylistPlayer Program**

This is somewhat peculiar, because it means that calls to the Play and Pause methods of BackgroundAudioPlayer behave differently depending whether they're called from a program or from the OnUserAction override.

I also added a ValueChanged handler for the Slider to move to a particular location in the track. The handler calculates a new position for the track and sets an appropriate TimeSpan object to the Position property of the BackgroundAudioPlayer. Similar to the case with Play and Pause calls, setting this property does not change the position of the track. Instead, it generates a call to the OnUserAction override in AudioPlayer with an action argument of UserAction.Seek and the TimeSpan encoded in the param argument. The OnUserAction override then sets this TimeSpan object to the Position property of the BackgroundAudioPlayer, and that's what actually causes the jump.

In practice, this Slider works fine when you just tap it to move ahead or back in the track by 10 percent. If you try to slide it, multiple Seek calls seem to build up, and the result is an audible mess. I would much prefer to use a regular ScrollBar rather than a Slider because then I could wait for an EndScroll event, which occurs when the user stops manipulating the control. Unfortunately, I've never been able to persuade the Windows Phone ScrollBar to work at all.

The Limitations

It's been interesting to see how Windows Phone 7.5 has given programmers more direct access to the phone's hardware (such as the video feed) as well as the ability to perform some background processing. But I can't help thinking there's a piece missing from this implementation of background audio.

Suppose a program wants to let the user pick from a list of music files to play in sequence. The program can't hand off the entire list to the DLL, so it needs to take responsibility to setting the Track property of the BackgroundAudioPlayer when each track ends. But this can happen only when the program is running in the foreground.

It's possible to pass some information between the application and the background DLL through the string Tag property of the AudioTrack class. But don't bother deriving a new class from AudioTrack in hopes of passing additional information to the DLL: A copy is made of the AudioTrack object created by the application for passing to the DLL overrides.

Fortunately, the DLL has access to the application's area of isolated storage, so the program can save a small file describing the playlist and then the DLL can access the playlist from the file. The bonus program for this column is a project named PlaylistFilePlayer, which demonstrates a simple approach to this technique. ■

CHARLES PETZOLD *is a longtime contributing editor to MSDN Magazine. His recent book, "Programming Windows Phone 7" (Microsoft Press, 2010), is available as a free download at bit.ly/cpebookpdf.*

THANKS to the following technical expert for reviewing this article: *Mark Hopkins*

Don't Get Me Started

DAVID PLATT

A Ring Around My Neck

This column marks the start of my third year ranting in this space as *MSDN Magazine*'s designated curmudgeon. On reflection, I'd call this the biennium of the smartphone, which has crossed over from technophile early adopters to the middle class mainstream. I drank the Kool-Aid and got mine last summer, the day before Verizon shut off its unlimited data plan. Now, like any proud geek, I whip out my phone and demonstrate my cool apps at the slightest excuse, annoying the heck out of everyone else. ("Have you seen this flashlight app? I use it at night, when I'm fumbling for my keys and …")

Whenever you buy an expensive toy, you start noticing people who have the same thing. Take a look at the picture (right) I took last summer at a commuter rail station near Boston. Every one of the waiting passengers has his or her nose stuck into a palmtop device. I did too, until I noticed the others around me, and surfaced long enough to snap the picture. I wonder how long it'll be before someone drifts onto the tracks and gets crushed like a bug.

We laugh at the Microsoft commercial of the guy checking his phone at the urinal (bit.ly/ufwwyW), but I see that every day. And I have seen them get dropped in, more than once. Really.

I wish I had Superman's X-ray vision to spy on my fellow travelers and their gadgets. How many are doing mundane tasks such as paying bills, how many are joining brain power with others into a superhuman Overmind, and how many are just looking at dirty pictures?

What is wrong with us? Can we not be alone with our thoughts for the five minutes it takes the train to arrive? Can we not contemplate the summer leaves, or the concept of railroads, or our families, or tonight's dinner? Do we have to stream action videos every instant? Have we irretrievably devolved from admiring a fellow commuter's sleek figure and wondering how much she paid for her clothes, to admiring her sleek Droid Razr and wondering how much she pays for her data plan?

"Daddy, I hope you're not going to become one of those boring geeks who always has his nose in his phone," said my daughter, 11, who desperately wants one of her own.

I try, but I'm astounded how difficult it is. Like Frodo in Tolkien's "Lord of the Rings," carrying the One Ring of Power around his neck, I find my hand creeping toward my smartphone, and I have to exert a conscious effort to refrain from pulling it out at the Thanksgiving dinner table. Boston's transportation authority had to ban even the *possession* of a cell phone while on duty, as employees seem unable to refrain from using them and causing crashes (bit.ly/u4D8Yg). Do we need Gollum to save us from ourselves by biting these things out of our hands?

David Platt looked up from his smartphone during his commute one morning and took this snapshot.

My phone does the opposite of the Ring—instead of making me invisible to others, it makes them invisible to me. This self-encapsulation started with the Sony Walkman 30 years ago, when we started playing music in headphones to isolate ourselves from our surroundings.

We retreat from physical society and join a virtual one. Former *Economist* editor Frances Cairncross called this process "The Death of Distance" in her book of that title. But as technology closes the distance between people around the world, it simultaneously creates distance between people in close proximity.

Our smartphones cast a strange field that attracts our hands strongly within a range of about a meter, like the strong nuclear force. It then repels other people out to a range of about 5 meters, like similar electrical charges. The field then attracts people with similar interests, out to infinity like gravity, but doesn't vary with distance. I've discovered the fifth fundamental force of physics. My Nobel Prize awaits.

Our thoughts bounce around the world but bypass the people sitting with us on the couch or standing next to us on the train platform, the ones we're sharing food with or competing for food against.

This much I know: When my daughter sends me a text message at dinner asking me to pass the salt, things have gone too far. ■

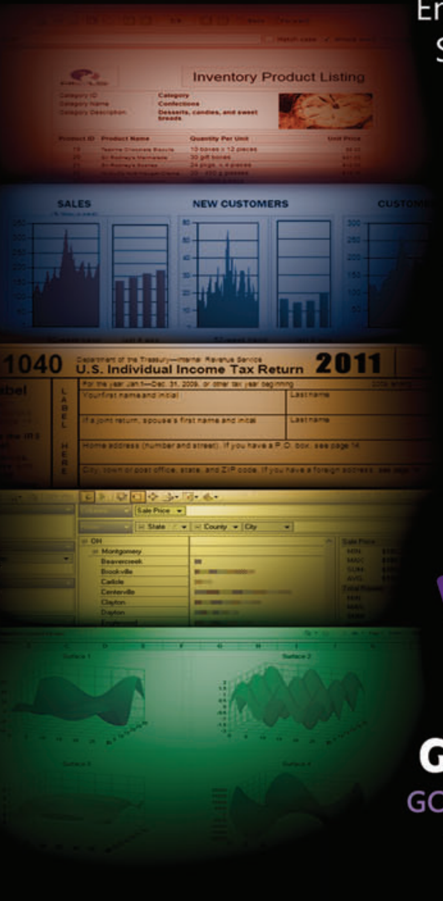
David S. *Platt teaches Programming .NET at Harvard University Extension School and at companies all over the world. He's the author of 11 programming books, including "Why Software Sucks" (Addison-Wesley Professional, 2006) and "Introducing Microsoft .NET" (Microsoft Press, 2002). Microsoft named him 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.*

Presenting the best Suite created since Mozart...

Dear GrapeCity,

Thanks for finally creating the ***PowerSuite.*** *I, and most developers, evaluate suites by three important categories: reporting, grids and charts. You have now packaged all of the amazing components I love together and I would like to thank you on behalf of .NET devlopers everywhere for giving us a great set of components, all in one valuable package!*

Carl Franklin
Host, .Net Rocks!



Enterprise-Ready Reporting, Spreadsheets, Grids, Data Analysis, and Charting Components for .NET Applications

AVAILABLE TOGETHER IN ONE AFFORDABLE PACKAGE FOR THE FIRST TIME EVER!

POWERSUITE
GrapeCity PowerTools
GCPowerTools.com/PowerSuite
GVTV.GCPowerTools.com