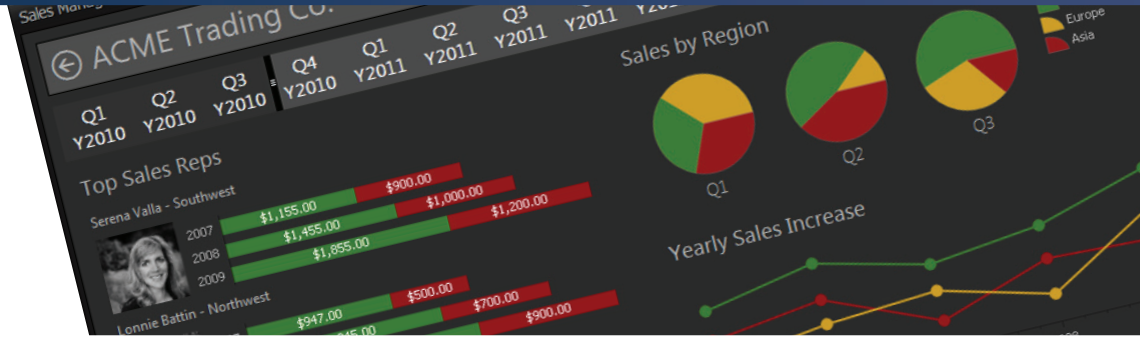
Microsoft
GOLD CERTIFIED
Partner



DXv2 is here

Simply Stunning.

Today's users expect beautiful apps in every part of their lives, from work to home. Now, with DevExpress tools, you can apply sophisticated themes and incorporate elegant Office-inspired controls into your designs. DXv2 delivers the tools you need to inspire and be inspired.

Download the 30-day free trial to experience the next generation of developer productivity tools at **www.DevExpress.com**

DevExpress

THE MICROSOFT JOURNAL FOR DEVELOPERS

MARCH 2012 VOL 27 NO 3

msdn magazine

Building the Internet of Things
Torsten Grabs and Colin Miller 30

Develop Hybrid Native and Mobile Web Apps
Shane Church 40

Create a Continuous Client Using Portable Class Libraries
David Kean 48

New Concurrency Features in Visual C++ 11
Diego Dagum 56

Windows Phone Data Binding
Jesse Liberty 62

Adding HTML5 Drag and Drop to SharePoint Lists
Andrey Markeev 68

COLUMNS

THE CUTTING EDGE
Build a Progress Bar with SignalR
Dino Esposito, page 6

DATA POINTS
Entity Framework Code First and DbContext FAQs
Julie Lerman, page 14

FORECAST: CLOUDY
Exploring Cloud Architecture
Joseph Fultz, page 18

THE WORKING PROGRAMMER
Talk to Me, Part 2: ELIZA
Ted Neward, page 74

CLIENT INSIGHT
Knockout's Built-in Bindings for HTML and JavaScript
John Papa, page 78

TOUCH AND GO
Streaming Audio in Windows Phone
Charles Petzold, page 84

DON'T GET ME STARTED
Touch, Not the Mouse
David Platt, page 88

Microsoft

Write Once, Experience Many

check out infragistics.com/jquery

NetAdvantage® for jQuery

TREE

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

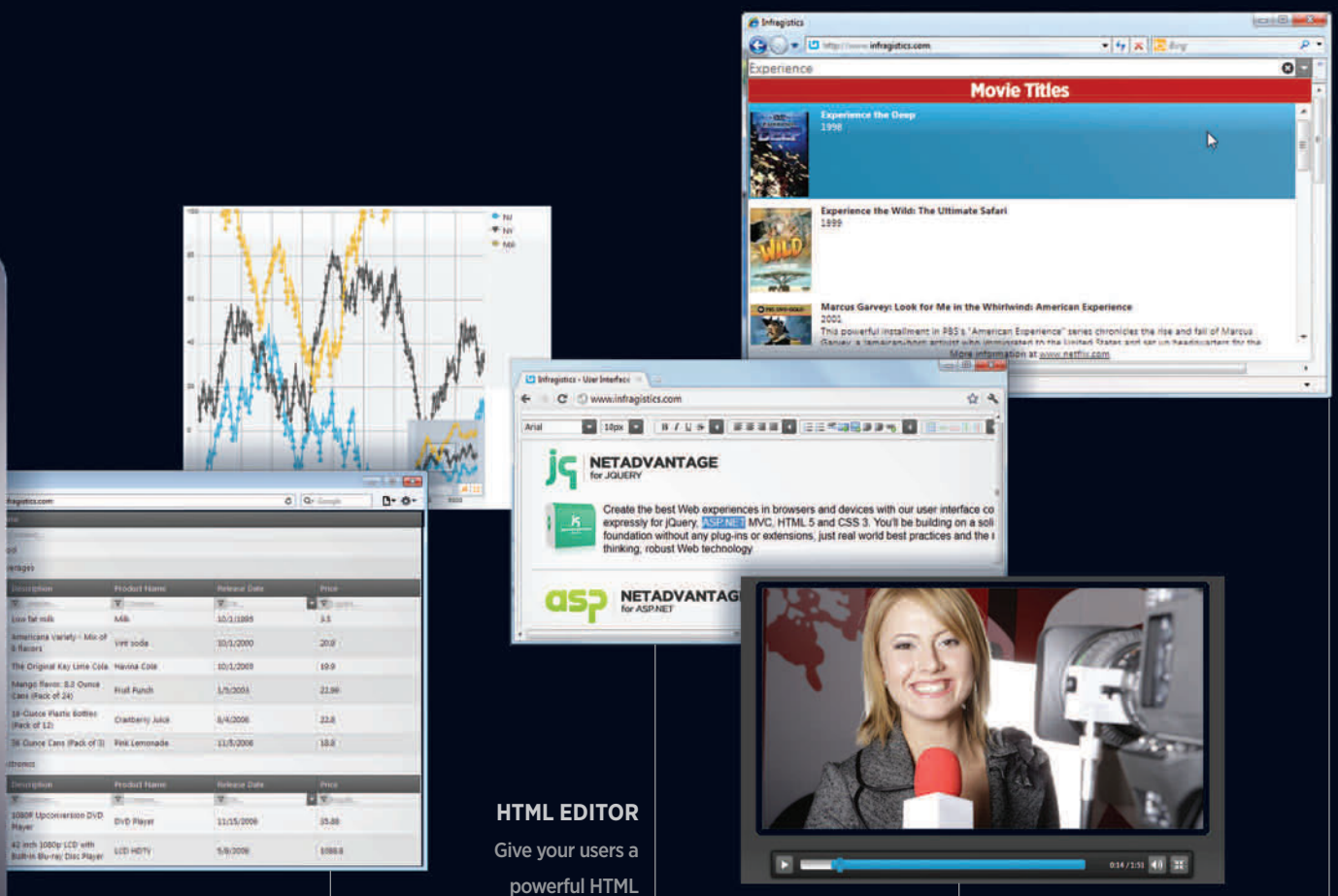
BUSINESS CHARTING

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



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.

msdn magazine

MARCH 2012 VOLUME 27 NUMBER 3

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.

LEADTOOLS®

DOCUMENT SDKs

• PDF READ/WRITE/EXTRACT/VIEW/EDIT • OCR/ICR/OMR/MICR

• DOCUMENT READERS/WRITERS/CLEANUP/PRE-PROCESSING
• FORMS RECOGNITION/PROCESSING • ANNOTATIONS

• IMAGE FORMATS & COMPRESSION • SCANNING • VIEWER CONTROLS

• C++ • .NET • SILVERLIGHT • WINDOWS PHONE • CLOUD SDK
• C DLL • WCF SERVICES • WF ACTIVITIES • WPF • ASP.NET & COM

DOWNLOAD OUR 60 DAY EVALUATION
WWW.LEADTOOLS.COM

LEAD TECHNOLOGIES INCORPORATED

SALES@LEADTOOLS.COM
800.637.1840

EDITOR'S NOTE

MICHAEL DESMOND

What You Want

Over the past two months in this space I've written about Charles Petzold and John Papa, two longtime *MSDN Magazine* columnists who have seen their share of change over their tenures. Petzold, of course, came on board while developers were still arguing about the relevance of GUIs. Today, he leads our coverage of cutting-edge mobile and touch UI development.

For Papa, the changes are no less significant. As the man once responsible for our Data Points column, he felt that data access and development had long been neglected by the industry. 10 years later, we've experienced a burst of data-oriented advancements in the Microsoft .NET Framework space—notably around technologies such as LINQ and the Entity Framework—and our coverage has scaled to match. More to the point, Papa is arguably the leading authority on Silverlight development. Yet you'll find him now writing as much about HTML5, jQuery and JavaScript as you will about Silverlight and its XAML sibling, Windows Presentation Foundation (WPF). Times change.

And yet, many of the most widely read *MSDN Magazine* articles over the past year were written not in 2011, but before 2010. Readers on the Web keep returning to articles with titles such as, "WPF Apps with the Model-View-ViewModel Design Pattern" (February 2009), "Data Binding in WPF" (December 2007) and "Regular Expressions Make Pattern Matching and Data Extraction Easier" (February 2007). Content that was important in June 2002 ("XML Comments Let You Build Documentation Directly from Your Visual Studio .NET Sources Files") or in July 2001 ("What You Need to Know to Move from C++ to C#") clearly remains important to developers today.

If you look at the traffic a bit, patterns emerge. WPF is a point of overwhelming interest, representing four of the top 10 most-visited articles in 2011. There's also a clear focus on design patterns, as three of the top 20 articles (including two in the top five) are focused on this topic. And I'm not at all surprised to see data-oriented features and columns peppering the ranks of most-read articles on the *MSDN Magazine* Web site over the past year. There's also healthy interest in C# development—based on articles dealing with issues like migrating to C# from C++ and calling Win32 DLLs using p/Invoke—as well as in ASP.NET.

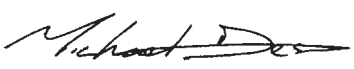
Obviously a lot of this traffic is being driven by searches, links from referring newsletters and Web sites, and who knows what other organic sources. But it's interesting that a list dominated by broadly deployed technologies includes the presence of one very fresh face: HTML5. Despite being published in the second half of 2011, Brandon Satrom's feature, "Building Apps with HTML5: What You Need to Know" (August 2011), ranks among the 10 most-read articles over the entire year. We've seen similarly impressive traffic for other HTML5-related features and columns published over the past several months.

Do these figures mean that there's little developer interest in articles on contemporary Microsoft platforms such as Windows Phone and Windows Azure? Not hardly. A glance at average page views per article by topic shows that articles on Windows Phone and Windows Azure published in 2011 actually produce Web readership slightly higher than data-oriented articles.

It's always a challenge to balance coverage of both established and emerging platforms and technologies in the magazine. Our goal is to ensure that developers are getting insight into the tools and techniques they need to succeed today, even as we introduce them to important new technologies that will define their jobs tomorrow. HTML5 is a case in point. Microsoft has been vocally supporting the emerging World Wide Web Consortium (W3C) standard for about two years now, and the announcements at the BUILD conference last September certainly proved out Redmond's conviction. Our response: Produce an ongoing series of features on HTML5 development and launch a new column—John Papa's Client Insight—to address this important new flank in the Microsoft developer ecosystem.

Still, the challenge remains: How can we best gauge which topics and technologies are most valuable to *MSDN Magazine* readers? In truth, the only way to answer that question is to ask the developers themselves.

So we're asking. What do you want to see in the pages of *MSDN Magazine* in 2012 and beyond? What topics do you feel are being given short shrift, and what technologies and issues are we perhaps over-covering? E-mail us at mmeditor@microsoft.com and let us know!



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

axosoft
800.653.0024

Everything You Need to Ship Software OnTime

Powerful bug tracking.

Manage bugs, defects, & issues with automatic notifications & alerts. Customize the tracker to match your development workflow. Track features and requirements, too—with drag and drop subitems support.

The **best** Scrum tool.

Utilize Scrum development or another agile process? OnTime is easy to adapt, and includes intelligent burndown & burnup charts, sprint & release backlogs, and the best interactive planning board out there.

A **beautiful & fast** UI.

You shouldn't spend all of your time in a software dev tool when you could be actually coding. OnTime's UI is fast and simple, and you can get what you need done right away. We think you'll really like it.

Plus: OnTime Mobile • GitHub Integration • Customer Portal & Help Desk • Built-in Scratchpad Remote Server • Windows, Visual Studio & Eclipse Clients • Premium Hosting • SDK • and more!

Visit axosoft.com to learn more.

Free 2-user hosted license. **Forever.**

The best way to find out how OnTime can help your team is to sign up for a free 2-user license. Hosted on our reliable, secure OnTime Now! service, you can get started right away. Plus, for 30 days you can add up to 10 total users—but you'll still have 2 users forever.

Did you **know?**

OnTime is built on the Microsoft .NET framework, and it has a Microsoft SQL Server back-end. You can use OnTime as a hosted service or install it in your own environment. With OnTime's APIs your dev team can extend the functionality of OnTime inside your apps.

Find us on:

@axosoft Axosoft OnTime /axosoft

axosoft OnTime11

Team up. Collaborate. Build. **Ship great software OnTime.**

DINO ESPOSITO

Build a Progress Bar with SignalR

In the past two installments of this column I discussed how to build an ASP.NET solution for the evergreen problem of monitoring the progress of a remote task from the client side of a Web application. Despite the success and adoption of AJAX, a comprehensive and widely accepted solution for displaying a context-sensitive progress bar within a Web application without resorting to Silverlight or Flash is still lacking.

To be honest, there aren't many ways in which one can accomplish this. You might craft your own solution if you want, but the underlying pattern won't be that different from what I presented—specifically targeting ASP.NET MVC—in the past columns. This month, I'm back to the same topic, but I'll discuss how to build a progress bar using a new and still-in-progress library: SignalR.

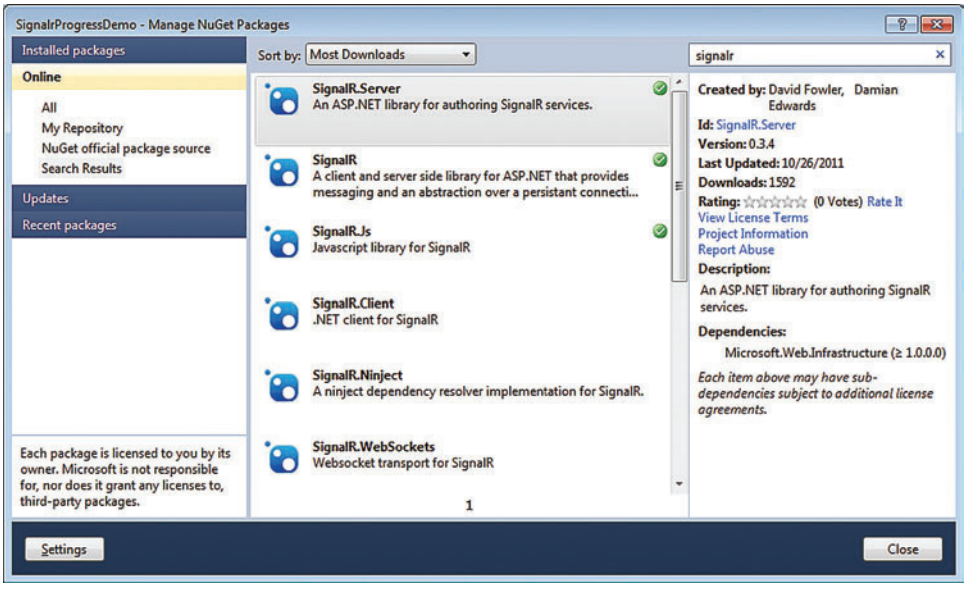


Figure 1 **SignalR Packages Available on the NuGet Platform**

SignalR is a Microsoft .NET Framework library and jQuery plug-in being developed by the ASP.NET team, possibly to be included in future releases of the ASP.NET platform. It presents some extremely promising functionality that's currently missing in the .NET Framework and that more and more developers are demanding.

SignalR at a Glance

SignalR is an integrated client-and-server library that enables browser-based clients and ASP.NET-based server components to have a bidirectional and multistep conversation. In other words, the conversation isn't limited to a single, stateless request/response data exchange; rather, it continues until explicitly closed. The conversation takes place over a persistent connection and lets the client send multiple messages to the server and the server reply—and, much more interesting—send asynchronous messages to the client.

It should come as no surprise that the canonical demo I'll use to illustrate the main capabilities of SignalR is a chat application. The client starts the conversation by sending a message to the server; the server—an ASP.NET endpoint—replies and keeps listening for new requests.

Code download available at code.msdn.microsoft.com/mag201203CuttingEdge.

SignalR is specifically for a Web scenario and requires jQuery 1.6 (or newer) on the client and ASP.NET on the server. You can install SignalR via NuGet or by downloading the bits directly from the GitHub repository at github.com/SignalR/SignalR. **Figure 1** shows the NuGet page with all SignalR packages. At minimum, you need to download SignalR, which has dependencies on SignalR.Server for the server-side part of the framework, and SignalR.Js for the Web-client part of the framework. The other packages you see in **Figure 1** serve more specific purposes such as providing a .NET client, a Ninject dependency resolver and an alternate transportation mechanism based on HTML5 Web sockets.

Inside the Chat Example

Before I attempt to build a progress bar solution, it would be useful to get familiar with the library by taking a look at the chat example distributed with the downloadable source code (code.msdn.microsoft.com/mag201203CuttingEdge) and other information referenced in the (few) related posts currently available on the Web. Note, though, that SignalR is not a released project.

In the context of an ASP.NET MVC project, you start by referencing a bunch of script files, as shown here:

```
<script src="@Url.Content("~/Scripts/jquery-1.6.4.min.js")"
  type="text/javascript"></script>
<script src="@Url.Content("~/Scripts/jquery.signalr.min.js")"
  type="text/javascript"></script>
<script src="@Url.Content("~/signalr/hubs")"
  type="text/javascript"></script>
```

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

Figure 2 **Setting Up the SignalR Library for a Chat Example**

```
<script type="text/javascript">
  $(document).ready(function () {
    // Add handler to Send button
    $("#sendButton").click(function () {
      chat.send($('#msg').val());
    });

    // Create a proxy for the server endpoint
    var chat = $.connection.chat;
    // Add a client-side callback to process any data
    // received from the server
    chat.addMessage = function (message) {
      $('#messages').append('<li>' + message + '</li>');
    };

    // Start the conversation
    $.connection.hub.start();
  });
</script>
```

Note that there's nothing specific to ASP.NET MVC in SignalR, and the library can be used equally well with Web Forms applications.

An interesting point to emphasize is that the first two links reference a specific script file. The third link, instead, still references some JavaScript content, but that content is generated on the fly—and that depends on some other code you have within the host ASP.NET application. Also note that you need the JSON2 library if you intend to support versions of Internet Explorer prior to version 8.

Upon the page loading, you finalize the client setup and open the connection. **Figure 2** shows the code you need. You might want to call this code from within the ready event of jQuery. The code binds script handlers to HTML elements—unobtrusive JavaScript—and prepares the SignalR conversation.

It's worth noting that the $.connection object is defined in the SignalR script file. The chat object, in contrast, is a dynamic object in the sense that its code is generated on the fly and is injected into the client page via the Hub script reference. The chat object is ultimately a JavaScript proxy for a server-side object. At this point it should be clear that the client code in **Figure 2** means (and does) little without a strong server-side counterpart.

The ASP.NET project should include a reference to the SignalR assembly and its dependencies such as Microsoft.Web.Infrastructure. The server-side code includes a managed class that matches the JavaScript object you created. With reference to the code in **Figure 2**, you need to have a server-side object with the same interface as the client-side Chat object. This server class will inherit from the Hub class defined in the SignalR assembly. Here's the class signature:

```
using System;
using SignalR.Hubs;

namespace SignalrProgressDemo.Progress
{
  public class Chat : Hub
  {
    public void Send(String message)
    {
      Clients.addMessage(message);
    }
  }
}
```

Every public method in the class must match a JavaScript method on the client. Or, at least, any method invoked on the JavaScript object must have a matching method on the server class. So the Send method you see invoked in the script code of **Figure 2** ends up placing a call into the Send method of the Chat object, as defined earlier. To send data back to the client, the server code uses the Clients property on the Hub class. The Clients member is of type dynamic, which enables it to reference dynamically determined objects. In particular, the Clients property contains a reference to a server-side object built after the interface of the client object: the Chat object. Because the Chat object in **Figure 2** has an addMessage method, the same addMessage method is expected to be exposed also by the server-side Chat object.

Figure 3 **The Final Version of the Hub Class**

```
public void BookFlight(String from, String to)
{
  // Book first leg
  Clients.displayMessage(
    String.Format("Booking flight: {0}-{1} ...", from, to));
  Thread.Sleep(2000);

  // Book return
  Clients.displayMessage(
    String.Format("Booking flight: {0}-{1} ...", to, from));
  Thread.Sleep(3000);

  // Book return
  Clients.displayMessage(
    String.Format("Booking flight: {0}-{1} ...", to, from));
  Thread.Sleep(2000);

  // Some return value
  Clients.displayMessage("Flight booked successfully.");
}
```

Toward a Progress Bar Demo

Now let's use SignalR to build a notification system that reports to the client any progress being made on the server during a possibly lengthy task. As a first step, let's create a server-side class that encapsulates the task. The name you assign to this class, while arbitrarily chosen, will affect the client code you'll write later. This simply means you have one more reason to choose the class name with care. Even more important, this class will inherit from a SignalR provided class named Hub. Here's the signature:

```
public class BookingHub : Hub
{
  ...
}
```

The BookingHub class will have a few public methods—mostly void methods accepting any sequence of input parameters that makes sense for their intended purpose. Every public method on a Hub class represents a possible endpoint for the client to invoke. As an example, let's add a method to book a flight:

```
public void BookFlight(String from, String to)
{
  ...
}
```

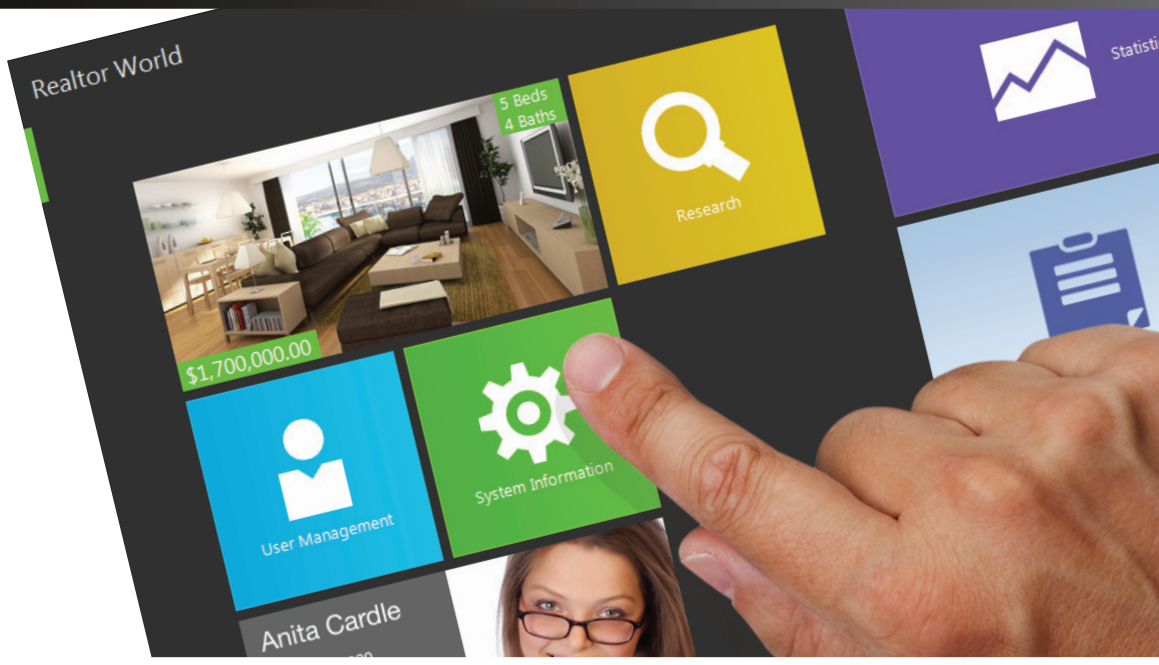
This method is expected to contain all the logic that executes the given action (that is, booking a flight). The code will also contain at various stages calls that in some way will report any progress back to the client. Let's say the skeleton of method BookFlight looks like this:

```
public void BookFlight(String from, String to)
{
  // Book first leg
  var ref1 = BookFlight(from, to);

  // Book return flight
  var ref2 = BookFlight(to, from);

  // Handle payment
  PayFlight(ref1, ref2);
}
```

Microsoft
GOLD CERTIFIED
Partner



DXv2 is here

Touch at your fingertips.

Bring your software to life with intelligent touch-based applications. Use your existing development skills to tap into the growing demand for stunning tablet & touch-enabled apps across all platforms, including WinForms, WPF and ASP.NET. Build for today as you begin to re-imagine business applications for the Windows 8 Metro design aesthetic. DXv2 delivers the gestures, themes, and controls to put Touch within your reach, right now.

DXv2 is the next generation of tools that can take your applications to a whole new level. Your users are ready—what will you build for them? Download your free 30-day trial at **www.DevExpress.com**

www.DevExpress.com

DevExpress

In conjunction with these main operations, you want to notify the user about the progress made. The Hub base class offers a property called Clients defined to be of type dynamic. In other words, you'll invoke a method on this object to call back the client. The form and shape of this method, though, are determined by the client itself. Let's move to the client, then.

As mentioned, in the client page you'll have some script code that runs when the page loads. If you use jQuery, the $(document).ready event is a good place for running this code. First, you get a proxy to the server object:

```
var bookingHub = $.connection.bookingHub;

// Some config work
...

// Open the connection
$.connection.hub.start();
```

The name of the object you reference on the $.connection SignalR native component is just a dynamically created proxy that exposes the public interface of the BookingHub object to the client. The proxy is generated via the signalr/hubs link you have in the <script> section of the page. The naming convention used for names is camelCase, meaning that class BookingHub in C# becomes object bookingHub in JavaScript. On this object you find methods that match the public interface of the server object. Also, for methods, the naming convention uses the same names, but camelCased. You can add a click handler to an HTML button and start a server operation via AJAX, as shown here:

```
bookingHub.bookFlight("fco", "jfk");
```

You can now define client methods to handle any response. For example, you can define on the client proxy a displayMessage method that receives a message and displays it through an HTML span tag:

```
bookingHub.displayMessage = function (message) {
  $("#msg").html(message);
};
```

Note that you're responsible for the signature of the displayMessage method. You decide what's being passed and what type you expect any input to be.

Figure 4 **Creating an HTML-Based Gauge Bar**

```
var GaugeBar = GaugeBar || {};

GaugeBar.generate = function (percentage) {
  if (typeof (percentage) != "number")
    return;
  if (percentage > 100 || percentage < 0)
    return;

  var colspan = 1;
  var markup = "<table class='gauge-bar-table'><tr>" +
    "<td style='width:" + percentage.toString() +
    "%' class='gauge-bar-completed'></td>";
  if (percentage < 100) {
    markup += "<td class='gauge-bar-tobedone' style='width:" +
      (100 - percentage).toString() +
      "%'></td>";
    colspan++;
  }

  markup += "</tr><tr class='gauge-bar-statusline'><td colspan='" +
    colspan.toString() +
    "'>" +
    percentage.toString() +
    "% completed</td></tr></table>";
  return markup;
}
```

To close the circle, there's just one final issue: who's calling displayMessage and who's ultimately responsible for passing data? It's the server-side Hub code. You call displayMessage (and any other callback method you want to have in place) from within the Hub object via the Clients object. **Figure 3** shows the final version of the Hub class.

Note that in this case, the displayMessage name must match perfectly the case you used in the JavaScript code. If you mistype it to something such as DisplayMessage, you won't get any exception—but no code will execute, either.

The Hub code is implemented as a Task object, so it gets its own thread to run and doesn't affect the ASP.NET thread pool.

If a server task results in asynchronous work being scheduled, it will pick up a thread from the standard worker pool. The advantage is, SignalR request handlers are asynchronous, meaning that while they're in the wait state, waiting for new messages, they aren't using a thread at all. When a message is received and there's work to be done, an ASP.NET worker thread is used.

A True Progress Bar with HTML

In past columns, as well as in this one, I used the term progress bar frequently without ever showing a classic gauge bar as an example of the client UI. Having a gauge bar is only a nice visual effect and doesn't require more complex code in the async infrastructure. However, **Figure 4** shows the JavaScript code that builds a gauge bar on the fly given a percentage value. You can change the appearance of the HTML elements via proper CSS classes.

You call this method from a button click handler:

```
bookingHub.updateGaugeBar = function (perc) {
  $("#bar").html(GaugeBar.generate(perc));
};
```

The updateGaugeBar method is therefore invoked from another Hub method that just uses a different client callback to report progress. You can just replace displayMessage used previously with updateGaugeBar within a Hub method.

Not Just Web Clients

I presented SignalR primarily as an API that requires a Web front end. Although this is probably the most compelling scenario in which you might want to use it, SignalR is in no way limited to supporting just Web clients. You can download a client for .NET desktop applications, and another client will be released soon to support Windows Phone clients.

This column only scratched the surface of SignalR in the sense that it presented the simplest and most effective approach to program it. In a future column, I'll investigate some of the magic it does under the hood and how packets are moved along the wire. Stay tuned. ■

DINO ESPOSITO *is the author of "Programming Microsoft ASP.NET MVC3" (Microsoft Press, 2011) and coauthor of "Microsoft .NET: Architecting Applications for the Enterprise" (Microsoft Press, 2008). Based in Italy, he's a frequent speaker at industry events worldwide. You can follow him on Twitter at twitter.com/despos.*

THANKS to the following technical expert for reviewing this article: *Damian Edwards*

Kendo UI

THE ART OF WEB DEVELOPMENT

Everything You Need

For JavaScript & HTML5 development

Kendo UI is a complete, integrated package that provides developers a jQuery-based toolset which includes a powerful data source, dynamic data visualizations, and blazing fast micro-templates, all backed by industry leading professional support. Download Kendo UI and start building amazing sites and apps today.

iOS HTML CSS jQuery

Download the future of JavaScript development at **www.kendoui.com** or scan

Develop Once, Deploy Anywhere.

ComponentArt's unique technology targets the widest range of devices with a single code base. Develop once in Visual Studio and XAML, deploy anywhere in a variety of formats.

HTML5

HTML5 is supported by virtually any device: Apple iPad & iPhone, Android Tablets & Phones, Windows Tablets & Phones, Blackberry Playbook & Phones, and any PC or Mac.

XAML/WinRT

Windows 8 features native XAML support and a new WinRT library. Deploy immersive Metro XAML apps through ComponentArt Data Visualization for XAML/WinRT.

Silverlight & WPF

Classic Windows interface continues to be supported through mature .NET development tools: ComponentArt Data Visualization for Silverlight & WPF.

Single code base. Any output format.

Powered By: Microsoft Visual Studio · ComponentArt Data Visualization · ComponentArt Dashboard Server

Sounds unbelievable?

See for yourself at www.ComponentArt.com

JULIE LERMAN

Entity Framework Code First and DbContext FAQs

This month, I'd like to share the answers to questions I'm frequently asked about Code First and DbContext in Entity Framework 4.2. To maximize space, I'll point to additional resources where you can find detailed information to supplement the material I provide.

Will Code First and DbContext Be Added to the .NET Framework?

No. Code First and DbContext will be distributed separately from the Microsoft .NET Framework release cycle, allowing the Entity Framework team to release updates when they're ready without waiting for the next official release of the .NET Framework. Code First and the DbContext API are part of the EntityFramework.dll assembly, which you can add to your projects dynamically using NuGet. Installing the NuGet extension in Visual Studio gives you the Library Package Manager, which lets you add a reference to EntityFramework.dll to your project. **Figure 1** shows the EntityFramework package (currently the most popular download in the NuGet package library) in the Visual Studio Library Package Manager.

You can read more about the team's plans for releasing features of Entity Framework and why they decided to focus on using NuGet for deploying certain types of features outside of the .NET APIs in their October 2011 blog post, "How We Talk About EF and Its Future Versions" (bit.ly/owoWgn).

Can You Filter Related Data with Include?

No. The Include method allows you to eagerly load related data when executing queries against an Entity Data Model. Unfortunately, you can retrieve only complete sets of related data with eager loading. This is also true for lazy loading and, in all but one case, for explicit loading. That single case: When you explicitly load through the change tracker API, you can apply a Query method followed by a Where filter:

```
context.Entry(personInstance)
  .Collection(p => p.Aliases)
  .Query()
  .Where(a=>a.FirstName == "Natasha")
  .Load();
```

But to be clear, you can't do this for eager loading with Include.

This is how Include has worked since the first release of Entity Framework in the .NET Framework 3.5. If you want to combine eager loading with filtering, consider using projections. You can read more about this in the June 2011 Data Points column, "Demystifying Entity Framework Strategies: Loading Related Data," at msdn.microsoft.com/magazine/hh205756. I've also written a blog post on the topic, "Use Projections and a Repository to Fake a Filtered Eager Load," at bit.ly/uZdnxy.

How Can I Use Lambdas with Include?

With the ObjectContext, the only way to use Include is by passing in a string to represent the navigation property to be eagerly loaded. For example, if you have a class of Road that has a property, Trees, which is an ICollection<Tree>, you would query for Roads with their Trees like this:

```
context.Roads.Include("Trees")
```

Many developers have created their own extension methods to allow them to use a strongly typed expression instead:

```
context.Roads.Include(r => r.Trees)
```

The ability to use lambda expressions with Include is now built into the DbContext API. You'll find an example in Part 6 of Arthur Vickers' wonderful 12-part blog series about the DbContext API (bit.ly/wgl5zW). Your attempts to use this feature might have been unsuccessful, however—as were mine. I had to put my ego aside and ask someone on the team how to get the lambda support. It turns out that in addition to a reference to EntityFramework.dll, you need a using directive (or Imports for Visual Basic) for System.Data.Entity in the code file where you're attempting to use Include with a lambda. If you want to expand the eager loading on a collection to multiple levels of relationships using the lambdas, you have to incorporate extra Select methods. For example, if your Tree class has an ICollection<Leaf> and you want to eager load the leaves along with the trees, the string representation would look like this:

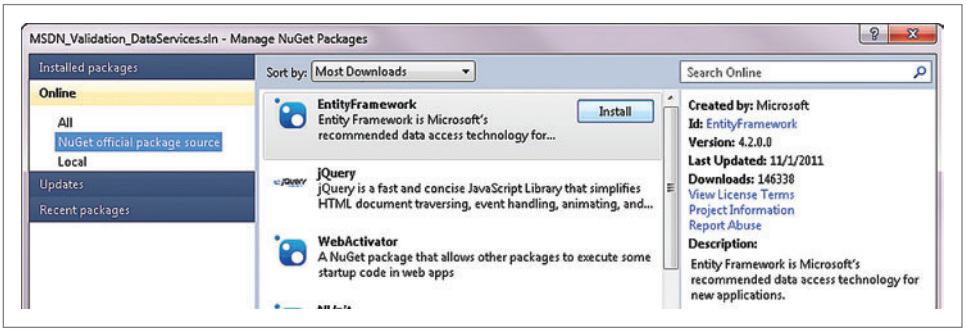


Figure 1 **Installing Code First and DbContext into a Project via EntityFramework.dll**

The best ideas evolve.

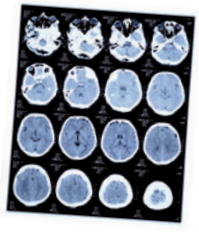
Great ideas don't just happen. They evolve. Your own development teams think and work fast.

Don't miss a breakthrough. Version *everything* with Perforce.

Software and firmware. Digital assets and games. Websites and documents. More than 5,500 organizations and 400,000 users trust Perforce for enterprise version management.

Try it now. Download the free 20-user, non-expiring Perforce Server from perforce.com/free20

Or request an evaluation license for any number of users.



PERFORCE
Version everything.

```
context.Roads.Include("Trees.Leaves")
```

However, the lambda relies on additional logic to provide the next level of the relationship. Here's the syntax you'd use to select roads along with the trees and their leaves:

```
context.Roads.Include(r => r.Trees.Select(t => t.Leaves))
```

I can't talk about using multiple levels of Include without adding a reminder that the more relationships you attempt to eager load with Include, the more complex and degraded your SQL query could become. I always highly recommend profiling the SQL generated by Entity Framework (EF). In fact, the December 2010 Data Points column demonstrates various ways to profile the database when using EF.

I was surprised to learn that if you're using the ObjectContext API, a reference to EntityFramework.dll plus a System.Data.Entity *using* directive will let you use lambdas with Include here, as well.

Do I Have to Wait for the .NET Framework 4.5 If I Want to Use Code First with WCF Data Services?

There are two assemblies for creating and consuming WCF Data Services in the .NET Framework 4: System.Data.Services.dll and System.Data.Services.Client.dll. If you try to use these with a DbContext and Code First classes, they won't work out of the box. The issue is with DbContext, not Code First. DbContext didn't exist when those assemblies were built, so they don't understand it. In March 2011, Microsoft released a Community Technology Preview (CTP) that contained fixed-up assemblies (Microsoft.Data.Services.dll and Microsoft.Data.Services.Client.dll) that know how to work with the DbContext. So all you have to do is replace the default assemblies with these new assemblies and, in your data services, specify DataServiceProtocolVersion as V3 rather than V2. Here's an example of what a simple data service might look like exposing a context (PersonModelContext) that inherits from DbContext:

```
public class DataService : DataService<PersonModelContext>
  {
    public static void InitializeService(DataServiceConfiguration config)
    {
      config.SetEntitySetAccessRule("People", EntitySetRights.All);
      config.DataServiceBehavior.MaxProtocolVersion =
DataServiceProtocolVersion.V3;
    }
```

The problem with this solution is that the new assemblies are only CTPs and not for production purposes. You'll have to wait for the next Released to Manufacturing (RTM) release of Data Services to be able to use them in production.

But it's possible to use a DbContext with .NET Framework 4 System.Data.Services assemblies. The CTP assemblies exist just to help you avoid writing extra code. All you need to do is access the underlying ObjectContext that supports the DbContext and provide that to the service. In July 2010, Rowan Miller, a key player at Microsoft behind Code First, demonstrated this in a blog post, "EF CTP4 Tips & Tricks: WCF Data Service on DbContext" (bit.ly/c2EA0l). That blog was written for an early preview of the DbContext, so I've updated his example to work with the version of DbContext class in Entity Framework 4.2 using PersonModelContext, a simple model that exposes one DbSet for a Person class. The context and class are shown in **Figure 2**.

The data service is then defined to work with an ObjectContext, rather than specifically for the PersonModelContext that derives from DbContext. Then, by overriding the data service CreateDataSource method, you can expose the ObjectContext that underlies the PersonModelContext, thereby providing the needed ObjectContext to the service, as shown in **Figure 3**.

Because I want to focus on providing the ObjectContext, this is a simplified view of the code you might have in WCF Data Services.

The downside to this approach is that you won't be able to use DbContext features such as Validation (as described in my December 2011 Data Points column, "Handling Entity Framework Validations in WCF Data Services," at msdn.microsoft.com/magazine/hh580732) because the service is designed to work with an ObjectContext.

Should I Use Data Annotations or the Fluent API for Code First Configurations?

I get asked this question a lot. To understand the answer, it's helpful to first understand the high-level differences between these two options.

Code First extends the Data Annotations that are already part of the .NET Framework 4 System.ComponentModel.DataAnnotations namespace. In addition to annotations such as ValidationAttributes (Required, StringLength, RegEx), Entity Framework adds attributes to specify database mappings (for example, Column, Table and Timestamp) or other model-specific features such as ComplexType. Annotations are simple to apply in a class, as the Person class in **Figure 2** demonstrates.

There are some possible drawbacks to using Data Annotations to configure your classes for Code First. One is that you might not be a fan of adding persistence-related logic to your business classes. If you've asked yourself, "What does the name of a database column have to do with my domain model?" you might not want to use the Data Annotations. Another is that these mappings require you to reference EntityFramework.dll from assemblies that contain your business classes. (This changes as of the .NET Framework 4.5, which will take ownership of the EF-related Data Annotations.) That, too, might not be something you wish to do, especially if you're distributing your application across tiers. Moreover, the Code First Data Annotations don't expose the complete lineup of configurations that can be applied using Code First. Some configurations, for example those specifying particular details for a Table Per Hierarchy (TPH) mapping, can be achieved only using the Fluent API.

My recommendation to developers is to pick the style you like. If you prefer to have your configurations expressed by the Entity Framework DbContext, then use the Fluent API. If you like the simplicity of applying attributes and don't mind having the Code

Figure 2 **Simple DbContext and Class Used by a Data Service**

```
public class PersonModelContext : DbContext
{
  public DbSet<Person> People { get; set; }
}

public class Person
{
  public int PersonId { get; set; }
  [MaxLength(10)]
  public string IdentityCardNumber { get; set; }
  public string FirstName { get; set; }
  [Required]
  public string LastName { get; set; }
}
```

Figure 3 **A WCF Data Service that Uses a DbContext with .NET Framework 4 Service APIs**

```
public class DataService : DataService<ObjectContext>
{
  public static void InitializeService(DataServiceConfiguration config)
  {
    config.SetEntitySetAccessRule("People", EntitySetRights.All);
    config.DataServiceBehavior.MaxProtocolVersion =
      DataServiceProtocolVersion.V2;
  }

  protected override ObjectContext CreateDataSource()
  {
    var dbContext = new PersonModelContext();
    var oContext = ((IObjectContextAdapter)dbContext).ObjectContext;
    return oContext;
  }
}
```

First mapping logic in your classes, use Data Annotations. If you've chosen to use Data Annotations and come across a configuration that can be expressed only with the Fluent API, then add that configuration fluently and use a combination of Data Annotations and Fluent configurations.

Can I Use Code First with an Existing Database?

Absolutely! Although the default behavior for Code First is to (helpfully) create a database for you, you can point to an existing database with your own connection string and have Code First use that. There are some internal features of Code First that let you programmatically specify the database as well. For example, you can overload the DbContext constructor with a database name or a database connection string. You can also specify and even customize DbContext ConnectionFactory types (such as SqlConnectionFactory).

You'll need to ensure that Code First conventions plus any configurations you've added accurately represent how your classes will map to the existing database. Microsoft has been working on a tool that can reverse-engineer a database into classes plus Code First fluent configurations, which can provide a great head start. You can learn more about this tool in my May 2011 blog post, "Quick Look at Reverse Engineer DB into Code First Classes" (bit.ly/lHJ20r).

When specifying the location, avoid the schema validation and database initialization that DbContext does by default with Code First. You can completely disable this feature via Code First database initialization:

```
Database.SetInitializer<MyContext>(null)
```

You'll find more on this topic in the MSDN Data Developer Center article and video, "Database Initializers in Entity Framework 4.1" (msdn.microsoft.com/data/hh272552). For detailed examples of working with the ConnectionFactory classes and DbContext overloads, be sure to check out "Programming Entity Framework: Code First" (O'Reilly Media, 2011), a book I coauthored with Rowan Miller from the Code First team.

Additional Resources

There are many places to learn about working with Code First and DbContext. The Entity Framework team blog at blogs.msdn.com/adonet is filled with walk-throughs and examples. Some team member blogs, such as Rowan Miller's (romiller.com) and Arthur Vickers' (blog.oneunicorn.com/author/ajcvickers), are great places to read about advanced techniques. The MSDN forums and StackOverflow.com continue to be great places to read threads or ask your own questions. Miller and I recently wrote two short books, the aforementioned "Programming Entity Framework: Code First" and "Programming Entity Framework: DbContext" (O'Reilly Media, 2012). You can get these in print or digital format.

There's no need to struggle with trying to figure things out on your own. ■

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 experts for reviewing this article: *Mike Flasko and Diego Vega*

JOSEPH FULTZ

Exploring Cloud Architecture

I decided a while back that at least one of my columns would cover general architectural considerations and some base high-level designs for creating cloud-based solutions. I know from all of the sessions I've presented at or attended that if a technical audience doesn't get to play with some code, the sessions aren't usually well-received. But I'm hoping this article will benefit folks by at least helping to frame and give some basic targets for their thinking about solution architecture.

I'll be using Platform as a Service (PaaS)—in particular, Windows Azure—as the contextual corkboard on which to hang the ideas and considerations, but the ideas I want to convey should also apply to Infrastructure as a Service (IaaS) and Software as a Service (SaaS) with little translation. I'll point out some of the key design considerations when creating a cloud solution, categorize the platform tools Windows Azure provides, illustrate some base designs and provide a decision matrix for selecting your design style.

Cloud Considerations

Seasoned developers and architects might get into a habit of identifying some problem characteristics, matching a pattern and pronouncing a design. I'm going to highlight some typical designs for cloud solutions, but I highly encourage taking an active approach to solving problems versus just using a cookie cutter from the previous batch of solutions that were baked up. The best approach is to design a solution that's ideal and then start overlaying conditions that the cloud adds and conditions of the enterprise ecosystem. Add them one at a time and solve any associated problems, because just like math or proper debugging, if you add too many variables, you'll introduce errors whose source can't be easily ascertained.

What does it mean to design a "cloud solution?" I want to start with a strong emphasis that a Web Role queuing to a Worker Role and then picking the result up off a return queue isn't the defining style of cloud design. In fact, in many regards, using such architecture as a base from which to launch a design might in fact reduce the overall benefit of a cloud implementation. As private cloud infrastructure becomes readily available, the line blurs between cloud and not-cloud because the delineating factor isn't location. So the focus for differentiation needs to be on what it does rather than what it is or where it is.

Here are the things that using Windows Azure as a means to host my Web or service applications does for me:

- Frees me from infrastructure acquisition, setup or management
- Frees me from constrained capacity; computing and storage capacity are available on demand
- Frees me from steep capital cost and frees me to adjust cost based on consumption

The challenges addressed directly with the cloud shift are compute and storage elasticity and capital expense.

I don't mean to trivialize it, but that's it; no need to complicate it. Indeed, there are a lot of nontrivial features within the cloud platform, but those aren't what differentiate the cloud from a more traditional hosted solution. Individual features (queuing, storage, structured storage and so on) that are added for a platform such as Windows Azure—while new for the platform—aren't fundamentally new software capabilities that weren't available before. The challenges addressed directly with the cloud shift are compute and storage elasticity and capital expense, along with ongoing care and feeding of the hardware infrastructure. This is good news for those designing software because it means designs and patterns remain mostly the same, with a little extra emphasis on two

Figure 1 **Platform Toolbox Categories**

Windows Azure	Compute	Storage	Integration & Services
Compute	Web Roles		
	Worker Roles		
	VM Role		
Storage		Blob Storage	Queue Storage
		Table Storage	
Formerly AppFabric			Service Bus
			Connect
			Access Control
			Caching
SQL Azure		Database	
	Reporting		
			SQL Azure Data Sync

VSP²

VISUAL STUDIO PARTNER PROFILE

A Visual Studio 2012 Q&A with Rakesh Malhotra of Apprenda

Rakesh Malhotra
VP of Products
Apprenda

Prior to joining Apprenda, Malhotra was at Microsoft for more than nine years where he most recently was principal group program manager for cloud and data center management and was among the one percent of employees nominated to Microsoft's Corporate Leadership Bench Program.

Apprenda is an Open Platform-as-a-Service (PaaS) stack for .NET that enables organizations to transform their existing infrastructure into a self-service cloud application platform.

Q What makes Apprenda different from Azure and how does it fit into the Microsoft ecosystem?
A We share the same world view as Microsoft; that customers will embrace a hybrid cloud strategy. By helping customers create symmetry, both technically and operationally, between on-premise private clouds and public clouds like Azure, Apprenda makes it easier for customers to consume Azure and integrate it into their datacenter portfolio as a part of a hybrid cloud strategy. If you're a Microsoft customer who has built or is building a private cloud using Windows Server Hyper-V and System Center, you get a terrific infrastructure-as-a-service solution (IaaS). Apprenda extends that IaaS capability by providing a private platform-as-a-service (PaaS) offering which helps both developers and IT Pros get additional benefits.

Q What are the benefits of providing a private platform-as-a-service offering?
A Simply put—more value and less friction. Apprenda enables developers and IT pros to work independently, but toward the same goals in a frictionless way. Apps get out the door faster, developers are more productive (and happier), and IT gets to maintain control and quality of service.

Q How does Apprenda help developers?
A For developers, Apprenda lets them focus on applications rather than infrastructure (which is still prominent in an IaaS private cloud). We provide an SDK which allows them to accelerate adding features like chargeback and billing. We have a self-service portal allowing developers to instantly deploy applications into their private cloud without worrying about servers, load balancers, storage arrays or other infrastructure components. Once running, Apprenda enhances their application by automatically providing capabilities such as multi-tenancy, high availability and the ability to scale in/out on demand to meet business needs.

Q What about the IT Pros?
A For IT Pros, Apprenda enables them to operate like a world-class cloud services provider; giving their developers all of the agility of public cloud but with the controls, compliance, SLAs and enterprise integration features that they require. Apprenda provides the richest .NET PaaS environment on the market today.

Q What's the easiest way for a developer to get started?
A Download Apprenda Express. It's a completely free edition of the platform, that's only limited by the underlying infrastructure footprint it can be installed on. It provides all of the same enterprise grade capabilities, but for small scale environments. It's great for smaller organizations or independent developers, and as a way for organizations to test the platform out.

considerations: cost and latency. Certainly these two concerns were important before, but the cloud changes the equation enough as to require special attention.

The Cost Factor

As many of you already seasoned from cloud implementations know, it isn't always cheaper. Fortunately, cost efficiency doesn't equate to cheap or even free. You can determine the cost to be efficient when the equation can't be changed to reduce cost without also reducing the services or functionality—meaning that there's no cost above and beyond what's necessarily consumed. There's no doubt that there are cost optimizations to be had for an enterprise switching from a private datacenter to a cloud-based solution set, but to really see that benefit, there needs to be an affinity toward guiding principles: a shift toward cloud-first solutions and design for cost optimization.

There must be a shift toward cloud computing and away from the private hosting scenario. It's hard to realize the cost benefit of hosting in the cloud while there's unused or underused infrastructure available to consume that has already been purchased and is in place. Thus, as systems age or new software comes on board, you should look to the cloud first instead of buying more infrastructure bandwidth. There are times when owning the datacenter is indeed cheaper, but for most enterprises that's not the reality. Contract and bulk rates aside, generally the closer you get to consumption-based cost, the closer you get to an efficient cost model.

> Cost efficiency doesn't equate to cheap or even free.

With the first step taken, the next step is to ensure you're not a cloud glutton. Each application needs to be designed around optimizing for cost. While the first point is often difficult from a business perspective, the second point can be a painful pill to swallow in a technical design. That's because there will be times when designing to optimize for cost means putting it in front of performance or elegance. Usually I find that creating equilibrium between performance and elegance is where I spend a good bit of my effort, but throw cost optimization in the picture and it gets further complicated by turning my 2D problem into a 3D one.

Optimizing cost means being minutely aware of the following:

- Ingress and egress, both to the end consumer and to the corporate backbone
- Retention policy for data in storage so as to not use unneeded space
- How to responsibly scale up and scale down compute resources

If a design takes into account these three considerations, then it's definitely moving in the right direction.

The Latency Factor

Traditionally, latency becomes an issue when the information that must be served to the end user exists in one or more of these states:

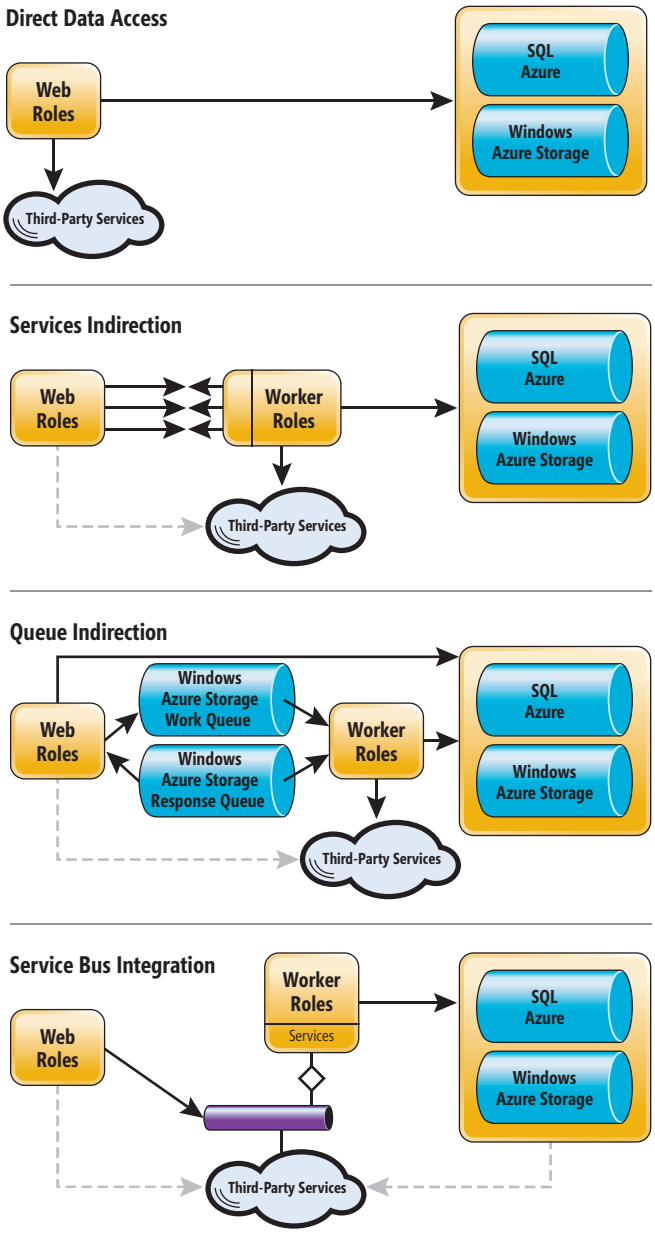


Figure 2 **Basic High-Level Designs**

- It's in a legacy or otherwise constrained system
- It exists across systems and must be aggregated first
- It requires one or more transformations before being handed to the UI
- It exists outside of the enterprise and must be remotely fetched

For a cloud-based solution, any and all of these items might be true, but it's the last item that applies without exception to designing enterprise solutions for the cloud. Certainly the other items will likely be factors as well, but a universal truth for enterprises today is that data will have to be somehow synchronized between the cloud solutions and the corporate data stores in the home datacenter.

I'll cover a few ways of integrating data in the base designs I set forth here, but it really comes down to two primary strategies: data synchronization and real-time service calls back home.

Less Pain, More Gain

check out infragistics.com/reporting

NetAdvantage® for Reporting

EXPORT TO EXCEL, WORD AND PDF

Export reports from the client and server side in the popular format of your choice!

DATA ACCESS SUPPORT

Create MVVM-friendly reports with data accessed from an SQL Server, Oracle or any Object Data Source.

DESIGN INTERFACE

Optimize your data presentation and build attractive reports with an integrated and easy-to-use design-time experience.

REPORT VIEWER

View pixel-perfect reports with vector graphics in our Silverlight, ASP.NET, WPF and Windows Forms report viewer.

INFRAGISTICS
DESIGN / DEVELOP / EXPERIENCE

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

High-Level Design

Thinking about the generalization of design for enterprise solutions, one can usually lump each of the big gears into one of these categories:

- Compute
- Storage
- Integration

This is how I'll categorize the Windows Azure platform capabilities in this context. The grid in **Figure 1** shows a subset of the Windows Azure platform capabilities.

Focusing on the big bricks of compute, storage and integration, you can easily put together some base designs from which to start and expand or even recombine, as shown in **Figure 2**.

> One basic way to connect your cloud solutions to your corporate ecosystem is to simply use Windows Azure Connect.

While the designs shown in **Figure 2** aren't representative of all the possible permutations, they're bigger blocks of design that have their purpose:

- **Direct Data Access:** This is usually the most basic implementation where the front-end Web interface not only accesses data directly but also carries the burden of all the work.
- **Services Indirection:** This is a service-oriented architecture (SOA) design that's the next level of indirection and workload balancing, moving data access and business logic off of the roles serving the UI. A good SOA design can not only reduce the fragility of the solution but also reduce the load on the back end and optimize the data responses to the client.
- **Queue Indirection:** Work is offloaded to background workers via Windows Azure Storage Queues. This is a first step at optimizing the workload but is generally suitable when the result of the offloaded work is accepted as not requiring near-time response to the client.
- **Service Bus Integration:** Adding in a pub/sub facility to the services infrastructure can really augment a SOA design by moving the layer of indirection away from the endpoints and moving toward topics of interest. This design can provide the most flexible implementation.

Corporate Backplane Design

The base designs don't address the corporate backplane specifically. Two of the most important pieces of a Windows Azure solution are how it connects to the corporate systems that contain proprietary business logic and how it accesses corporate data. There are some basic designs for the corporate backplane that are used by themselves or in combination, much like the previous designs, as shown in **Figure 3**.

One basic way to connect your cloud solutions to your corporate ecosystem is to simply use Windows Azure Connect as discussed

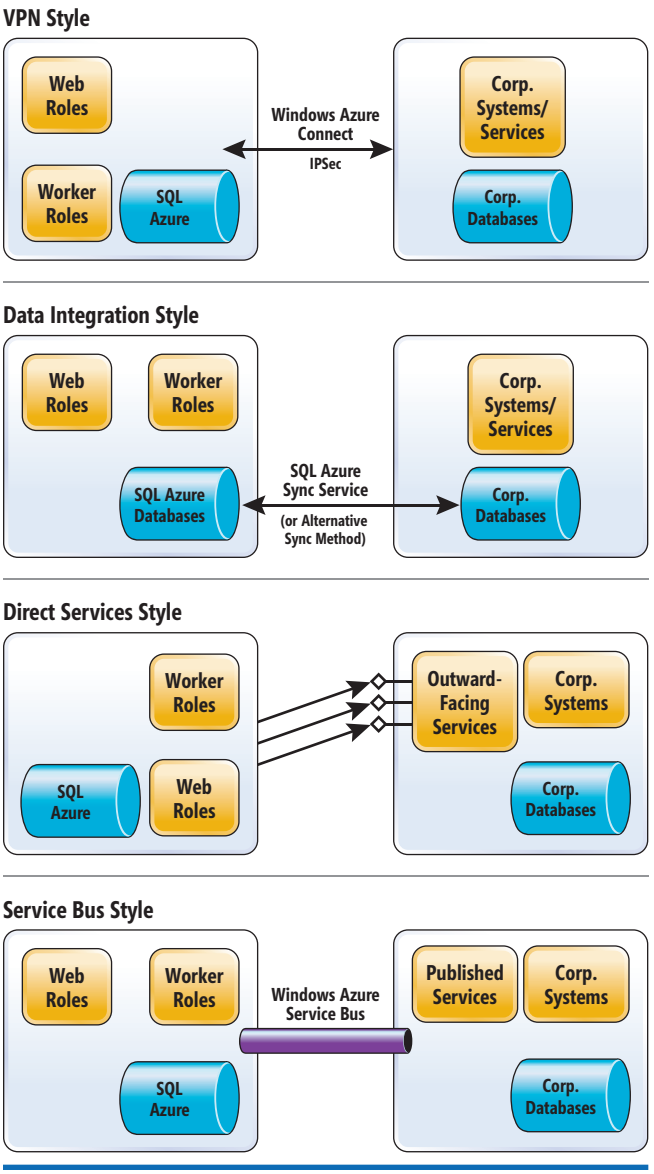


Figure 3 **Corporate Backplane Basic Designs**

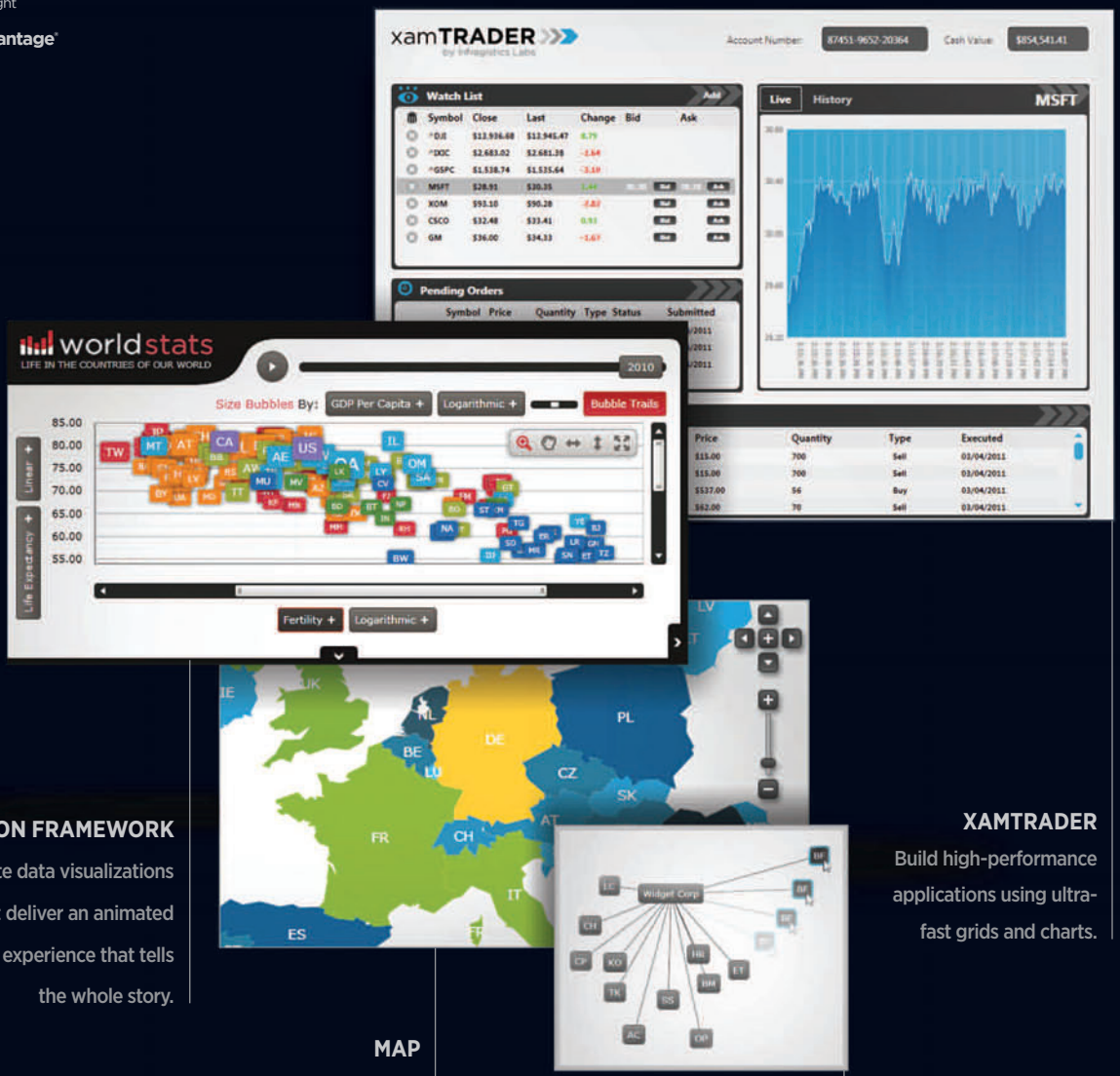
by Anna Skobodzinski, one of my colleagues from the Microsoft Technology Center (MTC), at bit.ly/wnyqj9. This is depicted in the VPN Style design in **Figure 3**. The system can then be used as is, but with a tax of about a second. Additionally, it will be worth doing some work ahead of time to make sure the systems being exposed to the cloud applications are ready for the additional load. If you need it in a hurry, the cloud is a great solution; if you need it in a hurry and it has to integrate with existing interfaces, VPN Style integration might be the fastest and easiest way to mitigate risk to your timeline.

The most common type of integration that I come across is the type depicted under Data Integration Style. This requires some means of synchronizing data in one or both directions. There are numerous ways to accomplish this, each with its own set of pros and cons. Using an extract, transform and load (ETL) tool such as SQL Server Integration Services (SSIS) is probably the lowest barrier to entry for most. SSIS as a means to synchronize data also has a very mature ecosystem for monitoring and management. The next-easiest

XAML-IFY YOUR APPS

check out infragistics.com/xaml

- **NetAdvantage®** for Silverlight Data Visualization
- **NetAdvantage®** for WPF Data Visualization
- **NetAdvantage®** for Silverlight
- **NetAdvantage®** for WPF



MOTION FRAMEWORK

Create data visualizations that deliver an animated user experience that tells the whole story.

MAP

Ensure your geospatial data really goes places with a feature-laden, interactive Map Control for your applications.

XAMTRADER

Build high-performance applications using ultra-fast grids and charts.

NETWORK NODE

Help your users make the connection with visual representations of simple or complex network relationships.

INFRAGISTICS
DESIGN / DEVELOP / EXPERIENCE

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

Figure 4 **Decision Matrix**

	Time to Solution	Solution Complexity	Monitoring & Management	Agility & Scalability	Latency	Freshness
Basic Designs						
Direct Data Access						
Queue Indirection						
Services Indirection						
Service Bus						
Backplane Designs						
VPN						
Data Integration						
Services Integration*						
Service Bus						

** Service infrastructure already available in corporate infrastructure*

mechanism is to use SQL Azure Data Sync, which is currently available as a preview (bit.ly/p14qC6). It's built on Sync Framework technology that I covered in my January 2011 (msdn.microsoft.com/magazine/gg535668) and February 2011 (msdn.microsoft.com/magazine/gg598920) columns. This will require some changes to the tables being synchronized. In both the SSIS and Sync Framework methods, it's a good idea to make sure to only move the fields of data that are necessary between cloud and corporate premises. By not moving data unnecessarily, the cost of ingress, egress and overall cloud storage usage can be kept down.

Moving the needle forward a little bit and taking some of the work away from the database back end is the Direct Services Style shown in **Figure 3**, where data needed from corporate systems is fetched via a set of exposed services. The two big upsides here are that the cost of data transport is removed and the data is always current. Less positively, it does introduce latency and put additional strain on systems that might not be ready to be incorporated into a cloud solution. It also implies that most of the intelligence has to live in the corporate systems and services, as it will have the bulk of the data to analyze.

The last of the four patterns in **Figure 3** shows the use of a service bus to integrate the cloud and the corporate systems. This Service Bus Style would entail that both the running cloud roles and the corporate systems publish and subscribe to data and events that are relevant to each of them. This has about the same pros and cons as the Direct Services Style of architecture, with the added benefit of maximum flexibility for creating new publishers and consumers of information.

The decision matrix is a tool to help identify potential problem areas when considering the use of one of the basic design styles over another. The decision matrix in **Figure 4** isn't an absolute, but rather reflects my experience in working with folks on cloud architecture. There isn't necessarily a clear winner, and in many cases a real design will incorporate some mix of these elements. For example, one might deploy a new Web application that has a set of REST services working against a back-end database that's synchronized via an ETL run twice per day. **Figure 5** shows an example of such a combined design.

Additionally, there's a set of file-based resources in Windows Azure Storage that's used by the site and is kept in sync by a custom Sync Framework implementation. This high-level design is the end result of collecting information and making some informed guesses, but it's only the start of the work. It provides the target for end design and implementation but leaves a lot of details to be decided (for example, caching, access control and so on).

Design and Cost Efficiency

I tried to tease out and identify the most important considerations when designing a cloud solution—in particular, one to be incorporated into an existing enterprise. Hopefully, by boiling off the plethora of features and capabilities a bit, it helps to clear the path for design. The real trick to designing a cloud solution is to take an ideal solution and make the design efficient while still including cost efficiency. As the cloud movement puts enterprise-level computing within reach of anyone willing to lean over and flip the switch, the elegance in design will account for consumption cost.

I touched upon a base Web design and a base corporate integration design, but left all of the details for a future drill-down; it would simply take too much space to cover at once. Because this is the only column I've written that focuses solely on design and has no implementation associated with it, I hope it will be useful. If you'd like me to drill down into some of the detail design pieces, please comment on this article on msdn.microsoft.com/magazine. ■

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: *George Huey*

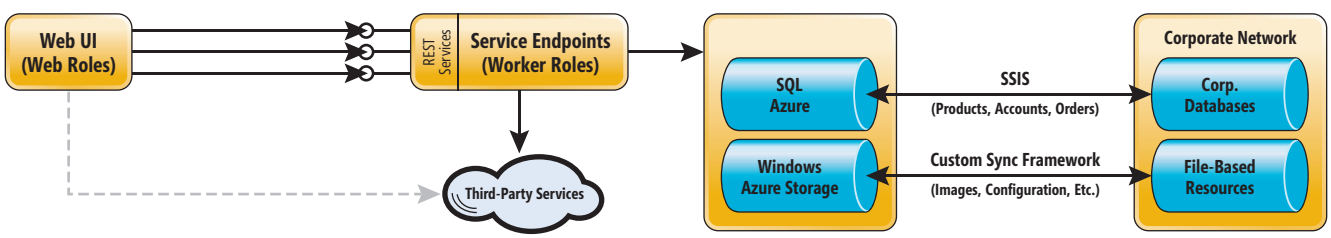


Figure 5 **Example Combined Design**

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

Coding, Casinos and More!

Learn how to **maximize the development capabilities of Visual Studio and .NET** over 5 information-packed days in Las Vegas. Join us for 60+ sessions & workshops full of unbiased, hard-hitting and practical education lead by industry experts and Microsoft insiders. Trust us, what you learn in Vegas won't stay in Vegas – your boss will thank you!

"The sessions were informative and will prove highly useful in my job. It really motivated me and gave me new ides for projects."

Michael Cross, Niagara Regional Police Service

"The quality of speakers was excellent – they were knowledgeable and interesting. I also really liked the location – Vegas!"

Kiran P. Mody, ASP.NET programmer, Avail Technologies, Inc.

Register Today!

vslive.com/lasvegas

View the full conference agenda by scanning this QR code or visiting vslive.com/lasvegas

Use Promo Code MARAD

A Few of Our Experts and Their Sessions at Visual Studio Live! Las Vegas:

Rachel Appel - *Developer Evangelist, Microsoft*

- **CTW1** ***Chalk Talk: How Orchard CMS Works***
- **W02** ***Creating a Data Driven Web Site Using WebMatrix and ASP.NET Razor***

Andrew Brust - *Founder & CEO, Blue Badge Insights*

- **TH19** ***Microsoft's Big Play for Big Data***
- **TH15** ***Power View: Analysis and Visualization for Your Application's Data***
- **MWKS3** ***Workshop: SQL Server for Developers***

Miguel Castro - *Architect, IDesign*

- **W06** ***MVC for WebForms Developers: Comparing and Contrasting***
- **FWKS1** ***Workshop: Programming with WCF in One Day***
- **TH01** ***WPF Validation - Techniques & Styles***

Miguel de Icaza - *CTO, Xamarin*

- **CTT3** ***Visual Studio for Mobile Apps on iOS Android and WP7***

Rich Dudley - *Technical Evangelist, ComponentOne*

- **T14** ***Building Windows 8 Applications with HTML5 and jQuery***
- **T10** ***HTML5/jQuery On-Ramp***
- **TH12** ***LightSwitch Onramp***

Billy Hollis - *Next Version Systems*

- **TH05** ***Infinite Whitespace: Implementing Viewport Navigation in XAML***
- **MWKS2** ***Workshop: Creating Today's User Experiences - An Entry Point for Developers***

Rockford Lhotka - *Principal Technology Evangelist, Magenic*

- **W08** ***Building Mobile Apps with CSLA .NET***
- **T01** ***Introduction to the Windows Runtime***
- **MWKS1** ***Workshop: Full Application Lifecycle with TFS and CSLA .NET***

Leonard Lobel - *CTO, Sleek Technologies, Inc.*

- **T03** ***Introducing SQL Server Data Tools (Codenamed "Juneau")***
- **T07** ***So Many Choices, So Little Time: Understanding Your .NET 4.0 Data Access Options***
- **MWKS3** ***Workshop: SQL Server for Developers***

Brian Randell - *Senior Consultant, MCW Technologies*

- **T04** ***Application Lifecycle Management and Visual Studio: What's Next***
- **MWKS1** ***Workshop: Full Application Lifecycle with TFS and CSLA .NET***

Michael Washington - *Programmer, ADefWebserver.com*

- **TH20** ***Creating LightSwitch Control Extensions***
- **TH16** ***Incorporating LightSwitch Into Your Existing ASP.NET Applications***

Chris G. Williams - *Microsoft MVP, Principal Consultant, Magenic*

- **TH04** ***Consuming Async Web Services In Your Windows Phone Apps & Games***
- **W12** ***Making Money on Your WP7 Apps & Games with the Advertising SDK***

Keynote Speaker:

Tim Huckaby - *Chairman/Founder and CEO/Chairman, InterKnowlogy and Actus Interactive Software*

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

Register at vslive.com/lasvegas

Use Promo Code MARAD

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: Brian Harry, Microsoft Product Unit Manager for Team Foundation Server

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

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*	CTT3 **Visual Studio for Mobile Apps on iOS, Android and WP7** - *Miguel de Icaza*

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

CTW1 **Chalk Talk: How Orchard CMS Works** *Rachel Appel*	CTW2 **Chalk Talk: Parallel Programming 101** *Tiberiu Covaci*	CTW3 **Particle Swarm Optimization (PSO)** *James McCaffrey*

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 with Developer Duel

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.*

Building the Internet of Things

Torsten Grabs and Colin Miller

There's a lot of buzz about the "Internet of Things" (IoT) lately, and for good reason. Ericsson CEO Hans Vestberg estimates 50 billion devices will be connected to the Web by 2020 (bit.ly/yciS7r [PDF download]). To put this number in perspective, right now there are about 1.5 billion Web-connected PCs and fewer than 1 billion Web-connected phones—50 billion is about seven devices for every human being on the planet! Market research firm IDC in turn estimates that more than 16 billion devices will be connected to the Internet by 2015 (see **Figure 1**). Admittedly, some more conservative projections exist as well, but by anyone's numbers this represents a huge shift in the role of the Internet—from providing information and entertainment to people to supplying the connectivity for an emerging class of device-enabled applications.

The reason these large numbers are plausible is that powerful drivers—opportunity and necessity—are propelling a rapid increase in such solutions. As a recent issue of *The Economist* (economist.com/node/17388368) notes "… the bandwagon is not just rolling for the benefit of technology companies and ambitious politicians. It has gained momentum because there is a real need for such systems." **Figure 2** shows the growth of this type of solution by application area. For example, the mandated implementation of smart energy systems in Europe is being driven by necessity. We can't build the energy-generation capability required if we don't also manage energy use. On the opportunity side, the simple, ubiquitous vending machine is a good example. Once that device is connected, service personnel can be dispatched when needed rather than on some sub-optimal schedule. Prices can even be dynamically altered if there's increased local demand or if the contents are nearing expiration. Power outages can be reported to trigger replacement of perishable items immediately. In other words, connectivity enables an entirely new and more efficient business model.

Characterizing this shift by referencing the number of connected devices is certainly dramatic, but it's also somewhat misleading—this isn't about full employment for the hundreds of thousands of traditional embedded programmers. These devices are just the endpoints for complex solutions that integrate other devices into all aspects of the Internet, including analytics, the cloud, Web applications, PC and mobile interfaces, and much more.

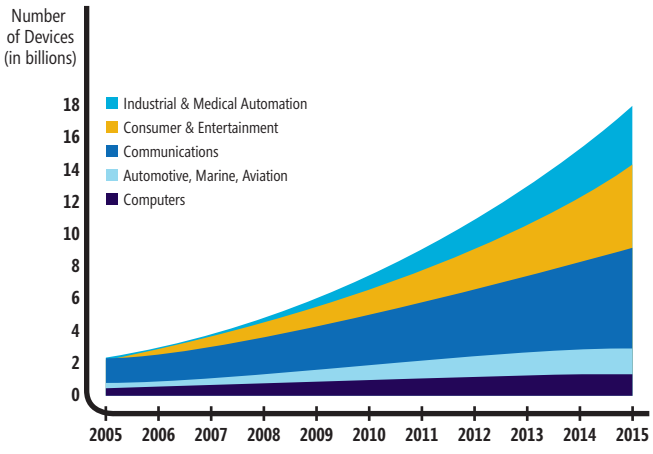
The way to look at this is that everybody currently building Web-based applications will be faced with integrating devices and helping to develop new businesses and new business models as a result. In other words, even if you're not an embedded developer and don't work in a shop that builds embedded devices, this is a very compelling opportunity for you to evaluate. Your current Microsoft skills will enable you to succeed in the IoT.

This article discusses:

- The proliferation of connected devices
- The need for near real-time analysis of data
- Creating an Internet of Things application

Technologies discussed:

.NET Micro Framework, Microsoft StreamInsight, Visual Studio



Source: John Gantz, The Embedded Internet, Methodology and Findings, IDC, January 2009

Figure 1 **Estimating the "Embedded Internet," from IDC**

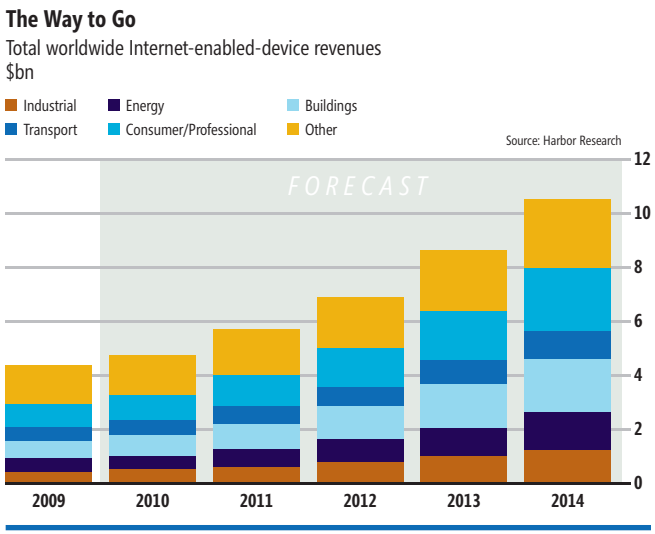


Figure 2 **Application Growth by Segment**

Running Analytics over Device Data

"Data has become the new currency," said Kevin Dallas, general manager of the Windows Embedded team in a recent interview (bit.ly/wb1Td8). Compared with current Internet applications, the IoT is about information generation, management and access. Let's compare the data characteristics of a typical Internet application today with an IoT application. You and perhaps several million others pay your bills online using a popular mechanism that's shared by many financial institutions. You log on several times a month, view some pages and submit payment information. All of this is drawn from a traditional database with queries run when you start interacting with the system. The pages you download can be large, but the interactions are very sparse, though they generate valuable information (payment information, personal information updates and so forth) that needs to be retained indefinitely.

Contrast this with an energy management system where there might be 50 million buildings (commercial and residential) providing input. The input is generated by multiple local endpoints inside, for example, a house, with a single aggregated view posted to the back end. That data includes instantaneous usage information that becomes the basis for pricing and billing, and possibly mandatory controls, back to the building. This system involves very frequent interactions with relatively low-value data that's not necessarily interesting once you compute the current state of the system and possibly the trend data for that endpoint. However, the system needs to be able to respond instantly to situations that potentially threaten it, such as demand surges that can cause grid overload and power outages. In that case, broadcasting the information might reduce energy consumption immediately. Such a system needs to continuously analyze the incoming data and compare trends over time to identify patterns that indicate a higher risk of power outages.

Figure 3 illustrates a typical architecture for IoT applications. The bottom of the stack shows various assets or devices that are instrumented with different kinds of sensors depending on the application. The sensors generally produce continuous data feeds that the application needs to process and analyze quickly. Depending on its capabilities, the device itself might be able to do some of the processing locally. This is called local analytics, and tools like the .NET Micro Framework can help you perform that local processing before the device passes on the data. IoT applications use Internet protocols to pass along the device data so that global analytics can be performed on the data. The results from global analytics, such as the overall health of a power grid, are of interest to end users who manage operations or to business decision makers. Analytics may also drive closed systems that take action automatically based on the conditions revealed in the incoming data. These approaches are particularly powerful if assets can receive feedback from the global analytics, for example, to effect a change in behavior or to improve operations. The global analytics driving these processes need to be calculated continuously and the results made available as quickly as possible. Moreover, the analytics frequently refer to time and to timestamps provided with the sensor data. Hence, just placing this kind of data in a database and running periodic queries on it is not the right approach. Fortunately, Microsoft StreamInsight allows a different approach.

Microsoft StreamInsight

Microsoft StreamInsight is designed to provide timely reactions to continuously arriving data without writing the data to disk for analysis and querying. Many IoT applications need to analyze incoming data in near-real time—right after the data is acquired from the sources. Think of that smart grid application we mentioned

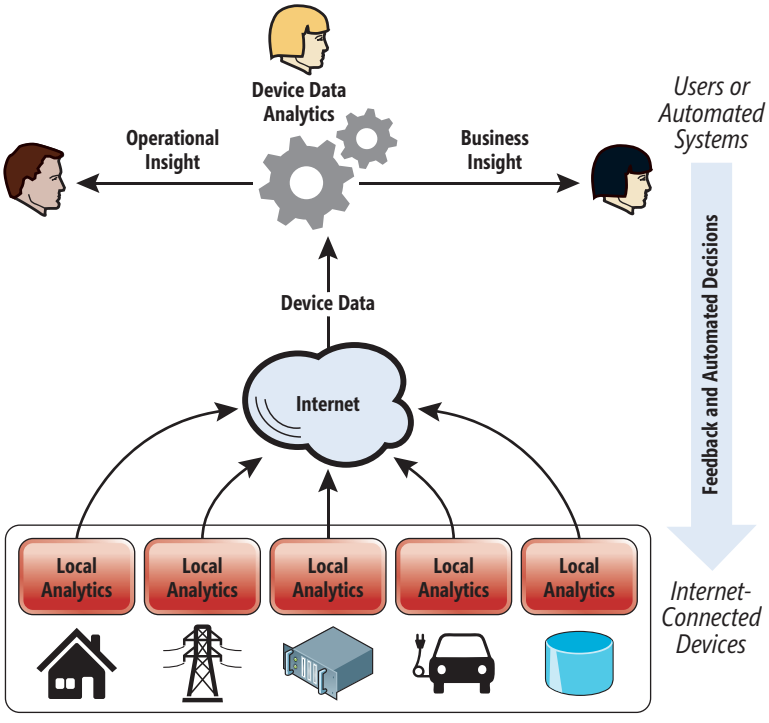


Figure 3 **Typical Architecture for Internet of Things Applications**

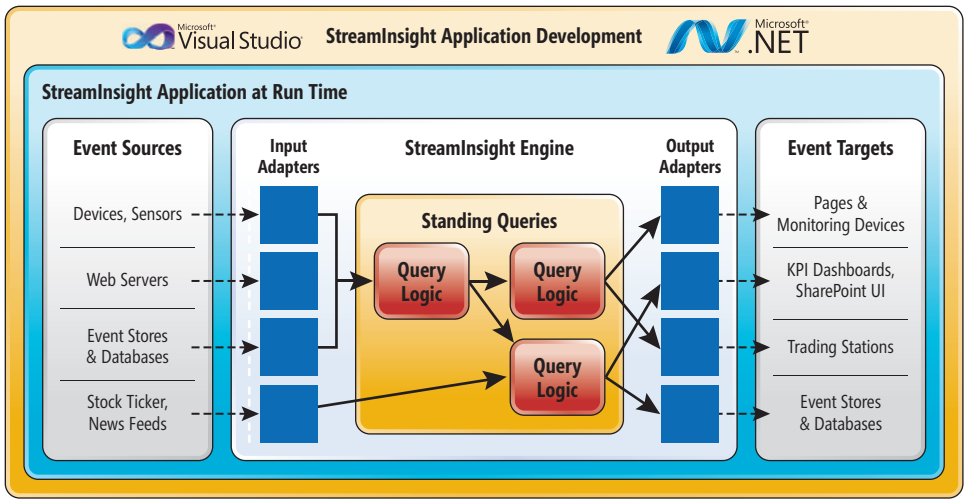


Figure 4 **StreamInsight Application Development and Runtime**

that needs to react quickly to a surge in electricity demand to rebalance the power grid. Many IoT applications have the same needs: data processing that requires continuous analysis and compelling latency. The analysis must be continuous because the data sources unceasingly produce new data. And many scenarios need to identify and react quickly to conditions that become apparent only from the analysis of the incoming data, so they require low-latency analysis with results available almost immediately. These requirements make it impractical to store the data in a relational database before performing the analysis.

We call these applications *event-driven applications*, and the IoT is just one scenario where such capabilities are highly useful. StreamInsight is a powerful platform for building these highly scalable, low-latency, event-driven applications. It's been part of Microsoft SQL Server since version 2008 R2. StreamInsight complements SQL Server with event-driven processing and rich, expressive time-based analytics. With StreamInsight, business insight is delivered at the speed the data is produced, as opposed to the speed at which traditional database reports are processed.

Analytical results that are available for human consumption right away or that enable an application to react to events automatically help businesses get a timelier and more relevant view into their operations, or even automate parts of their operations. They can also react more quickly to critical situations, opportunities or trends emerging from the sensor or device data.

To write StreamInsight applications, developers use familiar tools such as the Microsoft .NET Framework, LINQ and Microsoft Visual Studio. **Figure 4** depicts the developer and runtime experience of a StreamInsight application and introduces some of the key concepts.

Figure 5 **Submitting Sensor Data**

```
protected void submitSensorData(string uri, string payload)
{
  // Message format

  StringBuilder sb = new StringBuilder(256);

  sb.Append(
    "POST /Website/Services/DataService.aspx?method=SaveDeviceData HTTP/1.1\n");
  sb.Append("User-Agent: NetduinoPlus\n");
  sb.Append("Host: 192.168.1.101\n");
  sb.Append("Connection: Keep-Alive\n");
  sb.Append("Content-Length: ");
  sb.Append(payload.Length.ToString());
  sb.Append("\n");
  sb.Append(payload);
  sb.Append("\n");

  try
  {
    HttpResponse response = webServer.SendRequest(uri, 80, request);
  }
  catch
  {
    ...
  }
}
```

A Simple IoT Application

Let's look more in depth at a potential IoT application; then we'll build it out. For our end-to-end example, we'll focus on a simple scenario that uses motion sensors to monitor rotating equipment such as turbines or windmills. This is vital because too much vibration could point to a critical condition in which the equipment is likely to break down and cause significant damage if not stopped quickly. To detect this condition reliably, each piece of equipment has multiple sensors tracking motion. A surge in motion from a single sensor might just indicate unreliable data readings from that sensor, but abnormally high motion from multiple sensors at the same time is considered a critical condition. For a large turbine, for example, you'd probably want to raise an alarm or even shut down the equipment automatically. Besides checking continuously for such conditions, we also want to give operators a dashboard that provides a near-real-time view of equipment status.

To build out this scenario, we need to address the following requirements and technical challenges:

- What data does the device need to capture?
- What sensors do we use to measure it?
- How does the device communicate its sensor readings to the Internet?
- How do we collect the device data in one place for analytics?
- How can we continuously analyze the incoming data and react quickly to critical conditions?
- How do we correlate sensor readings in time across multiple devices so we can check for global conditions?

Let's take a look at how we address these requirements and implement the end-to-end scenario.

An IoT Application: Implementation Highlights

Here are some of the key steps to implement an IoT application as outlined in the previous section. We'll discuss the device first,

Figure 6 **Populating the DeviceData Object**

```
private int SaveDeviceData()
{
...
  List<string> data = record.Split(',').ToList();
  DeviceData deviceData = new DeviceData();

  deviceData.MAC = NormalizeMAC(data[0].Trim());
  deviceData.DateTime = DateTime.UtcNow;
...
  deviceData.Motion = Convert.ToDecimal(data[2].Substring(data[2].
IndexOf(":") + 1));
...

  // Communicate each new device data record to StreamInsight
  DeviceDataStreaming streaming = (DeviceDataStreaming)
    HttpContext.Current.Application[Global.StreamingIdentifier];
    streaming.TrackDeviceData(deviceData);
...
}
```

jump to the visualization of the output and then move to the analytics across devices that populate the dashboard.

The Device To build a sensor device, we started with the Netduino Plus, a popular small board with 128K SRAM that runs the .NET Micro Framework. We added a common hobbyist Wi-Fi radio called the WiFly GSX Breakout and mounted the actual sensors, including a three-axis accelerometer, on a custom PCB board. We programmed the device to send an update of the sensor readings every second to a Web service that acts as the hub to collect and process the data from all devices.

We're using a RESTful connection with the Web service—just an HTTP POST containing comma-separated name-value pairs. Of course, you can do this from any kind of device that supports HTTP. We opted to use the .NET Micro Framework so that the entire application—including the devices, the Web service, the StreamInsight adapters, the Silverlight dashboard and so forth—could all be written using a single programming model (.NET) and tool chain (Visual Studio). Clearly, if you have .NET skills, you don't need to hire new staff or farm out parts of your IoT project to an external embedded shop; you have the skills to do it all. For example, setting up the accelerometer requires only a few lines of code to access the AnalogInput class and call the Read method:

```
this.analogInputX = new AnalogInput(pinX);
this.analogInputY = new AnalogInput(pinY);
this.analogInputZ = new AnalogInput(pinZ);

...

rawZ = analogInputZ.Read();
rawY = analogInputY.Read();
rawX = analogInputX.Read();
```

Once the sensor input is read and the HTTP message content formatted, all that's required to send it is included in **Figure 5**.

On the server side, we implement the method SaveDeviceData, to which the devices POST their messages. We split the message string and parse the MAC address, the timestamp and payload data, such as the motion readings from the accelerometer. We use all of this to populate a DeviceData object (see **Figure 6**) that we pass to StreamInsight for subsequent analysis.

The Dashboard Now we want to build a dashboard that lets the equipment operator view the current status of the sensors on the equipment. For ease of presentation, we'll focus on just a single piece of equipment. **Figure 7** shows an example of such a dashboard. Let's start on the left and look at the different views of the sensor data.

Moving Averages View: The data grid on the bottom left shows the sensor readings from the device with values for light, temperature and motion, as well as the device ID and a timestamp. As you can see from the timestamps, these values are updated every second. But, instead of displaying the raw sensor values, the dashboard shows moving averages over 10 seconds worth of sensor data. This means every second the values are updated with the average of the last 10 seconds worth of data. Using a moving average is a common, simple technique to protect against the effect of outliers and bad data that occur occasionally with low-cost sensors.

Trendline View: On the lower right, the dashboard shows the trendlines for the sensors. The trendline view is driven by the moving averages shown in the data grid on the left.

Alarm View: The view at the upper right displays a data grid for alarms. If a critical condition is detected, an alarm is raised that shows the time and additional information such as severity and status.

Analytics Now let's take a look behind the scenes and discuss the analytics that are processing the incoming sensor data and calculating the results the dashboard visualizes. We use StreamInsight to do the analytics. The following class represents the device data, including the MAC address, a timestamp and the sensor values:

```
public class DeviceData
{
  public string MAC { get; set; }
  public DateTime DateTime { get; set; }
  public decimal? Light { get; set; }
  public decimal? Temperature { get; set; }
  public decimal? Motion { get; set; }
}
```

This class defines the shape of a single event, but we want to start reasoning about many events. To do this, we define an Observable data source for StreamInsight. This is simply a data source that implements the System.IObservable interface:

```
public class DeviceDataObservable : IObservable<DeviceData>
  {
    ...
  }
```

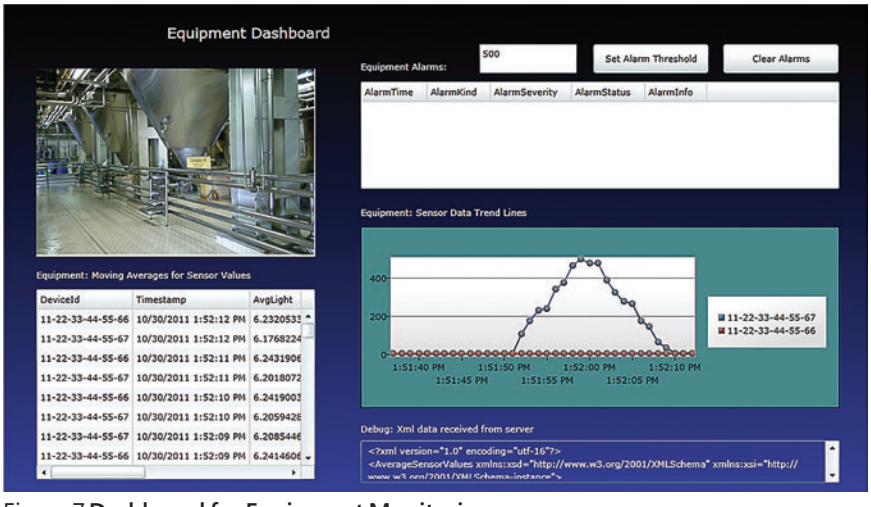


Figure 7 **Dashboard for Equipment Monitoring**

Apprenda /a'pren'da/: An Open Platform as a Service (PaaS) stack for .NET

Deploy .NET Apps in minutes.
A PaaS solution in a class of its own.

www.apprenda.com/makethegrade

apprenda®
apprenda.com

Figure 8 **Getting the Moving Averages**

```
public static CepStream<AverageSensorValues> GroupedAverages(
             Application application,
             DeviceDataObservable source)
  {
    var q1 = from e1 in source.ToPointStream(application,
      e => PointEvent.CreateInsert(
        new DateTimeOffset(
          e.DateTime.ToUniversalTime()),e),
      AdvanceTimeSettings.StrictlyIncreasingStartTime,
      "Device Data Input Stream")
           select e1;

    var q2 = from e2 in q1
             group e2 by e2.MAC into groups
             from w in groups.HoppingWindow(
               TimeSpan.FromSeconds(10),
               TimeSpan.FromSeconds(1))
             select new AverageSensorValues
             {
               DeviceId = groups.Key,
               Timestamp = null,
               AvgTemperature = w.Avg(t => t.Temperature),
               AvgLight = w.Avg(t => t.Light),
               AvgMotion = w.Avg(t => t.Motion)
             };

    return q2;
  }
```

Once you have a .NET Framework sequence like an Enumerable or an Observable like this, you can start writing StreamInsight queries over these collections. Let's take a quick look at some of the key queries. The first one takes the Observable as an input and produces a stream of StreamInsight point events, using the DateTime field in the device data as the timestamp for the StreamInsight event. We take this stream as input in the next LINQ statement and group the data by the MAC address. For each group, we then apply a hopping window (a time-based subset of events) with a window size of 10 seconds and let the window recalculate every second. Within each window, we calculate the averages for temperature, light and motion. That gives us a moving average per device that recalculates every second. **Figure 8** shows the code for this wrapped in a function that returns the result as a stream of StreamInsight events.

This is a great place to think about implementing the alarm query. Remember, the alarm is to be triggered when multiple motion sensors move above the motion threshold at the same time. We can handle this with just a couple of StreamInsight LINQ statements over the grouped averages just calculated. The first query, q3, applies a neat trick by representing changes of the alarm threshold as a stream of events called AlarmThresholdSignal. The query joins the thresholds with the averages stream from the previous query and then just filters for the events above the threshold:

```
var q3 = from sensor in GroupedAverages(application, source)
         from refdata in AlarmThresholdSignal(application, alarmsthresholds)
         where (sensor.AvgMotion !=
           null && (double) sensor.AvgMotion > refdata.Threshold)
         select new
         {
           AlarmDevice = sensor.DeviceId,
           AlarmInfo = "This is a test alarm for a single device",
         };
```

The next query uses the StreamInsight snapshot window to identify points in time when event statuses change. If a new event results from the previous filter query, this is a new snapshot and the snapshot operation produces a new window containing all events that coincide or overlap with the event that triggered the snapshot window. The following code counts events above the alarm threshold when the snapshot window is created:

```
var alarmcount = from win in q3.SnapshotWindow()
                 select new
                 {
                   AlarmCount = win.Count()
                 };
```

The final step checks whether the count shows that multiple devices are indicating alarms:

```
var filteralarms = from f in alarmcount
                   where f.AlarmCount >= 2
                   select new AlarmEvent
                   {
                     AlarmTime = null,
                     AlarmInfo = "Now we have an alarm across multiple devices",
                     AlarmKind = 0,
                     AlarmSeverity = 10,
                     AlarmStatus = 1
                   };
```

Now we just need to get the output streams with the average sensor values and alarms from StreamInsight to the UI.

Getting the Stream to the UI

With StreamInsight producing the result streams on the server side, we need a way to communicate these streams to the consumers. Consumers probably won't run in the server process and might use a lightweight Web application to visualize the results. If you're using Silverlight, the duplex protocol is convenient because it supports continuous push-based delivery from the server to the client. HTML5 Web sockets are a compelling alternative, too. In any case, you want to make it easy to add new analytics on the server side and be able to easily wire them up with the UI—without tearing apart the client-server interfaces between the UI and the process hosting StreamInsight. If your load between UI and server

Figure 9 **Annotating the Event Structures**

```
[DataContract]
public class AverageSensorValues : BaseEvent
{
  [DataMember]
  public new static Guid TypeGuid =
    Guid.Parse("{F67ECF8B-489F-418F-A01A-43B606C623AC}");
  public override Guid GetTypeGuid() { return TypeGuid; }

  [DataMember]
  public string DeviceId { get; set; }
  [DataMember]
  public DateTime? Timestamp { get; set; }
  [DataMember]
  public decimal? AvgLight { get; set; }
  [DataMember]
  public decimal? AvgTemperature { get; set; }
  [DataMember]
  public decimal? AvgMotion { get; set; }
}
```

Figure 10 **Sending Result Events from the Server**

```
static public void CallClient<T>(T eventData) where T : BaseEvent
  {
    if (null != client)
    {
      var xmlSerializer = new XmlSerializer(typeof(T));
      var stringBuilder = new StringBuilder();
      var stringWriter = new StringWriter(stringBuilder);
      xmlSerializer.Serialize(stringWriter, eventData);

      client.Receive(stringBuilder.ToString(), eventData.GetTypeGuid());
    }
  }
```

5 YEARS OF EXCELLENCE

XCEED
DataGrid
for WPF

Mature, feature-packed, and lightning-fast. The most adopted and trusted WPF datagrid.

IBM®
U2 SystemBuilder™

"IBM U2 researched existing third-party solutions and identified Xceed DataGrid for WPF as the most suitable tool. The datagrid's performance and solid foundation take true advantage of WPF and provide the extensibility required for IBM U2 customers to take their applications to the next level in presentation design and styling."

Vincent Smith
U2 Tools Product Manager at IBM

Microsoft®
Visual Studio® Team System 2010

"Using Xceed DataGrid for WPF in Microsoft Visual Studio System 2010 helped us greatly reduce the time and resources necessary for developing all the data presentation feature we needed. Working with Xceed has been a pleasure."

Norman Guadagno
Director of Product Marketing
for Microsoft Visual Studio Team System

XCEED Professional Themes for WPF

Theme your entire app in minutes. Flawless styles for all official WPF controls.

XCEED DataGrid for Silverlight

Incredible streaming technology. Speed up your app and say goodbye to paging.

XCEED Ultimate ListBox for WPF

NEW

The world's first streaming listbox. Simple, drop-in upgrade to the WPF listbox.

XCEED Ultimate ListBox for Silverlight

NEW

Fast and fluid, with ground-breaking streaming technology.

Visual Studio PARTNER

XCEED MULTI-TALENTED COMPONENTS

Figure 11 **Receiving and Deserializing the Event on the Client**

```
void proxy_ReceiveReceived(object sender, ReceiveReceivedEventArgs e)
{
  if (e.Error == null)
  {
    if (AverageSensorValues.TypeGuid == e.guid)
    {
      ProcessAverageSensorValues(Deserialize<AverageSensorValues>(e.eventData));
    }
    else if (AlarmEvent.TypeGuid == e.guid)
    {
      ProcessAlarms(Deserialize<AlarmEvent>(e.eventData));
    }
    else
    {
      ProcessUnknown();
    }
  }
}
```

is moderate, you can serialize the results on the server side into XML and deserialize them on the client side. That way, you only need to worry about XML across the wire and in your client-server interfaces, plus an additional cookie to indicate what types to expect for deserialization. Here are a couple of the key pieces of code.

The first code snippet is the Windows Communication Foundation contract for flowing the event data as an XML-serialized string, along with a GUID to indicate the type:

```
[ServiceContract]
public interface IDuplexClient
{
  [OperationContract(IsOneWay = true)]
  void Receive(string eventData, Guid guid);
}
```

Now we can annotate our result event structures with a data contract to make them serializable, as shown in **Figure 9**.

We can now easily serialize result events on the server side and communicate them to the client, as **Figure 10** shows.

On the client side, we deserialize the event in the callback method for the duplex service and then branch into different methods based on the type of event received, as shown in **Figure 11**.

With these queries and the communication to the Web application in place, you can now pick up several devices and shake them until some are above your alarm threshold. The UI will then produce one of those nice red alarms like the one shown in **Figure 12**.

Because new data is constantly coming in with a near-real-time dashboard, ObservableCollections are extremely useful for updating your UI. If you base your data grids and trendlines on these Observable collections, you don't need to worry about the updating part in your code. The collections are doing this for you automatically behind the scenes.

The Outlook

In this implementation, the devices communicate with a regular Web service that could be running on an ordinary PC connected to the Internet. But cloud computing is an attractive alternative; you don't necessarily need to own the hardware and run the software for your own Web server. Instead, a service in the cloud can serve as the hub where all the device data is being collected for your application. This also makes it very easy for you to elastically scale your processing power as the number of devices grows or you deploy additional analytics over the device data. Microsoft is planning to provide StreamInsight capabilities as a service in Windows Azure (StreamInsight Project Codename "Austin"). By providing predefined communication endpoints and protocols, Austin will make it easy to connect your devices to rich analytical processing capabilities in the Microsoft cloud. If you deploy your IoT applications in Windows Azure, you'll automatically get the cloud benefits of elastic scale and pay-as-you go to manage device connections and to perform rich analytics over device data.

Another important shift is happening with the recent standardization effort from the W3C. The most important initiatives for IoT applications are HTML5 and Web sockets. HTML5 provides the platform for rich Web applications such as the dashboard we implemented. WebSocket in turn simplifies full-duplex communication between the browser and the Web server over TCP, in particular for the push model of results delivery that the continuous processing of sensor data requires.

Connected devices are opening up an exciting new world of applications, and the tools for building these IoT applications are available from Microsoft today. Here we have shown how you can use your .NET Framework skills at the device level, using familiar interfaces, and feed data through Web services into the powerful analytics capabilities of StreamInsight. Get started building your IoT applications using connected devices now! ■

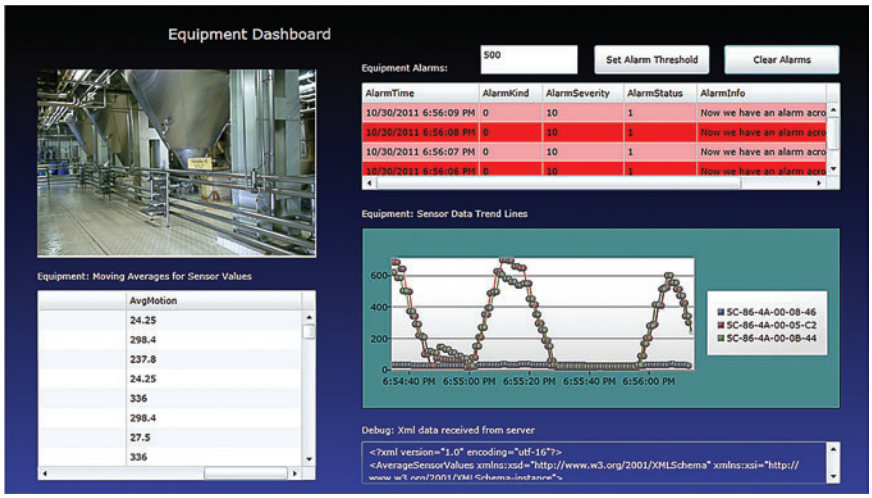


Figure 12 **Equipment Dashboard with Alarms**

TORSTEN GRABS *is a lead program manager in the Microsoft SQL Server division. He has more than 10 years' experience working with Microsoft SQL Server products and holds a Ph.D. in computer science from the Swiss Federal Institute of Technology, Zurich, Switzerland.*

COLIN MILLER *has worked for 25 years (including 15 at Microsoft) on PC software, including databases, desktop publishing, consumer products, Word, Internet Explorer, Passport (LiveID) and online services. He's the product unit manager of the .NET Micro Framework.*

THANKS to the following technical experts for reviewing this article: *Rafael Fernandez Moctezuma and Lorenzo Tessiore*

NEW RELEASE

GrapeCity PowerSuite | from $1,959.02

GrapeCity PowerTools

Enterprise-ready .NET control suite for building world-class Business Applications. Includes:

- **ActiveReports** – fast, flexible and robust reporting engine
- **Spread** – full-featured Excel platform providing a complete, familiar spreadsheet experience
- **ActiveAnalysis** – interactive data analysis control providing pivoting, drill-down, drill-up, etc
- **MultiRow** – innovative Grid component allowing complete data layout customization
- **ActiveChart** – chart control providing enhanced graphical presentation of data, and more

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

BEST SELLER

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

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

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

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

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

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

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

GSA Schedule Contract GS-35F-0188R

Develop Hybrid Native and Mobile Web Apps

Shane Church

You want to build a mobile application but you're bewildered by the array of available devices and APIs to learn. Which mobile platform should you choose? The Apple iOS (iPhone and iPad) uses Objective C, Google Android uses Java and Windows Phone uses Silverlight, yet each one of these options has a distinct API and a distinct market. Choosing to focus on one particular technology stack could leave 50 percent of the market—or more—unable to use your application. If you choose to try to support all of these platforms, you have at least three distinct codebases to maintain, significantly increasing your development and maintenance costs.

There is another option: You could build a mobile Web application, because it can be viewed on any of these devices. But this approach also has some hurdles. The biggest obstacle for developing an entire business application using HTML and JavaScript is the lack of access to many native device hardware features such as the camera, GPS or accelerometer.

Clearly the mobile market is only going to grow, so how do you support all of these device options while providing the best user experience possible? In this article, I'll show you how to build a mobile application that takes advantage of the best of both worlds by wrapping a mobile Web application with a native application shell.

This article discusses:

- The hybrid application concept
- Building the Web application
- Using jQuery Mobile
- Native mobile application shells
- Targeting Windows Phone, Android and iOS
- Graceful degradation

Technologies discussed:

ASP.NET MVC 3, Windows Phone

Code download available at:

code.msdn.microsoft.com/mag201203MobileWeb

The Hybrid Application Concept

The basic concept of a hybrid application is to wrap a mobile-optimized Web application in a device-specific native application shell. The native application shell hosts a Web browser control that's configured to launch the specific mobile application URL when the shell application launches. Other UI elements can be provided in the native application shell as needed, but only the Web browser control is required. The native Web browser control then listens to the URLs being requested as the user navigates the site. When the user requests a specific URL that requires native functionality, the Web browser control interrupts the navigation event and instead invokes the native functionality. As the user completes the native process, the application navigates the Web browser control back into the Web site flow in the appropriate location.

To illustrate how this is done, I'll walk through my experience building an application with my EffectiveUI colleagues for a client. Built for a mobile field worker who processes a number of maintenance

work orders for municipal assets such as signs, benches and fire hydrants, the application takes advantage of browser-supported features to get the user's current location, and native hardware access to take pictures of assets and upload them to the server. **Figure 1** shows the main menu of the completed application.

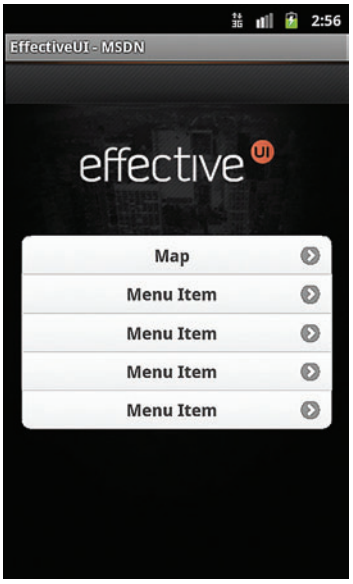


Figure 1 **The Completed Application Main Menu**

Building the Web Application

When building this mobile application, I followed a number of suggestions from Steve Sanderson's article, "Build a Better Mobile Browsing Experience" (msdn.microsoft.com/magazine/hh288079) in the July 2011 issue of *MSDN Magazine*. In addition to the recommendations in this article, I learned a few things along the way:

- **Optimize UI Elements for Touch** Most mobile users are using touch-based interaction. Touch interaction is inherently less precise than mouse-based interaction on the desktop. All interactive elements such as buttons and menu items need to be proportionally larger in the mobile interface than they are in the desktop experience.

> The basic concept of a hybrid application is to wrap a mobile-optimized Web application in a device-specific native application shell.

- **Optimize Your Mobile Views for Bandwidth** Most mobile devices are resource constrained, especially when considering bandwidth. Don't force your user to download a number of large images in order to use your site. Users on mobile devices expect responsive interfaces and will quickly abandon your site or application if it doesn't perform to their expectations.
- **Use HTML5 and CSS3** Because mobile Web browsers don't have the long legacy of desktop browsers, they're much quicker to adopt the emerging HTML5 and CSS3 standards than their desktop counterparts. In many cases, mobile browsers are far ahead of desktop browsers in implementing these features. Take advantage of this in your mobile views to lighten the payload that the mobile browser needs to download and let the browser do more of the stylistic rendering.

One of the technical requirements from my client when creating this application was to demonstrate sharing controller logic between desktop and mobile views of the site. This requirement is common to many customers and should be favored by developers as well, as it greatly streamlines the process of building an application that supports both desktop and mobile users. ASP.NET MVC 3 provides the ability to switch views based on request elements, such as the requesting browser, while still sharing controllers and models between multiple views. It also allows the developer to finely control the experience on the site for each of the different platforms, meaning the developer only needs to build the business logic once and then tailor the presentation for each platform. **Figure 2** shows a utility function to decide which view to present.

The utility function allowed me to meet the requirement to share the same code for making the decision about which view to present to the user based on the incoming request. If the incoming request is a script that's requesting JSON instead of HTML, the controller can also respond appropriately using the same business logic and model classes by simply setting the outputType parameter appropriately. I also use a precompiler statement looking for the MOBILE conditional compilation symbol to enable debugging the mobile views using my desktop browsers. This was enabled using an additional build target, "Mobile," in the ASP.NET MVC 3 project, and it allowed me to skip the check for Request.Browser.IsMobileDevice in the desktop debugging configuration, greatly improving my efficiency in building and debugging the mobile version of the application.

While building the application I also used separate master pages for the mobile and desktop versions of the site. The desktop and mobile versions of the master page are significantly different to address the disparities in presentation between the platforms. The mobile master page includes my mobile-specific CSS files and a simplified layout structure to ease the development of individual views using the jQuery Mobile Framework markup and syntax.

Figure 2 **Utility for Deciding Which View to Present**

```
private ActionResult SelectView(string viewName, object model,
  string outputType = "html")
{
  if (outputType.ToLower() == "json")
  {
    return Json(model, JsonRequestBehavior.AllowGet);
  }
  else
  {
    #if MOBILE
      return View(viewName + "Mobile", model);
    #else
      if (Request.Browser.IsMobileDevice)
      {
        return View(viewName + "Mobile", model);
      }
      else
      {
        return View(viewName, model);
      }
    #endif
  }
}
```

Figure 3 **Geolocation Using HTML5**

```
if(navigator.geolocation) {
  navigator.geolocation.getCurrentPosition(function (position) {
    $("#map_canvas").GoogleMap("addMarker", {
      id: "device_location",
      latitude: position.coords.latitude,
      longitude: position.coords.longitude,
      description: "Current Location",
      iconImageUrl: '@Url.Content("~/Content/images/my-location-dot.png")',
      callback: function () {
        $("#map_canvas").GoogleMap("panToLocation", {
          latitude: position.coords.latitude,
          longitude: position.coords.longitude
        });
      }
    });
  }, function (error) {

  }, {
    enableHighAccuracy: true
  });
}
```

All of the modern mobile platforms allow access to a device's GPS radio to determine the user's current location through the HTML5 World Wide Web Consortium (W3C) geolocation APIs. The use of the geolocation APIs was discussed in detail in Brandon Satrom's article, "Integrating Geolocation into Web Applications" (msdn.microsoft.com/magazine/hh580735) in the December 2011 issue. Although that article discusses using an HTML5 JavaScript polyfill to support location on browsers that don't natively support the HTML5 geolocation APIs, most current mobile browsers support the HTML5 geolocation APIs natively, so the polyfill technique most likely isn't necessary. You should consider the devices and browsers that you're targeting while you're evaluating the necessity of using the polyfill technique. One thing to note specifically for Android is that you'll need to make sure the enableHighAccuracy parameter in the geolocation call is set to "true" in order to successfully access the GPS functionality in the Android emulator, as shown in **Figure 3**.

Figure 4 **The Mobile Master Page**

```
<!DOCTYPE html>
<html>
<head>
  <title>@ViewBag.Title</title>
  <meta name="viewport" content="width=device-width,
    initial-scale=1.0, user-scalable=no, height=device-height" />
  <meta http-equiv="Content-type" content="text/html; charset=utf-8">
  <link href="@Url.Content("~/Content/eui_assets/css/reset.css")"
    rel="stylesheet" type="text/css" />
  <link href="@Url.Content("~/Content/jquery.mobile-1.0.min.css")"
    rel="stylesheet" type="text/css" />
  <link href="@Url.Content("~/Content/mobile.css")"
    rel="stylesheet" type="text/css" />

  <script src="@Url.Content("~/Scripts/jquery-1.7.1.min.js")"
    type="text/javascript"></script>
  @RenderSection("PreJQueryMobileInit", false)
  <script src="@Url.Content("~/Scripts/jquery.mobile-1.0.min.js")"
    type="text/javascript"></script>
  <script type="text/javascript">
    $('a[data-ajax="false"]').live('click', function (event) {
      if (!$(this).hasClass("camera-link")) {
        $.mobile.showPageLoadingMsg();
      }
    });
  </script>
  @RenderSection("Head", false)
</head>
<body class="eui_body" id="@ViewBag.BodyID">
  @RenderBody()
</body>
</html>
```

Figure 5 **Binding to the mobileinit Event**

```
@section PreJQueryMobileInit {
  <script type="text/javascript">
    $(document).bind("mobileinit", function () {
      $.mobile.listview.prototype.options.filterCompleteCallback = function () {
        // Note that filtercompletecallback is a custom
        // addition to jQuery Mobile and would need to be updated
        // in future revisions.
        // See comments in jquery.mobile-1.0.js with SSC 09/12/2011
        var ids = [];
        var $visibleItems = $("#js-work-orders-list").find(
          "li:not(.ui-screen-hidden)");
        for (var i = 0; i < $visibleItems.length; i++) {
          var item = $($visibleItems[i]).find("p");
          ids.push(item.text().substr(item.text().indexOf('#') + 1));
        }
        ids.push("device_location");
        $("#map_canvas").GoogleMap("hideAllMarkersExceptList", ids);
      }
    });
  </script>
}
```

Using jQuery Mobile

The jQuery Mobile Framework is "a unified HTML5-based user interface system for all popular mobile device platforms," according to the project's Web site (jquerymobile.com). It contains a number of touch-optimized widgets and greatly eases the task of building mobile Web applications that look and feel like native mobile applications. jQuery Mobile can be added to your ASP.NET MVC 3 project through NuGet using the NuGet Package Manager interface or from the Package Manager Console by running the command "Install-Package jquery.mobile." This adds the jQuery Mobile JavaScript and CSS files to your project. You'll still need to add references to the jQuery Mobile JavaScript and CSS files to your mobile master page, as shown in **Figure 4**.

jQuery Mobile does make some significant modifications to the patterns with which any jQuery developer is familiar. To quote the jQuery Mobile documentation:

> *The first thing you learn in jQuery is to call code inside the $(document).ready() function so everything will execute as soon as the DOM is loaded. However, in jQuery Mobile, [AJAX] is used to load the contents of each page into the DOM as you navigate, and the DOM ready handler only executes for the first page. To execute code whenever a new page is loaded and created, you can bind to the pageinit event.*

I used the pageinit event inside of all the pages in the application that contained the Google Maps control in order to initialize the map when the page is transitioned into view via AJAX.

An additional feature of the mobile master page is the @RenderSection("PreJQueryMobileInit", false) line, shown in **Figure 4**, which allows you to execute scripts before jQuery Mobile is initialized on the page. In the example application I used this feature to bind to the mobileinit event so I could set up a custom callback when the jQuery Mobile listview filter behavior is complete. I also added two lines of code to the jQuery Mobile library to add a filterCompleteCallback method to the listview prototype in order

Figure 6 **MapMobile.cshtml Markup**

```
<div data-role="page" id="map_page" data-fullscreen="true"
  data-url="map_page" data-theme="a">
  <header data-role="header" data-position="fixed">
    <a href="@Url.Action("Index", "Home")" data-icon="home"
      data-direction="reverse">Home</a>
    <h1>Map Demo</h1>
    <a href="#" data-icon="back" id="js-exit-street-view"
      class="ui-btn-hidden">Exit Street View</a>
  </header>
  <div data-role="content" class="main-content">
    <div id="map_canvas" style="width:100%;height:100%"></div>
  </div>
  <footer data-role="footer" data-position="fixed"
    data-id="fixed-nav" data-theme="a">
    <nav data-role="navbar">
      <ul>
        <li><a href="#map_page" class="ui-btn-active
          ui-state-persist">Map</a></li>
        <li><a href="#items_page">Work Orders</a></li>
      </ul>
    </nav>
  </footer>
</div>
<div data-role="page" id="items_page" data-url="items_page" data-theme="a">
  <header data-role="header" data-position="fixed">
    <a href="@Url.Action("Index", "Home")" data-icon="home"
      data-direction="reverse">Home</a>
    <h1>Map Demo</h1>
  </header>
  <div data-role="content" class="main-content">
    <div class="list-container">
      <ul data-role="listview" id="js-work-orders-list" data-filter="true">
      @foreach (MapItem item in Model.Items)
  {
      <li class="work-order-id-@item.ID">
        <a href="@Url.Action("Details", "Home", new { id = item.ID })"
          data-ajax="false">
          <h3>@item.Issue</h3>
          <p>Work Order #@item.ID</p>
        </a>
      </li>
    }
      </ul>
    </div>
  </div>
  <footer data-role="footer" data-position="fixed"
    data-id="fixed-nav" data-theme="a">
    <nav data-role="navbar">
      <ul>
        <li><a href="#map_page" data-direction="reverse">Map</a></li>
        <li><a href="#items_page" class="ui-btn-active
          ui-state-persist">Work Orders</a></li>
      </ul>
    </nav>
  </footer>
</div>
```

to get a notification when the built-in list filtering was complete. This allowed me to refresh the matched items on the map to match the filtered list. The callback function needed to be added to the jQuery Mobile listview before jQuery Mobile was applied to any of the markup; that code is executed in the mobileinit event handler shown in **Figure 5**.

jQuery Mobile takes significant advantage of new features in HTML5 such as the header and footer tags and the data-* attributes. The data-role attributes determine the behavior that should be attached to a given element. For example, in the MapMobile.cshtml view in **Figure 6**, I have two divs defined with the data-role="page" attribute.

This attribute tells jQuery Mobile that each of these divs should be treated as a separate page on the mobile device and to transition between them using AJAX without a page navigation occurring in the browser. This produces the effect shown in the screenshots in **Figure 7**. The jQuery Mobile Web site provides recommendations and more details on how to use each of the data-* attributes in the jQuery Mobile context.

Building the Native Mobile Application Shells

The basic pattern in developing each of the native application shells is designing an application that simply contains a full-screen Web browser control. Within this control I capture the event that's fired when the user requests a new page and compare the requested URL against a list of known URLs that should invoke native functionality. This is where the "magic" of a Web-based application in a native application shell happens. For the purposes of this application, the URL that I matched within the site is "Home/Image" to invoke the native camera functionality. When the user is on the Work Order details page, he'll see a camera icon in the upper-right corner of the screen as illustrated in **Figure 8**. Clicking on this icon invokes the native camera.

Windows Phone

Windows Phone uses Silverlight for all of the native functionality. In some ways, this makes Windows Phone the easiest platform to support for the mobile Web developer who is familiar with ASP.NET. The basic XAML layout for the native application shell is simple, as shown here:

```
<Canvas x:Name="LayoutRoot" Background="Black" Margin="0">
  <phone:WebBrowser HorizontalAlignment="Left" Name="webBrowser1"
    Navigating="webBrowser1_Navigating" IsScriptEnabled="True"
    IsGeolocationEnabled="True"
    Background="Black" Height="720" Width="480" />
</Canvas>
```

The key items to note here are that IsScriptEnabled is set to true—because, by default, the Web browser control in Windows Phone doesn't enable script—and that I'm handling the Navigating event.

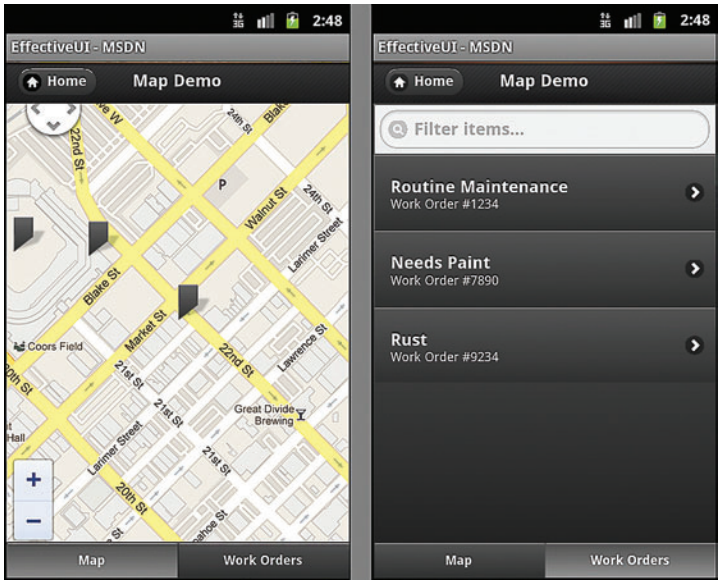


Figure 7 **Transitioning Between Pages Via AJAX**

In the MainPage.xaml.cs, shown in **Figure 9**, I handle the webBrowser1_Navigating event. If the navigation URL matches the URL that I'm looking for, I pick out the ID of the work order that I'm working with and invoke the native CameraCaptureTask while cancelling the Web browser navigation. After the user takes the picture with the camera, the photoCaptureOrSelectionCompleted method is invoked. Here, I upload the picture to the Web server using the same HTTP form POST action that the Web site would be using if I submitted a form that contained a file upload input button. When the photo upload completes, upload_FormUploadCompleted is invoked, returning the user to the Web application flow.

Windows Phone has some different behaviors when interacting with the Web-based version of the Google Maps or Bing Maps controls when compared with Android or iOS. Because of the way the Internet Explorer 9 mobile browser captures touch, swipe and pinch gestures without passing them through to the JavaScript engine, the Web-based maps can't zoom or pan with gestures and must use the zoom or pan controls provided by the map. Given this limitation, a future enhancement to this project would be to invoke the native Bing Maps control on Windows Phone where interactive map functionality is required and then return to the Web application on screens that didn't require interactive map functionality.

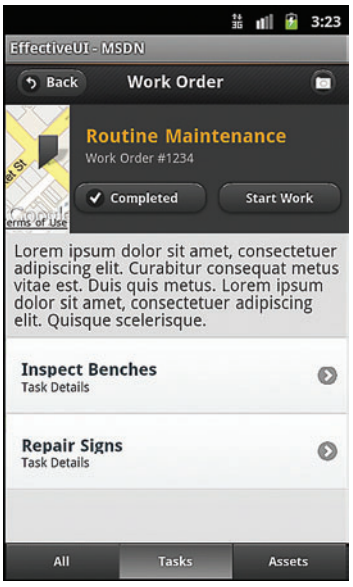


Figure 8 **Invoking Native Camera Functionality**

Android

The Java code for Android was written by my colleague, Sean Christmann, and is similar to the code for Windows Phone. The following layout XML defines a full-screen layout for the Android WebView control:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.
com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <WebView android:id="@+id/webView"
      android:layout_width="match_parent"
      android:layout_height="match_parent"></WebView>
</LinearLayout>
```

Within EffectiveUIActivity.java, shown in **Figure 10**, the override for onCreate sets up the WebViewClient to override the onLoadResource and shouldOverrideUrlLoading methods of the WebView control to search for the same matching string as used in Windows Phone and, if found, creates the camera activity and cancels the navigation. The code also overrides onGeolocationPermissionsShowPrompt to suppress the prompt to the user that would occur each time the application is run to allow permission for the WebView control to access the GPS location. After the camera activity is executed, the onActivityResult function posts the picture to the Web server using the same method as the earlier Windows Phone example and then returns the user to the Web application flow.

iOS

The Objective-C code for iOS was also written by my colleague, Sean Christmann, and is also similar to that used for Windows Phone and Android. Within WebCameraViewController.m shown in **Figure 11**, the UIWebView control executes the shouldStartLoadWithRequest method to do the pattern matching on the requested

Figure 9 **Windows Phone MainPage.xaml.cs**

```
public partial class MainPage : PhoneApplicationPage
{
  CameraCaptureTask cameraCaptureTask;
  BitmapImage bmp;
  string id = "";
  string baseURL = "http://...";

  // Constructor
  public MainPage()
  {
    InitializeComponent();
    cameraCaptureTask = new CameraCaptureTask();
    cameraCaptureTask.Completed +=
      new EventHandler<PhotoResult>(photoCaptureOrSelectionCompleted);
  }

  private void webBrowser1_Navigating(object sender, NavigatingEventArgs e)
  {
    // Catch Navigation and launch local camera
    if (e.Uri.AbsoluteUri.ToLower().Contains("home/image"))
    {
      id = e.Uri.AbsoluteUri.Substring(e.Uri.AbsoluteUri.LastIndexOf("/") + 1);
      cameraCaptureTask.Show();
      e.Cancel = true;
    }
  }

  void photoCaptureOrSelectionCompleted(object sender, PhotoResult e)
  {
    if (e.TaskResult == TaskResult.OK)
    {
      byte[] data = new byte[e.ChosenPhoto.Length];
      e.ChosenPhoto.Read(data, 0, data.Length);
      e.ChosenPhoto.Close();

      Guid fileId = Guid.NewGuid();
      Dictionary<string, object> postParameters = new Dictionary<string, object>();
      postParameters.Add("photo", new FormUpload.FileParameter(
        data, fileId.ToString() +
        ".jpg", "image/jpeg"));

      FormUpload upload =
        new FormUpload(baseURL + "Home/UploadPicture/" + id, postParameters);
      upload.FormUploadCompleted +=
        new FormUpload.FormUploadCompletedHandler(upload_FormUploadCompleted);
      upload.BeginMultipartFormDataPost();
    }
  }

  void upload_FormUploadCompleted(object source)
  {
    webBrowser1.Navigate(webBrowser1.Source);
  }

  private void PhoneApplicationPage_Loaded(object sender, RoutedEventArgs e)
  {
    webBrowser1.Navigate(new Uri(baseURL));
  }}
```

Figure 10 **Android EffectiveUIActivity.java**

```
public class EffectiveUIActivity extends Activity {
  /** Called when the activity is first created. */
  WebView webView;
  String cameraId;

  static String baseURL = "http://...";

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    webView = (WebView)findViewById(R.id.webView);
    webView.getSettings().setJavaScriptEnabled(true);
    webView.getSettings().setGeolocationEnabled(true);
    webView.setVerticalScrollbarOverlay(true);
    webView.loadUrl(baseURL);

    final EffectiveUIActivity activity = this;
    webView.setWebViewClient(new WebViewClient(){
      @Override
      public void onLoadResource(WebView view, String url) {
        super.onLoadResource(view, url);

        if(url.contains("Home/Image")){
          activity.createCamera();
        }
      }
      @Override
      public boolean shouldOverrideUrlLoading(WebView view, String url){
        String match = "Home/Image/";
        int i = url.indexOf(match);
        if(i>0){
          cameraId = url.substring(i+match.length());
          activity.createCamera();
          return true;
        }
        return false;
      }
    });
    webView.setWebChromeClient(new WebChromeClient(){
      @Override
      public void onGeolocationPermissionsShowPrompt(
        String origin, GeolocationPermissions.Callback callback) {
        super.onGeolocationPermissionsShowPrompt(origin, callback);
        callback.invoke(origin, true, false);
      }

    });
  }

  public void createCamera(){
    Intent intent = new Intent("android.media.action.IMAGE_CAPTURE");
    startActivityForResult(intent, 2000);
  }

  @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
      if (resultCode == Activity.RESULT_OK && requestCode == 2000) {
        Bitmap thumbnail = (Bitmap) data.getExtras().get("data");
        ByteArrayOutputStream bos = new ByteArrayOutputStream();
        thumbnail.compress(CompressFormat.JPEG, 75, bos);
        byte[] imagebytes = bos.toByteArray();

        ByteArrayBody bab = new ByteArrayBody(imagebytes, "image/jpeg",
          UUID.nameUUIDFromBytes(imagebytes).toString()+".jpg");
        HttpClient client = new DefaultHttpClient();
        HttpPost post = new HttpPost(baseURL+"Home/UploadPicture");
        MultipartEntity reqEntity =
          new MultipartEntity(HttpMultipartMode.BROWSER_COMPATIBLE);
        reqEntity.addPart("photo", bab);
        try {
          reqEntity.addPart("ID", new StringBody(cameraId, "text/plain",
            Charset.forName( "UTF-8" )));
        } catch (UnsupportedEncodingException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
          }
          post.setEntity(reqEntity);
          try {
            HttpResponse response = client.execute(post);
            BufferedReader reader = new BufferedReader(
              new InputStreamReader(
              response.getEntity().getContent(), "UTF-8"));
            String sResponse;
            StringBuilder s = new StringBuilder();

            while ((sResponse = reader.readLine()) != null) {
              s = s.append(sResponse);
            }
          } catch (ClientProtocolException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
          } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            }
            webView.loadUrl(webView.getUrl());
          }
        }
}
```

URL. If the URL string matches, the code returns "NO" to cancel the navigation and invokes the native UIImagePickerController. This allows the user to pick an image from the photo library or take a new picture with the onboard camera. After selecting the picture, the code then posts the picture back to the Web server using the ASIFormDataRequest library (allseeing-i.com/ASIHTTPRequest) before returning the UIWebView back to the normal application flow.

Graceful Degradation of the Mobile Experience

So what happens if the user of the mobile Web site isn't using the native application shell to access the camera? In this scenario, it's important to have a graceful degradation of the user experience. Graceful degradation is the concept of building your application so that it continues to function correctly even if it's viewed with less-than-optimal software. This doesn't mean that every feature works in exactly the same way or that it even looks similar to the intended experience, but it aims to ensure that all of the user's fundamental goals can still be accomplished even if the user isn't getting the best experience.

To enable graceful degradation in this application, I built an ASP.NET MVC 3 controller and view for the image capture URL, "Home/Image," that's being captured by the native application shells to provide a simple file upload form as shown in **Figure 12**. This form allows users who aren't using the enhanced mobile shells to accomplish the same task of adding a picture to a work order, although they aren't getting the integrated experience. The form posts to the same controller action used by the native application shells, encouraging code reuse among all of the different platforms and views.

> The hybrid application approach can provide significant cost advantages over unique native applications.

Figure 11 **iOS Code**

```
- (void) choosefromCamera {
    UIImagePickerController *picker = [[UIImagePickerController alloc] init];
    picker.delegate = self;
    picker.mediaTypes = [NSArray arrayWithObjects:(NSString*)kUTTypeImage, nil];
    if ([UIImagePickerController
      isSourceTypeAvailable:UIImagePickerControllerSourceTypeCamera]) {
      picker.sourceType = UIImagePickerControllerSourceTypeCamera;
    }else{
      picker.sourceType = UIImagePickerControllerSourceTypePhotoLibrary;
    }
    [self presentModalViewController:picker animated:YES];
}

- (void)imagePickerController:(UIImagePickerController *)picker
    didFinishPickingMediaWithInfo:(NSDictionary *)info {
    UIImage *image = [info objectForKey:UIImagePickerControllerOriginalImage];
    NSData *jpg = UIImageJPEGRepresentation(image, 0.3);
    [picker dismissModalViewControllerAnimated:YES];
    [picker release];
    NSString *url =
      [NSString stringWithFormat:@"%@:7511/WorkOrders/UploadPicture", baseURL];
    ASIFormDataRequest *request =
      [ASIFormDataRequest requestWithURL:[NSURL URLWithString:url]];
    [request addData:jpg withFileName:[
      NSString stringWithFormat:@"%@.jpg", [self GetUUID]]
      andContentType:@"image/jpeg" forKey:@"photo"];
    [request addPostValue:cameraId forKey:@"ID"];
    [request setDelegate:self];
    [request setDidFinishSelector:@selector(imageUploaded:)];
    [request setDidFailSelector:@selector(imageUploaded:)];
    [request startSynchronous];
    [webView reload];
}

-(void) imageUploaded:(ASIFormDataRequest *)request {
    NSString *response = [request responseString];
    NSLog(@"%@",response);
}

- (BOOL)webView:(UIWebView *)webView shouldStartLoadWithRequest:(
  NSURLRequest *)request
    navigationType:(UIWebViewNavigationType)navigationType {
    NSURL *url = [request URL];
    NSString *str = [url absoluteString];
    NSRange range = [str rangeOfString:@"WorkOrders/Image/"];
    if (range.location != NSNotFound) {
      cameraId = [str substringFromIndex:range.location+17];
      [cameraId retain];
      NSLog(@"%@", cameraId);
      [self choosefromCamera];       return NO;
    }else{
      return YES;
    }
}
```

Significant Cost Advantages

The hybrid application approach can provide significant cost advantages over unique native applications, both in the short and long term. Tools such as jQuery Mobile narrow the usability differences, which may lead to significant business advantages where native device access isn't required.

With mobile devices proliferating like wildfire, you have a few choices when looking to build a mobile application:

- **Build a Native Application for Each Platform You Want to Support** This has the distinct advantage of providing the best user experience and performance for each platform while allowing access to all the native features of the device and the marketing power of the app stores. The disadvantage, however, is that it might be significantly more expensive to build and maintain because it will require a separate codebase for each platform you wish to support. In addition, each new version of the application requires that the application be resubmitted to the app stores.
- **Build a Mobile Web Application** This has the advantage of being the simplest and cheapest option to develop, launch and update for all of the platforms, but the user experience can be compromised by the lack of access to native hardware features. Lack of access to the app stores can also compromise adoption of your application, pushing all of the marketing of your application to you.
- **Build a Hybrid Native and Mobile Web Application** This is the approach I discussed, which provides a solid compromise between the high costs of developing a unique native application for each platform and the lack of native hardware access for a mobile Web application. This option also provides access to the app stores, increasing the reach of your application.

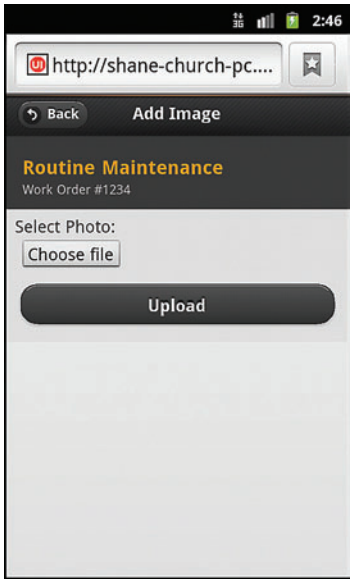


Figure 12 **A Simple File Upload Form for Graceful Degradation**

It's important to note that none of these approaches is inherently better than the others—all of them have their own strengths and weaknesses. A comprehensive cost/benefit analysis of each of the options will help determine which path is the right one for your users and your business. When making this decision, it's important to consider the user experience, up-front development costs and ongoing maintenance costs as well as more subtle factors such as marketing and user adoption.

For many business application scenarios, I advocate for inclusion of a mobile Web or hybrid application, as the additional effort of building unique native applications for each mobile platform could outweigh the business benefit. The business scenarios need to be carefully reviewed within the confines of a mobile Web or hybrid application deployment.

Mobile applications are here to stay and are increasingly important as computing shifts away from the traditional desktop experience to an array of mobile experiences. As you look to build applications in the mobile space, remember that compromise isn't always a dirty word and that it can result in the best of both worlds. ■

SHANE CHURCH *is a technical lead for EffectiveUI in Denver, Colo. He has been developing in the Microsoft .NET Framework with a focus on ASP.NET and Microsoft mobile technologies since 2002. His blog is located at s-church.net. You can learn more about EffectiveUI at effectiveui.com.*

THANKS to the following technical expert for reviewing this article: *Dr. James McCaffrey*

Create a Continuous Client Using Portable Class Libraries

David Kean

I feel lucky to live in the days of continuously connected devices. I love that I'm able to reply to e-mail using my phone while riding the bus home. It's amazing to be able to Skype with my family on the other side of the world and team up with like-minded gamers across the country on my Xbox. However, in this world of permanent Internet connectivity, there is, as Joshua Topolsky puts it, "a missing link in our computing experience" (engt.co/9GVeKl).

This missing link refers to the lack of what Topolsky calls a *continuous client*; that is, a solution to the broken workflow that occurs today when you move from one device to another. As I switch among my PC, tablet and phone in a typical day, my current browsing session, documents, windows and application state should naturally flow to all of them. That way, I'd spend less time on context switching and more time on actual work and play.

In this article, I'll show you how to build a simple continuous client application that spans multiple devices and platforms. I'll make use of the new Portable Class Libraries (PCLs) to ease the development of a cross-platform application, and the cloud—in particular Windows Azure AppFabric Service Bus—to handle the communication between the devices.

This is based on beta versions of Visual Studio 11 and Windows 8. All information is subject to change.

This article discusses:

- Organizing a cross-platform application
- Converting existing libraries to PCL
- The application's two views
- Storing state changes as a sequence of events
- Syncing data to the cloud

Technologies discussed:

Portable Class Libraries, Visual Studio 10 and Visual Studio 11 Beta, Windows Phone, Windows 8, Windows Azure AppFabric Service Bus

Code download available at:

code.msdn.microsoft.com/mag201203PCL

On Your Way Home ...

It's late afternoon and I'm at work trying to fix that last bug quickly so I can avoid peak-hour traffic. The inevitable phone call comes: "Honey, on your way home can you pick up some milk, bread and chickpeas?" I hang up, get to the store and realize I've forgotten what to buy. In the end, I head home with items we already have in the pantry. It's frustrating, and today's solution tends to involve a lot of back-and-forth phone calling: "Did you say frozen peas or chickpeas?" "Chickpeas. And while you're there, can you buy toilet paper?"

To help alleviate our marriage tensions around this particular issue (the others will have to wait for another day), I'll write a simple app called "On Your Way Home" that runs on our Windows Phone-based devices and Windows 8 beta tablets and allows my wife and me to easily track our shopping list. It will keep us both informed, in real time, of any changes to the shopping list so that at any time we know exactly what we need to buy.

Given that a smartphone running Window Phone and a Windows 8-based tablet are different devices, with differing flavors of the Microsoft .NET Framework and Windows, I'll use PCLs to abstract away platform differences and enable me to share as much application logic as possible, including all the of the communication with the Windows Azure AppFabric Service Bus. I'll also use the Model-View-ViewModel (MVVM) pattern (bit.ly/GW7I) to facilitate the use of the same Models and ViewModels from our device-specific Views.

Portable Class Libraries

In the past, cross-platform development in the .NET Framework hasn't been easy. While the .NET Framework had grand dreams as a cross-platform runtime, Microsoft hasn't yet fully delivered on the promise. If you've ever attempted to deliver a .NET Framework-based application or framework that spanned multiple devices, you'll have noticed that a few things got in the way.

On the Runtime Side The assembly factoring, versioning and assembly names are different among the .NET platforms. For example, System.Net.dll on the .NET Framework, which contains peer-to-peer networking APIs, means something entirely different on Silverlight, where it contains the core networking stack. To find those APIs on the .NET Framework, you'll need to reference System.dll. The assembly versions are also not the same; Silverlight adopts 2.0.5.0 for versions 2.0 to 4, whereas 2.0.0.0 and 4.0.0.0 were adopted for .NET Framework versions 2.0 to 4. These differences have, in the past, prevented an assembly compiled for one platform from running on another.

On the Visual Studio Side Right from the beginning you need to decide which platform to target—the .NET Framework, Silverlight or Windows Phone. Once that decision is made, it's extremely hard to move to or support a new platform. For example, if you're already targeting the .NET Framework, targeting the .NET Framework and Silverlight means creating a new project and either copying or linking the existing files into that project. If you're lucky, you might have factored your application in such a way that platform-specific pieces are easily replaced. If not (and this is probably more likely), you'll need to #if PLATFORM your way around each build error until you have a clean build.

This is where the new PCLs can help. PCLs, available as a free add-on to Visual Studio 2010 (bit.ly/ekNnsN) and built into Visual Studio 11 beta, provide an easy way to target multiple platforms using a single project. You can create a new PCL, choose the frameworks you'd like to target (see **Figure 1**) and start writing code. Under the covers, the PCL tools handle the API differences and filter IntelliSense so you see only classes and members that are available and work across all the frameworks you've selected. The resulting assembly can then be referenced and run, without *any* changes, on all indicated frameworks.

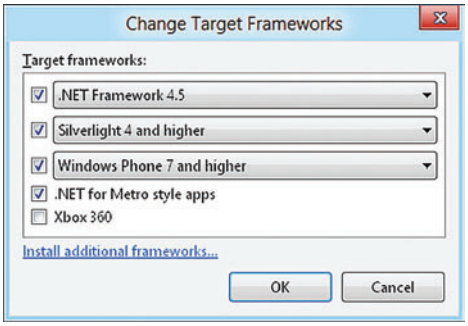


Figure 1 **Portable Class Library Target Frameworks**

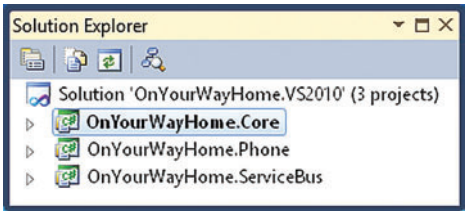


Figure 2 **Windows Phone Project Layout in Visual Studio 2010**

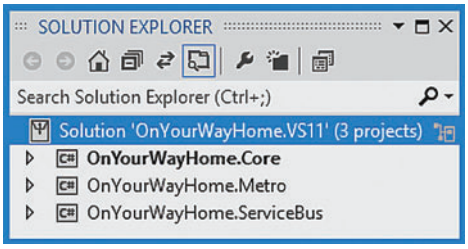


Figure 3 **Windows Metro-Style App Project Layout in Visual Studio 11**

Solution Layout

A typical way to organize a cross-platform app using a PCL is to have one or more portable projects containing the shared components, and have platform-specific projects for each platform that references these projects. For this application, I'll need two Visual Studio solutions—one created in Visual Studio 2010 (OnYourWayHome.VS2010) containing my Windows Phone app and one created in Visual Studio 11 (OnYourWayHome.VS11) containing my Windows Metro-style app. I need multiple solutions because at the time of writing, the Windows Phone SDK 7.1 works only on top of Visual Studio 2010, whereas the new Windows 8 tools are available only as part of Visual Studio 11. There isn't (currently) a single version that supports both. Don't despair, though; a new feature available in Visual Studio 11 helps me out here. I'm able to open most projects created in the earlier version without having to convert them to the new format. This allows me to have a single PCL project and reference it from both solutions.

Figures 2 and **3** show the project layout for my application. OnYourWayHome.Core, a PCL project, contains the models, view models, common services and platform abstractions. OnYourWayHome.ServiceBus, also a PCL project, contains portable versions of the APIs that will talk to Windows Azure. Both projects are shared between the Visual Studio 2010 solution and Visual Studio 11. OnYourWayHome.Phone and OnYourWayHome.Metro are platform-specific projects targeting Windows Phone 7.5 and .NET for Metro-style apps, respectively. These contain the device-specific views (such as the pages in the application) and implementations of the abstractions found in OnYourWayHome.Core and OnYourWayHome.ServiceBus.

Converting Existing Libraries to PCLs

To communicate with Windows Azure, I downloaded the Silverlight-based REST sample from servicebus.codeplex.com and converted it to a PCL project. Some libraries are easier to convert than others, but you'll inevitably run into situations where a given type or method isn't available. Here are some typical reasons a given API might not be supported in PCLs:

The API Isn't Implemented by All Platforms Traditional .NET Framework file IOs, such as System.IO.File and System.IO.Directory,

Figure 4 **HMAC SHA256 Implementations for Windows Phone and Windows 8**

```
// Windows Phone implementation
public class PhoneServiceBusAdapter : ServiceBusAdapter
{
  public override byte[] ComputeHmacSha256(byte[] secretKey, byte[] data)
  {
    using (var cryptoProvider = new HMACSHA256(secretKey))
    {
      return cryptoProvider.ComputeHash(data);
    }
  }
}

// Windows 8 implementation
public class MetroServiceBusAdapter : ServiceBusAdapter
{
  private const string HmacSha256AlgorithmName = "HMAC_SHA256";

  public override byte[] ComputeHmacSha256(byte[] secretKey, byte[] data)
  {
    var provider = MacAlgorithmProvider.OpenAlgorithm(HmacSha256AlgorithmName);
    var key = provider.CreateKey(_secretKey.AsBuffer());
    var hashed = CryptographicEngine.Sign(key, buffer.AsBuffer());

    return hashed.ToArray();
  }
}
```

fall into this bucket. Silverlight and Windows Phone use the System.IO.IsolatedStorage APIs (though different from the .NET Framework version), whereas Windows 8 Metro-style apps use Windows.Storage.

The API Isn't Compatible Across All Platforms Some APIs look and feel the same, but it's hard or impossible to write code against them in a portable and consistent way. ThreadStaticAttribute, which enables static fields to have a unique value for each thread, is an example. Though it's present on both the Windows Phone and Xbox platforms, neither of their runtimes supports it.

The API Is Considered Obsolete or Legacy These APIs either contain behavior that's unlikely to be present on future platforms, or they've been replaced by newer technologies. BackgroundWorker is an example of this; it was replaced by Task and the new asynchronous program features in Visual Studio 11 beta.

We Ran out of Time Most APIs weren't written with portability in mind. We spend a significant amount of time going through each API to make sure it can be programmed against in a portable manner. This might involve tweaking or adding to the API to make it portable. Because of the time and effort involved, in the first version of PCLs we made available on Visual Studio Gallery, we prioritized the high-value, highly used APIs. System.Xml.Linq.dll and System.ComponentModel.DataAnnotations.dll are examples of APIs that weren't available in that first version but are now available in the Visual Studio 11 beta release.

There are a couple of different ways of handling an API that falls into one of these scenarios. Sometimes there's a simple replacement. For example, Close methods (Stream.Close, TextWriter.Close and so forth) have been deprecated in PCL and replaced with Dispose. In such cases, it's just a matter of replacing a call to the former with the latter. But sometimes it's a little harder and takes more work. One situation I encountered while converting the Service Bus APIs involved the HMAC SHA256 hash code provider. It isn't available in a PCL because of the cryptography differences between Windows Phone and Metro-style apps. Windows Phone apps use .NET-based APIs to encrypt, decrypt and hash data, while Metro-style apps use the new native Windows Runtime (WinRT) APIs.

The code in particular that failed to build after the conversion was the following:

```
using (HMACSHA256 sha256 = new HMACSHA256(issuerSecretBytes))
{
  byte[] signatureBytes = sha256.ComputeHash(Encoding.UTF8.GetBytes(token));
  signature = Convert.ToBase64String(signatureBytes);
}
```

To help bridge the gaps between the Phone crypto APIs and the WinRT crypto APIs, I invented a platform abstraction representing the Service Bus requirement. In this case, the Service Bus needed a way to calculate an HMAC SHA256 hash:

```
public abstract class ServiceBusAdapter
{
  public static ServiceBusAdapter Current
  {
    get;
    set;
  }

  public abstract byte[] ComputeHmacSha256(byte[] secretKey, byte[] data);
}
```

I added ServiceBusAdapter to the portable project, as well as a static property for setting the current abstraction, which will become important later. Next, I created Windows Phone- and Windows 8-specific HMAC SHA256 implementations of this abstraction, and put these in their respective projects, as shown in **Figure 4**.

At startup in the Windows Phone project, I then "bootstrapped" the Service Bus by setting the Phone-specific adapter as the current adapter:

```
ServiceBusAdapter.Current = new PhoneServiceBusAdapter();
```

I did the same for the Windows 8 project:

```
ServiceBusAdapter.Current = new MetroServiceBusAdapter();
```

With everything in place, I then changed the original non-compiling code to call through the adapter:

```
var adapter = ServiceBusAdapter.Current;
byte[] signatureBytes = adapter.ComputeHmacSha256(issuerSecretBytes,
Encoding.UTF8.GetBytes(token));
```

So although there are two different ways of computing the hash depending on the platform, the portable project talks to both using a single interface. This can take a little bit of work up front, but I can easily reuse the infrastructure as I run into more APIs that need bridging between the platforms.

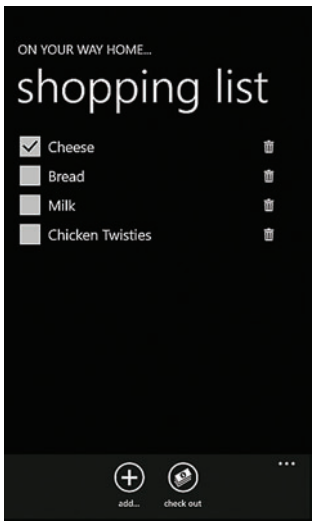


Figure 5 **ShoppingListView**

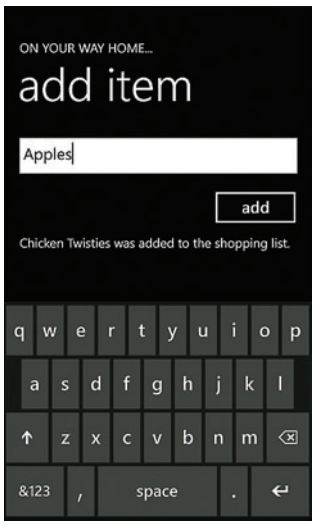


Figure 6 **AddGroceryItemView**

Figure 7 **AddGroceryItemView Controls for Window Phone**

```
<StackPanel>

  <TextBox Text="{Binding Name, Mode=TwoWay}"
           Width="459"
           Height="80" />

  <Button Command="{Binding Add}"
          Content="add"
          Margin="307,5,0,0" />

  <TextBlock Text="{Binding NotificationText}"
             Margin="12,5,0,0"/>
</StackPanel>
```

As a side note, I used a static property to access and register the adapter, which makes it easier to move existing APIs over to using the adapter. If you're using a dependency injection framework such as the Managed Extensibility Framework (MEF), Unity or Autofac, you'll find that it's natural to register the platform-specific adapter into the container and have the container "inject" the adapter into portable components that need it.

Application Layout

My shopping list application, On Your Way Home, has two simple views: ShoppingListView, which displays the current items on the shopping list; and AddGroceryItemView, which allows a user to add more items to the list. **Figures 5** and **6** show the Windows Phone versions of these views.

ShoppingListView shows all the items that are yet to be purchased, with the idea that as you walk around the store, you check off each item as you add it to the cart. After purchasing the items, clicking *check out* causes the checked items to be taken off the list, indicating they no longer need to be purchased. Devices sharing the same shopping list instantly (well, as instantly as the network behind it allows) see changes made by another person.

The Views, which live in the platform-specific projects, consist mainly of XAML and have very little codebehind, which limits the amount of code you need to duplicate between the two platforms. Using XAML data binding, the Views bind themselves to portable

Figure 8 **AddGroceryItemViewModel Class for Windows 8**

```
public class AddGroceryItemViewModel : NavigatableViewModel
{
  private string _name;
  private string _notificationText;
  private ICommand _addCommand;

  [...]

  public ICommand AddCommand
  {
    get { return _addCommand ?? (_addCommand = new ActionCommand(Add)); }
  }

  public string Name
  {
    get { return _name ?? String.Empty; }
    set { base.SetProperty(ref _name, value, "Name"); }
  }

  public string NotificationText
  {
    get { return _notificationText ?? string.Empty; }
    set { base.SetProperty(ref _notificationText, value, "NotificationText"); }
  }
}
```

Figure 9 **Item Added Event**

```
// Published when a grocery item is added to a shopping list
[DataContract]
public class ItemAddedEvent : IEvent
{
  public ItemAddedEvent()
  {
  }

  [DataMember]
  public Guid Id
  {
    get;
    set;
  }

  [DataMember]
  public string Name
  {
    get;
    set;
  }
}
```

ViewModels that provide the commands and data that run the Views. Because there's no common UI framework that ships across all platforms, PCL projects can't reference UI-specific APIs. However, when targeting frameworks that support them, they can take advantage of APIs that are typically used by ViewModels. This includes the core types that make XAML data binding work, such as INotifyPropertyChanged, ICommand and INotifyCollectionChanged. Also, although the WinRT XAML framework doesn't support them, System.ComponentModel.DataAnnotations and INotifyDataErrorInfo have been added for completeness, and this enables custom XAML validation frameworks to support portable ViewModels/Models.

Figures 7 and **8** show examples of View/ViewModel interactions. **Figure 7** shows the controls on the Window Phone version of AddGroceryItemView and their bindings. These controls are bound against the properties on the AddGroceryItemViewModel, which is shared with both the Windows Phone and Windows 8 projects, as shown in **Figure 8**.

Event Sourcing

On Your Way Home is based heavily around the concept of event sourcing (bit.ly/3SpC9h). This is the idea that all state changes to an application are published and stored as a sequence of *events*. In this

Figure 10 **Publishing the Event**

```
public class AddGroceryItemViewModel : NavigatableViewModel
{
  private readonly IEventAggregator _eventAggregator;

  [...]

  // Adds an item to the shopping list
  private void Add()
  {
    var e = new ItemAddedEvent();
    e.Id = Guid.NewGuid();
    e.Name = Name;

    _eventAggregator.Publish(e);

    NotificationText = String.Format("{0} was added to the shopping list.", Name);
    Name = string.Empty;
  }
}
```

Figure 11 **The ShoppingList Class**

```
public class ShoppingList : IEventHandler<ItemAddedEvent>
{
  public ShoppingList(IEventAggregator eventAggregator)
  {
    Requires.NotNull(eventAggregator, "eventAggregator");

    _eventAggregator = eventAggregator;
    _eventAggregator.Subscribe<ItemAddedEvent>(this);
  }

  [...]

  public ReadOnlyObservableCollection<GroceryItem> GroceryItems
  {
    get { return _groceryItems; }
  }

  public void Handle(ItemAddedEvent e)
  {
    var item = new GroceryItem();
    item.Id = e.Id;
    item.Name = e.Name;
    item.IsInCart = false;

    _groceryItems.Add(item);
  }
}
}
```

context, *event* doesn't refer to the thing defined by the C# event keyword (although the idea is the same), but rather to concrete classes that represent a single change to the system. These are published through what's called an event aggregator, which then notifies one or more handlers that do work in response to the event. (For more about event aggregation, see Shawn Wildermuth's article, "Composite Web Apps with Prism," at msdn.microsoft.com/magazine/dd943055.)

For example, the event that represents a grocery item being added to the shopping list looks something like what's shown in **Figure 9**.

The ItemAddedEvent class contains information about the event: in this case, the name of the grocery item that was added and an ID that's used to uniquely represent the grocery item within a shopping list. Events are also marked with [DataContract], which makes it easier for them to be serialized to disk or sent over the wire.

This event is created and published when the user clicks the *add* button on the AddGroceryItemView, as shown in **Figure 10**.

Note that this method doesn't directly make any change to the shopping list; it simply publishes the ItemAddedEvent to the event aggregator. It's the responsibility of one of the event handlers listening

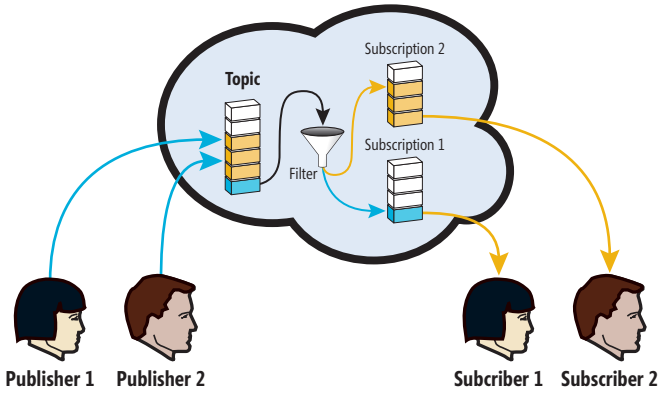


Figure 12 **Service Bus Topic**

Figure 13 **The AzureServiceEventHandler Class**

```
public class AzureServiceBusEventHandler : DisposableObject,
IEventHandler<IEvent>, IStartupService
{
  private readonly IAzureServiceBus _serviceBus;
  private readonly IAzureEventSerializer _eventSerializer;

  public AzureServiceBusEventHandler(IEventAggregator eventAggregator,
    IAzureServiceBus serviceBus, IAzureEventSerializer eventSerializer)
  {
    _eventAggregator = eventAggregator;
    _eventAggregator.SubscribeAll(this);
    _serviceBus = serviceBus;
    _serviceBus.MessageReceived += OnMessageReceived;
    _eventSerializer = eventSerializer;
  }

  [...]

  public void Handle(IEvent e)
  {
    BrokeredMessage message = _eventSerializer.Serialize(e);

    _serviceBus.Send(message);
  }
}
```

to this event to do something with it. In this case, a class called ShoppingList subscribes to and handles the event, as shown in **Figure 11**.

Every time ItemAddedEvent is published, ShoppingList creates a new GroceryItem using the data from the event and adds it to the shopping list. The ShoppingListView, which is indirectly bound to the same list via its ShoppingListViewModel, is also updated. This means that when the user navigates back to the shopping list page, the items he just added to the list are shown as expected. The process of removing an item from the shopping list, adding an item to a cart and checking out the cart are all handled using the same event publish/subscribe pattern.

It may at first seem like a lot of indirection for something as simple as adding items to a shopping list: the AddGroceryItemViewModel.Add method publishes an event to the IEventAggregator, which passes it onto the ShoppingList, which adds it to the grocery list. Why doesn't the AddGroceryItemViewModel.Add method simply bypass the IEventAggregator and add the new GroceryItem directly to the ShoppingList? I'm glad you asked. The advantage of treating all state changes to the system as events is that it encourages all the individual parts of the application to be very loosely coupled. Because the publisher and subscribers don't know about each other, inserting a new feature in the pipeline, such as syncing data to and from the cloud, is a lot simpler.

Syncing Data to the Cloud

I've covered the basic functionality of the application running on a single device, but there's still the problem of getting the changes a user makes to the shopping list to other devices, and vice versa. This is where the Windows Azure AppFabric Service Bus comes in.

Windows Azure AppFabric Service Bus is a feature that enables applications and services to easily talk with each other over the Internet, avoiding the complexities of navigating communication obstacles such as firewalls and Network Address Translation (NAT) devices. It provides both REST and Windows Communication Foundation (WCF) HTTP endpoints hosted by Windows Azure and sits in between the publisher and the subscriber.

Figure 14 **Deserializing a Received Message**

```
public class AzureServiceBusEventHandler : DisposableObject, IEventHandler<IEvent>,
  IStartupService
{
  private readonly IAzureServiceBus _serviceBus;
  private readonly IAzureEventSerializer _eventSerializer;

  [...]

  private void OnMessageReceived(object sender, MessageReceivedEventArgs args)
  {
    IEvent e = _eventSerializer.Deserialize(args.Message);

    _eventAggregator.Publish(e);
  }
}
```

There are three main ways to communicate using the Windows Azure AppFabric Service Bus; for the purposes of my application, however, I'll just cover Topics. For a full overview, check out "An Introduction to the Windows Azure AppFabric Service Bus" at bit.ly/uNVaXG.

For publishers, a Service Bus Topic is akin to a big queue in the cloud (see **Figure 12**). Completely unaware of who's listening, publishers push messages to the Topic, where they're held ad infinitum until requested by a subscriber. To get messages from the queue, subscribers pull from a *Subscription*, which filters messages published to the Topic. Subscriptions act like a particular queue, and messages removed from a Subscription will still be seen from other Subscriptions if their own filters include them.

In On Your Way Home, the AzureServiceEventHandler class is the bridge between the application and the Service Bus. Similar to ShoppingList, it also implements IEventHandler<T>, but instead of specific events, AzureServiceEventHandlers can handle them all, as shown in **Figure 13**.

Every change a user makes to the state of the shopping list is handled by AzureServiceBusEventHandler and pushed directly to the cloud. Neither AddGroceryItemViewModel, which publishes the event, nor ShoppingList, which handles it on the local device, is aware that this happens.

The trip back from the cloud is where an event-based architecture really pays off. When the AzureServiceEventHandler detects that a new message has been received on the Service Bus (via the IAzureServiceBus.MessageReceived C# event), it does the reverse of what it did earlier and deserializes the received message back into an event. From here, it gets published back via the event aggregator, which causes it to be treated as though the event came from *within* the application, as shown in **Figure 14**.

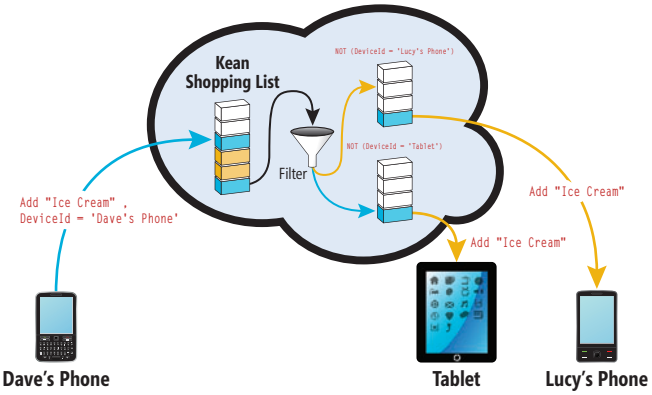


Figure 15 **Device-to-Device Workflow**

The ShoppingList isn't aware (nor does it care) about the source of the event and handles those coming from the Service Bus/cloud as though they came directly from a user's input. It updates its list of groceries, which in turn causes any of the views bound to that list to be updated as well.

If you pay special attention, you might notice one little problem with the workflow: Events that get sent to the cloud from the local device come back to that same device and cause duplication of the data. Worse, changes to other, unrelated shopping lists will also come to that device. I don't know about you, but I'm pretty sure I don't want to see other people's food choices appearing on my shopping list. To prevent this, a Service Bus Topic is created per list, and a Subscription per device, which listens to the Topic. When the messages are published to the Topic from the device, a property containing the device ID is sent along with the messages, which the Subscription filter uses to exclude messages that came from its own device. **Figure 15** shows this workflow.

Wrapping Up

I covered a lot in this article: Portable Class Libraries simplified my solution and significantly reduced the amount of code I needed to write to target the two platforms. Also, changing application state via events made it very easy to sync that state with the cloud. There's still a lot I've left unsaid, however, that you'll want to factor in when developing a continuous client. I didn't talk about offline event caching and fault tolerance (what if the network isn't available when I publish an event?), merge conflicts (what if another user makes a change that conflicts with mine?), playback (if I attach a new device to the shopping list, how does it get updated?), access control (how do I prevent unauthorized users accessing data they shouldn't?) and finally, persistence. In the sample code for the article, the application doesn't save the shopping list between launches. I'll leave this as an exercise for you; it might be an interesting challenge if you want to play around with the code. A naïve (or rather the traditional) way of approaching persistence might be to a put a hook directly into the ShoppingList class, mark the GroceryItem objects as serializable and save them off to a file. Before going down this route, though, stop and think about it: Given that the ShoppingList already handles events natively and already doesn't care where they come from, syncing data to and from the cloud looks surprisingly like saving and restoring data from disk, doesn't it? ■

DAVID KEAN *is a developer on the .NET Framework team at Microsoft, where he works on the Base Class Library (BCL) team. Prior to that, he worked on the often well-loved but also greatly misunderstood tool FxCop and its related sibling, Visual Studio Code Analysis. Originally from Melbourne, Australia, he's now based in Seattle, Wash., with his wife, Lucy, and three children, Jack, Sarah and Ben. He can be found blogging on davesbox.com.*

THANKS to the following technical experts for reviewing this article: *Nicholas Blumhardt and Immo Landwerth*

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

YOUR MAP TO THE .NET DEVELOPMENT PLATFORM

BIG CODE IN THE BIG APPLE

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

SUPPORTED BY

Microsoft | Microsoft Visual Studio | msdn | Visual Studio MAGAZINE

PRODUCED BY

1105 MEDIA

New Concurrency Features in Visual C++ 11

Diego Dagum

The latest C++ iteration, known as C++11 and approved by the International Organization for Standardization (ISO) in the past year, formalizes a new set of libraries and a few reserved words to deal with concurrency. Many developers have used concurrency in C++ before, but always through a third-party library—often directly exposing OS APIs.

This article discusses Visual C++ 11, a prerelease technology. All related information is subject to change.

This article discusses:

- Parallel execution
- Asynchronous tasks
- Threads
- Variables and exceptions
- Synchronization
- Atomic types
- Mutexes and locks
- Condition variables

Technologies discussed:

Visual C++ 11

Code download available at:

code.msdn.microsoft.com/mag201203CPP

Herb Sutter announced in December 2004 that the "free performance lunch" was over in the sense that CPU manufacturers were prevented from shipping faster CPUs by physical power consumption and increasing heat reasons. This led to the current, mainstream multicore era, a new reality to which C++—the standard one—has just made an important leap to adapt.

The rest of this article is organized in two main sections and smaller subsections. The first main section, starting with Parallel Execution, covers technologies that allow applications to run independent or semi-independent activities in parallel. The second main section, starting with Syncing up Concurrent Execution, explores mechanisms for synchronizing the way these activities handle data, thus avoiding race conditions.

This article is based on features included in the upcoming version of Visual C++ (for now, called Visual C++ 11). A few of them are already available in the current version, Visual C++ 2010. Although not a guide to model parallel algorithms, nor an exhaustive documentation about all the available options, this article is a solid introduction to new C++11 concurrency features.

Parallel Execution

When you model processes and design algorithms over data, there's a natural tendency to specify them in a sequence of steps. As long as performance is within acceptable bounds, this is the most recommendable schema because it's typically easier to understand—a requirement for maintainable code bases.

Figure 1 **Sequential Case Code**

```
int a, b, c;

int calculateA()
{
  return a+a*b;
}

int calculateB()
{
  return a*(a+a*(a+1));
}

int calculateC()
{
  return b*(b+1)-b;
}

int main(int argc, char *argv[])
{
  getUserData(); // initializes a and b
  c = calculateA() * (calculateB() + calculateC());
  showResult();
}
```

When performance becomes a worrisome factor, a classic initial attempt to overcome the situation is to optimize the sequential algorithm in order to reduce the consumed CPU cycles. This can be done until you come to a point where no further optimizations are available—or they're hard to achieve. Then the time to split the sequential series of steps into activities of simultaneous occurrence has come.

In the first section you'll learn about the following:

- **Asynchronous tasks:** those smaller portions of the original algorithm only linked by the data they produce or consume.
- **Threads:** units of execution administrated by the runtime environment. They relate to tasks in the sense that tasks are run on threads.
- **Thread internals:** thread-bound variables, exceptions propagated from threads and so on.

Asynchronous Tasks

In the companion code to this article, you'll find a project called Sequential Case, as shown in **Figure 1**.

The main function asks the user for some data and then submits that data to three functions: calculateA, calculateB and calculateC. The results are later combined to produce some output information for the user.

The calculating functions in the companion material are coded in a way such that a random delay between one and three seconds is introduced in each. Considering that these steps are executed sequentially, this leads to an overall execution time—once the input data is entered—of nine seconds in the worst-case scenario. You can try this code out by pressing F5 and running the sample.

So I need to revise the execution sequence and find steps to be performed concurrently. As these functions are independent, I can execute them in parallel by using the async function:

```
int main(int argc, char *argv[])
{
  getUserData();
  future<int> f1 = async(calculateB), f2 = async(calculateC);
  c = (calculateA() + f1.get()) * f2.get();
  showResult();
}
```

I've introduced two concepts here: *async* and *future*, both defined in the <future> header and the std namespace. The first one receives a function, a lambda or a function object (functor) and returns a future. You can understand the concept of a future as the placeholder for an eventual result. Which result? The one returned by the function called asynchronously.

At some point, I'll need the results of these parallel-running functions. Calling the get method on each future blocks the execution until the value is available.

You can test and compare the revised code with the sequential case by running the AsyncTasks project in the companion sample. The worst-case delay of this modification is about three seconds versus nine seconds for the sequential version.

This is a lightweight programming model that releases the developer from the duty of creating threads. However, you can specify threading policies, but I won't cover those here.

Threads

The asynchronous task model presented in the previous section might suffice in some given scenarios, but if you need a deeper handling and control of the execution of threads, C++11 comes with the thread class, declared in the <thread> header and located in the std namespace.

Despite being a more complex programming model, threads offer better methods for synchronization and coordination, allowing them to yield execution to another thread and wait for a determined amount of time or until another thread is finished before continuing.

In the following example (available in the Threads project of the companion code), I have a lambda function, which, given an integer argument, prints its multiples of less than 100,000 to the console:

```
auto multiple_finder = [](int n) {
  for (int i = 0; i < 100000; i++)
    if (i%n==0)
      cout << i << " is a multiple of " << n << endl;
};

int main(int argc, char *argv[])
{
  thread th(multiple_finder, 23456);
  multiple_finder(34567);
  th.join();
}
```

As you'll see in later examples, the fact that I passed a lambda to the thread is circumstantial; a function or functor would've sufficed as well.

Figure 2 **Associating Futures with Promises**

```
typedef int (*calculate)(void);
void func2promise(calculate f, promise<int> &p)
{
  p.set_value(f());
}

int main(int argc, char *argv[])
{
  getUserData();
  promise<int> p1, p2;
  future<int> f1 = p1.get_future(), f2 = p2.get_future();
  thread t1(&func2promise, calculateB, std::ref(p1)),
    t2(&func2promise, calculateC, std::ref(p2));
  c = (calculateA() + f1.get()) * f2.get();
  t1.join(); t2.join();
  showResult();
}
```

Figure 3 **Thread Local Storage and Thread Exceptions**

```
thread_local unsigned sum_total = 0;

void sum_until_element_with_threshold(unsigned element,
  unsigned threshold, exception_ptr& pExc)
{
  try{
    find_if_not(begin(v), end(v), [=](const unsigned i) -> bool {
      bool ret = (i!=element);
      sum_total+= i;
      if (sum_total>threshold)
        throw runtime_error("Sum exceeded threshold.");
      return ret;
    });
    cout << "(Thread #" << this_thread::get_id() << ") " <<
      "Sum of elements until " << element << " is found: " << sum_total << endl;
  } catch (...) {
    pExc = current_exception();
  }
}
```

In the main function I run this function in two threads with different parameters. Take a look at my result (which could vary between different runs due to timings):

```
0 is a multiple of 23456
0 is a multiple of 34567
23456 is a multiple of 23456
34567 is a multiple of 34567
46912 is a multiple of 23456
69134 is a multiple of 34567
70368 is a multiple of 23456
93824 is a multiple of 23456
```

I might implement the example about asynchronous tasks in the previous section with threads. For this, I need to introduce the concept of a promise. A promise can be understood as a sink through which a result will be dropped when available. Where will that result come out once dropped? Each promise has an associated future.

The code shown in **Figure 2**, available in the Promises project of the sample code, associates three threads (instead of tasks) with promises and makes each thread call a calculate function. Compare these details with the lighter AsyncTasks version.

Thread-Bound Variables and Exceptions

In C++ you can define global variables whose scope is bound to the entire application, including threads. But relative to threads, now there's a way to define these global variables such that every thread keeps its own copy. This concept is known as thread local storage and it's declared as follows:

```
thread_local int subtotal = 0;
```

If the declaration is done in the scope of a function, the visibility of the variable will be narrowed to that function but each thread will keep maintaining its own static copy. That is to say, values of the variable per thread are being kept between function invocations.

Although thread_local isn't available in Visual C++ 11, it can be simulated with a non-standard Microsoft extension:

```
#define  thread_local __declspec(thread)
```

What would happen if an exception were thrown inside a thread? There will be cases in which the exception can be caught and handled in the call stack inside the thread. But if the thread doesn't deal with the exception, you need a way to transport the exception to the initiator thread. C++11 introduces such mechanisms.

In **Figure 3**, available in the companion code in the project Thread-Internals, there's a function sum_until_element_with_threshold, which traverses a vector until it finds a specific element, summing all the elements along the way. If the sum exceeds a threshold, an exception is thrown.

If that happens, the exception is captured via current_exception into an exception_ptr.

The main function triggers a thread on sum_until_element_with_threshold, while calling that same function with a different parameter. When both invocations have finished (the one in the main thread and the one in the thread triggered from it), their respective exception_ptrs will be analyzed:

```
const unsigned THRESHOLD = 100000;

vector<unsigned> v;

int main(int argc, char *argv[])
{
  exception_ptr pExc1, pExc2;

  scramble_vector(1000);
  thread th(sum_until_element_with_threshold, 0, THRESHOLD, ref(pExc1));
  sum_until_element_with_threshold(100, THRESHOLD, ref(pExc2));
  th.join();

  dealWithExceptionIfAny(pExc1);
  dealWithExceptionIfAny(pExc2);
}
```

If any of these exception_ptrs come initialized—a sign that some exception happened—their exceptions are triggered back with rethrow_exception:

```
void dealWithExceptionIfAny(exception_ptr pExc)
{
  try
  {
    if (!(pExc==exception_ptr()))
      rethrow_exception(pExc);
    } catch (const exception& exc) {
      cout << "(Main thread) Exception received from thread: " <<
        exc.what() << endl;
  }
}
```

This is the result of our execution, as the sum in the second thread exceeded its threshold:

```
(Thread #10164) Sum of elements until 0 is found: 94574
(Main thread) Exception received from thread: Sum exceeded threshold.
```

Figure 4 **Lock with Wait**

```
void funcB()
{
  int successful_attempts = 0;
  for (int i = 0; i<5; ++i)
  {
    unique_lock<mutex> ul(mx, try_to_lock_t());
    if (ul)
    {
      ++successful_attempts;
      cout << this_thread::get_id() << ": lock attempt successful." <<
        endl;
      ... // Do something in the critical region
      cout << this_thread::get_id() << ": releasing lock." << endl;
    } else {
      cout << this_thread::get_id() <<
        ": lock attempt unsuccessful. Hibernating..." << endl;
      this_thread::sleep_for(chrono::seconds(1));
    }
  }
  cout << this_thread::get_id() << ": " << successful_attempts
    << " successful attempts." << endl;
}
```

Figure 5 **Executing the Sample Project Mutex**

```
funcB: lock attempt successful.
funcA: locking with wait ...
funcB: releasing lock.
funcA: lock secured ...
funcB: lock attempt unsuccessful. Hibernating ...
funcA: releasing lock.
funcB: lock attempt successful.
funcA: locking with wait ...
funcB: releasing lock.
funcA: lock secured ...
funcB: lock attempt unsuccessful. Hibernating ...
funcB: lock attempt unsuccessful. Hibernating ...
funcA: releasing lock.
funcB: 2 successful attempts.
funcA: locking with wait ...
funcA: lock secured ...
funcA: releasing lock.
```

Syncing up Concurrent Execution

It would be desirable if all applications could be split into a 100 percent-independent set of asynchronous tasks. In practice this is almost never possible, as there are at least dependencies on the data that all parties concurrently handle. This section introduces new C++11 technologies to avoid race conditions.

You'll learn about:

- **Atomic types:** similar to primitive data types, but enabling thread-safe modification.
- **Mutexes and locks:** elements that enable us to define thread-safe critical regions.
- **Condition variables:** a way to freeze threads from execution until some criteria is satisfied.

Atomic Types

The <atomic> header introduces a series of primitive types—atomic_char, atomic_int and so on—implemented in terms of interlocking operations. Thus, these types are equivalent to their homonyms without the atomic_ prefix but with the difference that all their assignment operators (==, ++, --, +=, *= and so on) are protected from race conditions. So it won't happen that in the midst of an assignment to these data types, another thread irrupts and changes values before we're done.

Figure 6 **Waking Up Threads Through Conditional Variables**

```
void consumer()
{
  unique_lock<mutex> l(m);
  int failed_attempts = 0;
  while (true)
  {
    mq.lock();
    if (q.size())
    {
      int elem = q.front();
      q.pop();
      mq.unlock();
      failed_attempts = 0;
      cout << "Consumer: fetching " << elem << " from queue." << endl;
      ... // Consume elem
    } else {
      mq.unlock();
      if (++failed_attempts>1)
      {
        cout << "Consumer: too many failed attempts -> Exiting." << endl;
        break;
      } else {
        cout << "Consumer: queue not ready -> going to sleep." << endl;
        cv.wait_for(l, chrono::seconds(5));
      }
    }
  }
}
```

In the following example there are two parallel threads (one being the main) looking for different elements within the same vector:

```
atomic_uint total_iterations;

vector<unsigned> v;

int main(int argc, char *argv[])
{
  total_iterations = 0;
  scramble_vector(1000);
  thread th(find_element, 0);
  find_element(100);
  th.join();

  cout << total_iterations << " total iterations." << endl;
 }
```

When each element is found, a message from within the thread is printed, telling the position in the vector (or iteration) where the element was found:

```
void find_element(unsigned element)
{
  unsigned iterations = 0;

  find_if(begin(v), end(v), [=, &iterations](const unsigned i) -> bool {
    ++iterations;
    return (i==element);
  });
  total_iterations+= iterations;

  cout << "Thread #" << this_thread::get_id() << ": found after " <<
    iterations << " iterations." << endl;
}
```

There's also a common variable, total_iterations, which is updated with the compounded number of iterations applied by both threads. Thus, total_iterations must be atomic to prevent both threads from updating it at the same time. In the preceding example, even if you didn't need to print the partial number of iterations in find_element, you'd still accumulate iterations in that local variable instead of total_iterations, to avoid contention over the atomic variable.

You'll find the preceding sample in the Atomics project in the companion code download. I ran it, getting the following:

```
Thread #8064: found after 168 iterations.
Thread #6948: found after 395 iterations.
563 total iterations.
```

Mut(ual) Ex(clusion) and Locks

The previous section depicted a particular case of mutual exclusion for writing access on primitive types. The <mutex> header defines a series of lockable classes to define critical regions. That way, you can define a mutex to establish a critical region throughout a series of functions or methods, in the sense that only one thread at a time will be able to access any member in this series by successfully locking its mutex.

A thread attempting to lock a mutex can either stay blocked until the mutex is available or just fail in the attempt. In the middle of these two extremes, the alternative timed_mutex class can stay blocked for a small interval of time before failing. Allowing lock attempts to desist helps prevent deadlocks.

A locked mutex must be explicitly unlocked for others to lock it. Failing to do so could lead to an undetermined application

behavior—which could be error-prone, similar to forgetting to release dynamic memory. Forgetting to release a lock is actually much worse, because it might mean that the application can't function properly anymore if other code keeps waiting on that lock. Fortunately, C++11 also comes with locking classes. A lock acts on a mutex, but its destructor makes sure to release it if locked.

The following code (available in the Mutex project in the code download) defines a critical region around a mutex mx:

```
mutex mx;

void funcA();
void funcB();

int main()
{
  thread th(funcA)
  funcB();
  th.join();
}
```

This mutex is used to guarantee that two functions, funcA and funcB, can run in parallel without coming together in the critical region.

The function funcA will wait, if necessary, in order to come to the critical region. In order to make it do so, you just need the simplest locking mechanism—lock_guard:

```
void funcA()
{
  for (int i = 0; i<3; ++i)
  {
    this_thread::sleep_for(chrono::seconds(1));

    cout << this_thread::get_id() << ": locking with wait... " << endl;
    lock_guard<mutex> lg(mx);

    ... // Do something in the critical region.

    cout << this_thread::get_id() << ": releasing lock." << endl;
  }
}
```

The way it's defined, funcA should access the critical region three times. The function funcB, instead, will attempt to lock, but if the mutex is by then already locked, funcB will just wait for a second before again attempting to get access to the critical region. The mechanism it uses is unique_lock with the policy try_to_lock_t, as shown in **Figure 4**.

The way it's defined, funcB will try up to five times to enter the critical region. **Figure 5** shows the result of the execution. Out of the five attempts, funcB could only come to the critical region twice.

Condition Variables

The header <condition_variable> comes with the last facility covered in this article, fundamental for those cases when coordination between threads is tied to events.

In the following example, available in project CondVar in the code download, a producer function pushes elements in a queue:

```
mutex mq;
condition_variable cv;
queue<int> q;

void producer()
{
  for (int i = 0;i<3;++i)
  {
    ... // Produce element

    cout << "Producer: element " << i << " queued." << endl;
    mq.lock();      q.push(i);  mq.unlock();
    cv.notify_all();
  }
}
```

Figure 7 **Synchronization with Condition Variables**

```
Consumer: queue not ready -> going to sleep.
Producer: element 0 queued.
Consumer: fetching 0 from queue.
Consumer: queue not ready -> going to sleep.
Producer: element 1 queued.
Consumer: fetching 1 from queue.
Consumer: queue not ready -> going to sleep.
Producer: element 2 queued.
Producer: element 3 queued.
Consumer: fetching 2 from queue.
Producer: element 4 queued.
Consumer: fetching 3 from queue.
Consumer: fetching 4 from queue.
Consumer: queue not ready -> going to sleep.
Consumer: two consecutive failed attempts -> Exiting.
```

The standard queue isn't thread-safe, so you must make sure that nobody else is using it (that is, the consumer isn't popping any element) when queuing.

The consumer function attempts to fetch elements from the queue when available, or it just waits for a while on the condition variable before attempting again; after two consecutive failed attempts, the consumer ends (see **Figure 6**).

The consumer is to be awoken via notify_all by the producer every time a new element is available. That way, the producer avoids having the consumer sleep for the entire interval if elements are ready.

Figure 7 shows the result of my run.

A Holistic View

To recap, this article has shown a conceptual panorama of mechanisms introduced in C++11 to allow parallel execution in an era where multicore environments are mainstream.

Asynchronous tasks enable a lightweight programming model to parallelize execution. The outcomes of each task can be retrieved through an associated future.

Threads offer more granularity than tasks—although they're heavier—together with mechanisms for keeping separated copies of static variables and transporting exceptions between threads.

As parallel threads act on common data, C++11 provides resources to avoid race conditions. Atomic types enable a trusted way to ensure that data is modified by one thread at a time.

Mutexes help us define critical regions throughout the code—regions to which threads are prevented access simultaneously. Locks wrap mutexes, tying the unlocking of the latter to the lifecycle of the former.

Finally, condition variables grant more efficiency to thread synchronization, as some threads can wait for events notified by other threads.

This article hasn't covered all the many ways to configure and use each of these features, but the reader now has a holistic vision of them and is ready to dig deeper. ■

DIEGO DAGUM *is a software developer with more than 20 years of experience. He's currently a Visual C++ community program manager with Microsoft.*

THANKS to the following technical experts for reviewing this article: *David Cravey, Alon Fliess, Fabio Galuppo and Marc Gregoire*

Windows Phone Data Binding

Jesse Liberty

Virtually every meaningful Windows Phone application has some kind of data, and the ability to connect that data to elements in the UI (the view) is absolutely essential. There are various ways to do this programmatically (assigning values as you go), but one of the more powerful and essential features of XAML programming is the ability to *bind* data to controls.

Here's the good news:

- It isn't difficult to understand.
- It isn't difficult to implement.

To illustrate data binding, you'll create the page shown in **Figure 1**. When it loads, it will be populated from a Person object that you'll create. Each value in the UI will be bound to a property in the Person object, and the actual binding of the data to the controls will be automatic—no C# code needed.

This article discusses:

- Creating a form
- Binding data to the form
- Changing the DataContext
- Using the INotifyPropertyChanged interface
- Binding modes
- Element binding
- Data conversion

Technologies discussed:

Windows Phone

Code download available at:

code.msdn.microsoft.com/mag201203WP7

Getting Started

To get started, create a new Windows Phone application in Visual Studio and name it DataBinding. Begin by creating the class that will serve as the data to which you'll be binding (also known as the DataContext). Right-click on the project and select Add | New | Class and name the class Person.cs.

> Each value in the UI will be bound to a property in the Person object, and the actual binding of the data to the controls will be automatic—no C# code needed.

Person will contain (at least) all the properties that you'll want to display in the view. The class consists of an enumeration and a set of automatic properties, as shown in **Figure 2**.

You can see pretty quickly how these properties will map to the various input controls shown in **Figure 1**. The Booleans can be either CheckBoxes or RadioButtons (depending on whether they're mutually exclusive).

Figure 1 **The Data Binding View in Windows Phone**

Creating the Form

The next task is to create the form you'll use to bind the data. Right-click on the project and select "Open in Expression Blend." As a rule, I tend to create my UI in Expression Blend and write my code in Visual Studio.

Create six rows and two columns in the content grid, and drag on the appropriate input controls. **Figure 3** shows the XAML you'll want to produce.

Binding

Each of the text-entry fields now has its value set using the Binding syntax. For example, to tell the TextBox to bind, identify which of its attributes will require the data—in this case, the Text attribute—and use the binding syntax, as shown earlier.

Bindings are within curly braces, and they use the keyword Binding, typically followed by the name of the property to which you're binding the attribute. For example, this XAML states that the Text for the TextBox will be obtained from a public property named Name:

```
<TextBox
  x:Name="Name"
  TextWrapping="Wrap"
  d:LayoutOverrides="Height"
  Grid.Column="1"
  HorizontalAlignment="Left"
  Width="200"
  VerticalAlignment="Center"
  Text="{Binding Name}" />
```

Similarly, with the checkboxes for facial hair, the IsChecked property is bound to the appropriate property:

```
<CheckBox
  x:Name="Moustache"
  Content="Moustache"
  HorizontalAlignment="Left"
  VerticalAlignment="Center"
  IsChecked="{Binding Moustache}" />
```

You don't yet know what object will have these properties (Name and Moustache). As noted earlier, the object that contains the bindable property is known as the DataContext. It can be just about anything, but in this case you're going to create an instance of the Person class, and then you're going to set that Person object to be the DataContext for the entire view.

Note that you can set a DataContext for a container, in this case the page, and all the view controls inside that container will share that DataContext—though you're free to assign other DataContexts to one or more individual controls.

You can instantiate the Person in the Loaded event handler of the codebehind page. The Loaded event is called once the page is loaded and the controls are initialized, as shown in **Figure 4**.

Now you can set the DataContext for every control in the ContentPanel to be the _currentPerson object you just instantiated (in the Loaded event handler):

```
ContentPanel.DataContext = _currentPerson;
```

Once it knows its DataContext, the TextBox can resolve the Name property and obtain the value ("Jesse") to display. The same is true for all the other controls, each bound to a property in the new Person object.

Run the application and you should see all the fields bound appropriately.

Changing the DataContext

To drive home the relationship between the binding and the display, let's create a number of Person objects and display them one by one. To do this, modify MainPage.xaml.cs to create a list of (randomly created) Persons and then iterate through the list with a new "Next" button on the UI, which you should add to the bottom row:

```
<Button
  Name="Next"
  Content="Next"
  Grid.Row="5"
  HorizontalAlignment="Center"
  VerticalAlignment="Center" />
```

> As a rule, I tend to create my UI in Expression Blend and write my code in Visual Studio.

Here's the modified code to interact with the Next button:

```
void MainPage_Loaded( object sender, RoutedEventArgs e )
{
  SetDataContext();
  Next.Click += Next_Click;
}

private void SetDataContext()
{
  ContentPanel.DataContext = GeneratePerson();
}

void Next_Click( object sender, RoutedEventArgs e )
{
  SetDataContext();
}
```

Notice both the page loaded event and the click event handler for the Next button must set the DataContext, so I've factored that out into a separate method: SetDataContext. That method, in turn, calls the GeneratePerson method, whose job is to create a Person at random.

Figure 2 **The Person Class**

```
public class Person
{
  public enum Sex
  {
    Male,
    Female,
  }
  public string Name { get; set; }
  public bool Moustache { get; set; }
  public bool Goatee { get; set; }
  public bool Beard { get; set; }
  public Sex WhichSex { get; set; }
  public double Height { get; set; }
  public DateTime BirthDate { get; set; }
  public bool Favorite { get; set; }}
```

Figure 3 **The XAML to Create the Form**

```
<Grid
  x:Name="ContentPanel"
  Grid.Row="1"
  Margin="24,0,0,0">
  <Grid.ColumnDefinitions>
    <ColumnDefinition
      Width="0.384*" />
    <ColumnDefinition
      Width="0.616*" />
  </Grid.ColumnDefinitions>
  <Grid.RowDefinitions>
    <RowDefinition
      Height="0.1*" />
    <RowDefinition
      Height="0.1*" />
    <RowDefinition
      Height="0.1*" />
    <RowDefinition
      Height="0.1*" />
    <RowDefinition
      Height="0.1*" />
    <RowDefinition
      Height="0.1*" />
    <RowDefinition
      Height="0.2*" />
  </Grid.RowDefinitions>
  <TextBlock
    x:Name="NamePrompt"
    TextWrapping="Wrap"
    Text="Name"
    Grid.Row="0"
    HorizontalAlignment="Left"
    VerticalAlignment="Center" />
  <TextBlock
    x:Name="SexPrompt"
    Grid.Row="2"
    TextWrapping="Wrap"
    HorizontalAlignment="Left"
    VerticalAlignment="Center"
    Text="Sex" />
  <TextBlock
    x:Name="HeightPrompt"
    TextWrapping="Wrap"
    Text="Height, StringFormat=F3"
    HorizontalAlignment="Left"
    Grid.Row="3"
    d:LayoutOverrides="Height"
    VerticalAlignment="Center" />
  <TextBlock
    x:Name="FavoritePrompt"
    TextWrapping="Wrap"
    Text="Favorite"
    HorizontalAlignment="Left"
    Grid.Row="4"
    d:LayoutOverrides="Height"
    VerticalAlignment="Center" />

  <TextBox
    x:Name="Name"
    TextWrapping="Wrap"
    d:LayoutOverrides="Height"
    Grid.Column="1"
    HorizontalAlignment="Left"
    Width="200"
    VerticalAlignment="Center"
    Text="{Binding Name}" />
  <StackPanel
    x:Name="BeardStackPanel"
    Grid.ColumnSpan="2"
    Grid.Row="1"
    Orientation="Horizontal">
    <CheckBox
      x:Name="Moustache"
      Content="Moustache"
      HorizontalAlignment="Left"
      VerticalAlignment="Center"
      IsChecked="{Binding Moustache}" />
    <CheckBox
      x:Name="Goatee"
      Content="Goatee"
      IsChecked="{Binding Goatee}" />
    <CheckBox
      Content="Beard"
      IsChecked="{Binding Beard}"/>
  </StackPanel>
  <StackPanel
    x:Name="SexStackPanel"
    Grid.Column="1"
    Grid.Row="2"
    Orientation="Horizontal">
    <RadioButton
      x:Name="Male"
      Content="Male"
      IsChecked="True"
      GroupName="Sex" />
    <RadioButton
      x:Name="Female"
      Content="Female"
      GroupName="Sex" />
  </StackPanel>
  <StackPanel
    x:Name="HeightStackPanel"
    Grid.Column="1"
    Grid.Row="3"
    Orientation="Horizontal">

    <TextBlock
      TextWrapping="Wrap"
      Text="{Binding Height}"
      VerticalAlignment="Center"
      HorizontalAlignment="Left"
      Margin="0,0,0,0" />
    <TextBlock
      VerticalAlignment="Center"
      HorizontalAlignment="Left"
      Margin="5,0,0,0"
      Text="meters" />

  </StackPanel>

  <ToggleButton
    x:Name="Favorite"
    Content="Favorite"
    Grid.Column="1"
    Grid.Row="4"
    d:LayoutOverrides="Width, Height"
    HorizontalAlignment="Left"
    VerticalAlignment="Center"
    IsChecked="{Binding Favorite}" />
</Grid>
```

You can now make all the changes in the codebehind. First, stop hardwiring the current Person, and instead set it by calling the GeneratePerson method.

Generating a Random Person

Here's the entire GeneratePerson method; you'll see that I've factored out the task of choosing true versus false into a method called FlipCoin:

```
private Person GeneratePerson()
{
  var newPerson = new Person
  {
    Beard = FlipCoin(),
    Favorite = FlipCoin(),
    Goatee = FlipCoin(),
    Height = _rand.NextDouble() + 1,
    Moustache = FlipCoin(),
    Name = names[_rand.Next(0, names.Count - 1)]
  };
  return newPerson;
}
```

FlipCoin uses the random number generator to return true 50 percent of the time:

```
private  bool FlipCoin()
{
  return _rand.Next( 1, 3 ) % 2 == 0;
}
```

Finally, to pick a name, create a list of half a dozen names that can be assigned to men or women, and use the random number generator to pick an offset into the list:

```
private readonly List<string> names = new List<string>()
{
  "Stacey",
  "Robbie",
  "Jess",
  "Robin",
  "Syd",
  "J.J.",
  "Terri",
  "Moonunit",
};
```

Run the application and click the Next button. As each Person object is created it's set as the DataContext and its properties are bound to the UI controls.

INotifyPropertyChanged

What happens if one of the properties on your Person object changes? This could easily happen if the object is contained in a database and other users have access to the same object. You would want your UI to be updated.

For this to work, your class (the DataContext) must implement INotifyPropertyChanged—a simple interface that allows each property to notify the UI when its value changes. It's common to create a helper method that checks to make sure the event has at least one

Figure 4 **The Loaded Event Is Called Upon the Page Loading**

```
private Person _currentPerson;
private Random _rand = new Random();

public MainPage()
{
  InitializeComponent();
  Loaded += MainPage_Loaded;
}

void MainPage_Loaded( object sender, RoutedEventArgs e )
{
  _currentPerson = new Person
  {
    Beard = false,
    Favorite = true,
    Goatee = false,
    Height = 1.86,
    Moustache = true,
    Name = "Jesse",
    WhichSex = Person.Sex.Male
  };
}
```

Figure 5 **The INotifyPropertyChanged Interface**

```
public class Person : INotifyPropertyChanged
{
  public string _name;
  public string Name
  {
    get { return _name; }
    set
    {
      _name = value;
      PropChanged( "Name" );
    }
  }

// Other properties


  public event PropertyChangedEventHandler PropertyChanged;
  private void PropChanged(string propName)
  {
    if (PropertyChanged != null)
    {
      PropertyChanged( this, new PropertyChangedEventArgs( propName ) );
    }

  }
}
```

method registered to it. If so, the helper method raises the event, passing in the name of the property that updated.

To see this at work, add a new button to the UI named Change. When you click the Change button, change the Name property to "Jacob":

```
void Change_Click( object sender, RoutedEventArgs e )
{
  _currentPerson.Name = "Jacob";
}
```

This will have no effect unless Person implements INotifyPropertyChanged and the Name property raises the PropertyChanged event, as shown in **Figure 5**.

With this in place, when you click the Change button the name shown in the UI will change to the new name in the object (Jacob).

> Some properties won't bind properly to a given UI control, or you might want greater control over how the value is displayed.

Two-Way Binding

What if the user interacts with the UI and changes a value directly (for example, types a new name into the Name TextBox)? You'll (usually) want that change pushed back to the underlying data (the DataContext object). To do that, you'll use two-way binding.

To modify the program to use two-way binding on the Name property, find the Name binding and modify it to this:

```
<TextBox
  x:Name="Name"
  TextWrapping="Wrap"
  d:LayoutOverrides="Height"
  Grid.Column="1"
  HorizontalAlignment="Left"
  Width="200"
  VerticalAlignment="Center"
  Text="{Binding Name, Mode=TwoWay}" />
```

There are three modes for binding:

1. One-time binding means that the data is bound but then never updated, even if the data is updated by the user.
2. One-way binding is the default; data is pulled from the source to the UI but not pushed back to the source.
3. Two-way binding allows data to be pulled from the source and pushed back to the source if modified in the UI.

Element Binding

Change the row for the Next button to row six and drag a Slider control onto row five. There are a number of settings that are important in a slider, including:

- Minimum
- Maximum
- Value
- LargeChange
- SmallChange

You can see these reflected in the code in **Figure 6**.

Minimum and Maximum set the range of the slider. In this case, because I'm using percentages, I've set them to 0 and 100, respectively.

Value is the current value of the slider, and will be: Minimum <= value <= Maximum

LargeChange and SmallChange are used much as they are in scrollbars; they indicate what clicking into the slider will do and what using another control (perhaps arrows) to incrementally move the slider will do, respectively.

Figure 6 **Adding a Slider**

```
<Slider
  x:Name="Likability"
  Grid.Row="5"
  Grid.Column="0"
  BorderBrush="White"
  BorderThickness="1"
  Background="White"
  Foreground="Blue"
  LargeChange="10"
  SmallChange="1"
  Minimum="0"
  Width="199"
  Maximum="100"
  Value="50"
  Height="90" />
```

Figure 7 **Binding the Value Property of a TextBlock**

```
<StackPanel
  x:Name="LikeabilityPercentStackPanel"
  Grid.Row="5"
  Grid.Column="1"
  Orientation="Horizontal">
  <TextBlock
    Text="Likeability: "
    HorizontalAlignment="Left"
    VerticalAlignment="Center"
    Margin="20,0,5,0" />
  <TextBlock
    x:Name="SliderValue"
    Text="{Binding Value, ElementName=Likeability, StringFormat=F3}"
    HorizontalAlignment="Left"
    VerticalAlignment="Center"
    Margin="5,0,0,0"/>
  <TextBlock
    Text="%"
    HorizontalAlignment="Left"
    VerticalAlignment="Center" />
</StackPanel>
```

Setting up the TextBlocks

In the right-hand column, you'll use three TextBlocks; the first and third as fixed labels (with the values "Likeability:" and "%," respectively). The middle TextBlock displays a numeric representation of the value of the slider.

To accomplish this, bind the Text Value property of the middle TextBlock to the Value property of the slider, identifying which element you're binding to with the keyword ElementName, as shown in **Figure 7**.

Run the program. As the Slider is adjusted, the value in the TextBlock is updated instantly.

> Data binding allows you to create powerful Windows Phone applications that reliably manage the relationship between underlying data and the controls and views that display that data.

Data Converters

Some properties won't bind properly to a given UI control, or you might want greater control over how the value is displayed. As a simple example, let's display the user's birth date by moving the buttons down a row and inserting a row with a prompt ("Birth date") and the value of the Person's BirthDate.

To do this, you have to modify the GeneratePerson method in MainPage.xaml.cs to generate a valid BirthDate, which you do by adding this line to create a random BirthDate within the past 20 years:

```
BirthDate = DateTime.Now - TimeSpan.FromDays(_rand.Next(1,365*20)),
```

If you just bind to the BirthDate property, you'll see the birth date and time. But you don't want the time—just the date, in short date format. To accomplish this, you need a DataConverter.

DataConverters are classes that implement IValueConverter. This interface requires two methods, shown in **Figure 8**.

In this case, you only need the first of the two (the second will never be called). The method is pretty simple to implement; make sure the target type is string and the value type is DateTime. If so, take the value, cast it to a DateTime and then call ToShortDateString on it, as show in **Figure 9**.

With this in place, you need a way for the XAML to access the value converter; you can accomplish this by making the converter a resource. Open App.xaml and add a namespace for your converter, based on the namespace of your application:

```
xmlns:mine="clr-namespace:DataBinding">
```

Next, in the same file, find the <Application.Resource> section and add a resource for your value converter:

```
<Application.Resources>
  <mine:DateConverter   x:Key="dateConverter" />
</Application.Resources>
```

You can now use the key in your XAML file. Update the Binding for the BirthDate to use the resource:

```
<TextBlock
  Grid.Row="6"
  Grid.Column="1"
  VerticalAlignment="Center"
  Text="{Binding BirthDate, Converter={StaticResource dateConverter}}" />
```

Run the program and you should see the date shown in short date format.

Figure 8 **The IValueConverter Interface**

```
public object Convert(
  object value,
  Type targetType,
  object parameter,
  System.Globalization.CultureInfo culture )
{
  throw new NotImplementedException();
}

public object ConvertBack(
  object value,
  Type targetType,
  object parameter,
  System.Globalization.CultureInfo culture )
{
  throw new NotImplementedException();
}
```

Figure 9 **The Convert Method to Shorten a DateTime**

```
public object Convert(
  object value,
  Type targetType,
  object parameter,
  System.Globalization.CultureInfo culture )
{
  if (targetType == typeof( string ) &&
    value.GetType() == typeof( DateTime ))
  {
    return (( DateTime ) value).ToShortDateString();
  }
  else  // Unable to convert
  {
    return value;
   }
}

public object ConvertBack(
  object value,
  Type targetType,
  object parameter,
  System.Globalization.CultureInfo culture )
{
  throw new NotImplementedException();
}
```

Powerful Applications

Data binding allows you to create powerful Windows Phone applications that reliably manage the relationship between underlying data and the controls and views that display that data. In this article you saw how to create simple data binding and two-way data binding, how to bind to elements and how to use data converters to massage the data into the format you want. ■

JESSE LIBERTY *is a senior developer-community evangelist on the Windows Phone team. Liberty hosts the popular Yet Another Podcast (jesseliberty.com/podcast), and his blog (jesseliberty.com) is required reading. He's the author of numerous best-selling books, including "Programming Reactive Extensions and LINQ" (Apress, 2011) and "Migrating to Windows Phone" (Apress, 2011). You can follow Liberty on Twitter at twitter.com/JesseLiberty.*

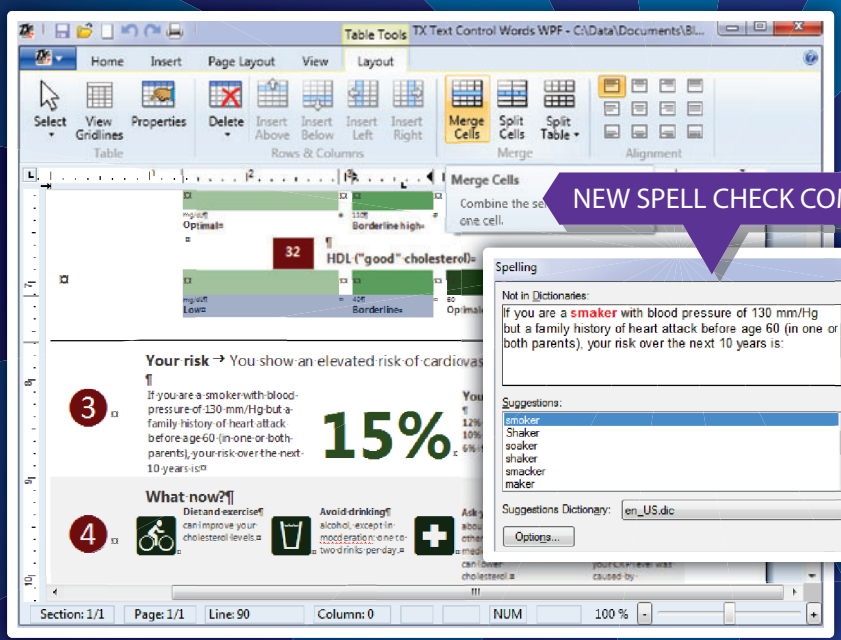
THANKS to the following technical expert for reviewing this article: *Jason Shaver*

WINDOWS FORMS | WPF | ASP.NET | ACTIVEX

WORD PROCESSING COMPONENTS

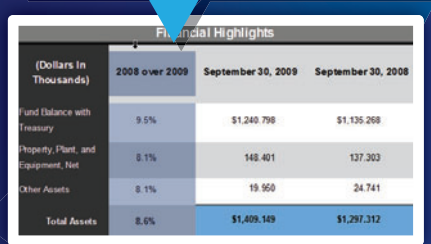
VERSION 17.0 RELEASED

MEET US AT Microsoft Visual Studio CONNECTIONS MARCH 26-29, 2012, LAS VEGAS

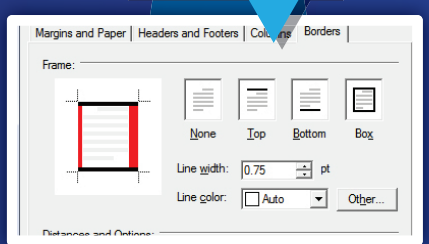


NEW SPELL CHECK COMPONENT

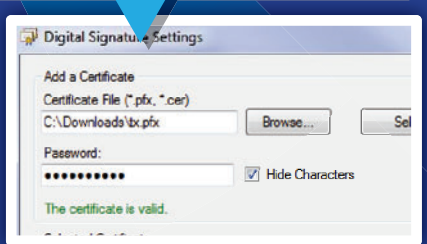
CELL MERGING, COL SELECTION



PAGE BORDERS



DIGITAL SIGNATURES IN PDF



TX TEXT CONTROL® 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

Adding HTML5 Drag and Drop to SharePoint Lists

Andrey Markeev

Microsoft SharePoint is an enterprise platform with a long history and vast variety of features, which is why it can't always react quickly enough to follow emerging Web technology trends. Despite a wide enterprise adoption of SharePoint and a huge effort to provide a broad number of features, SharePoint still lags behind modern CMS products in terms of immersive UIs, such as HTML5 and CSS3.

In my opinion, HTML5 is not only a hot new technology, but it truly has many practical benefits: it's easy, convenient and rich—and it's supported, more or less, by all the modern browsers (including mobile device browsers). Additionally, HTML5 and JavaScript are becoming major technologies for desktop programming in Windows.

This article discusses:

- Customizing existing functionality: XsltListViewWebPart
- Generating code with SharePoint Designer
- Using conditional formatting
- Making templates reusable
- Writing JavaScript handlers
- Building a deployable solution

Technologies discussed:

SharePoint 2010, HTML5, JavaScript

Code download available at:

code.msdn.microsoft.com/mag201203HTML5

So HTML5 definitely deserves its place in SharePoint to make portals much easier to use. And improving SharePoint interfaces can really help business users by enabling them to work better and faster.

Unfortunately, SharePoint doesn't have any built-in HTML5 goodness, but what it does have is great flexibility. In this article, I'm going to demonstrate how easy it is to add HTML5 drag-and-drop support to SharePoint—and how smooth it can make the standard interface, as shown in **Figure 1**.

To implement this, I'll use one of the essential SharePoint building blocks, which is also one of my favorite SharePoint tools—the XsltListViewWebPart and its XSL transformations (see the MSDN Library page at bit.ly/wZVSFx for details).

Why Not a Custom Web Part?

As always, when it comes to implementation, SharePoint offers a wide range of possibilities, and it's exceedingly important to pick the one that will serve you best.

For the HTML5 drag-and-drop challenge, considering that drag and drop is mainly for managing data, many SharePoint developers would probably prefer to build a custom Web Part, which in this case acts just like an ordinary ASP.NET control: the data is stored in a standard SharePoint list, retrieved through the object model or SPDataSource control, and rendered with the help of ASCX markup and ASP.NET controls.

Simple, clear, plain ... but the best choice?

Two years ago I thought so. Today, I'd prefer to customize XsltListViewWebPart using its XSL transformations. Why did I change my mind?

Starting with SharePoint 2010, almost all kinds of list views (with the single exception of Calendars) are displayed through this very Web Part. Just imagine: all these data types, all these different views and styles and list types, all this great variety of data is rendered using XsltListViewWebPart and its XSL transformations. It's a flexible and powerful tool.

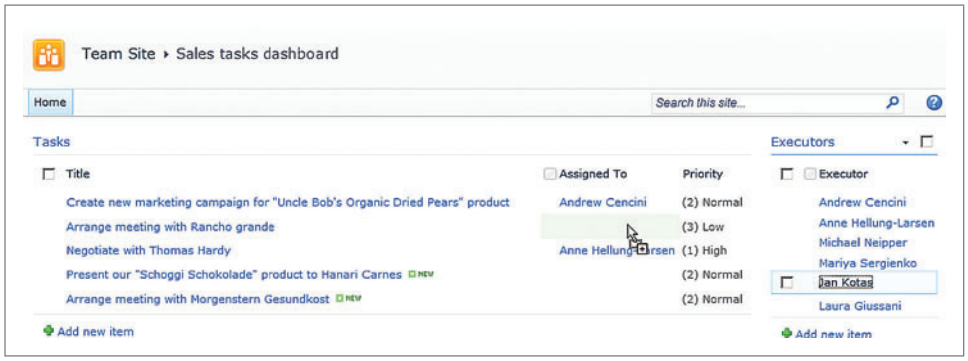


Figure 1 **Drag and Drop in SharePoint**

If you decide to jump in and build your own custom Web Part to render some HTML5 markup for displaying list data, you'll lose all of the built-in features. And based on my experience, that's a huge loss. By the way, I haven't seen a single custom Web Part yet that didn't, at the very least, end up implementing half of the out-of-the-box XsltListViewWebPart features.

> What I'm going to do is inject HTML5 drag and drop into SharePoint list views.

So, my plan is to reuse existing functionality rather than create a similar custom Web Part that would probably be much worse in terms of flexibility and power.

In fact, XsltListViewWebPart includes a bunch of useful features. It's integrated into SharePoint Designer, it supports all possible Web Part connections and it displays all the SharePoint data types properly. It supports grouping, subtotals, paging, item context menus, inline editing, item selections, presence indicators and more. It has a contextual Ribbon interface, provides a UI for sorting and filtering, offers some basic view styles and more again. In sum, XsltListViewWebPart has a great many useful features that would be very hard to re-implement using the custom Web Part approach.

XsltListViewWebPart

XsltListViewWebPart provides many integration points for developers: a programmatic interface, a CAML interface and, of course, XSL transformations in conjunction with parameter bindings. And don't forget, all these objects and properties also have their representations in the Client Object Model, so you can access your XsltListViewWebPart even from JavaScript or Silverlight.

So, XsltListViewWebPart is really a powerful tool. True, all this SharePoint-specific XML (or XSLT) looks a bit scary at initial glance, but there are some "life hacks" I'm going to show you that will help you puzzle it out.

The Scenario

Before I dive into the implementation details, let me describe the overall scenario.

What I'm going to do is inject HTML5 drag and drop into SharePoint list views to enable users to drag cells from one list to another list. My example will use a Tasks list and an Executors list, so the Project Manager can easily assign and reassign tasks, dragging executors to corresponding Tasks list cells.

As you might know, HTML5 introduces several new attributes for drag and drop, the most important of which is the "draggable" attribute. There are also a number of events for handling various stages of the drag-and-drop process. Handler functions for these events can be attached using corresponding attributes, such as "ondragstart," "ondragend" and so forth. (For details, read the World Wide Web Consortium [W3C] HTML5 specification draft, Chapter 7.6, at bit.ly/INL0FO.)

For my example, this means I just need to use XSLT to add some basic attributes to certain list view cells, and probably some additional custom attributes to attach the data values (that will be conveyed by dragging). Eventually, I'll need to provide the corresponding JavaScript code for the handler functions.

First Steps

I need two lists. I can create a Tasks list from the standard Tasks template, or I can just create a custom list and add some columns, including an obligatory "Assigned To" site column. I create a second list, Executors, as a custom list, adding "Executor" as a column of type "Person or group," making it required, indexed and unique.

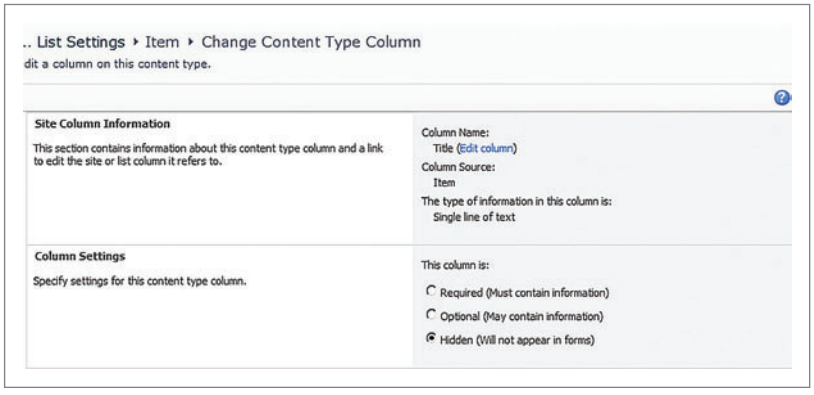


Figure 2 **Making the Title Column Hidden in SharePoint List Settings**

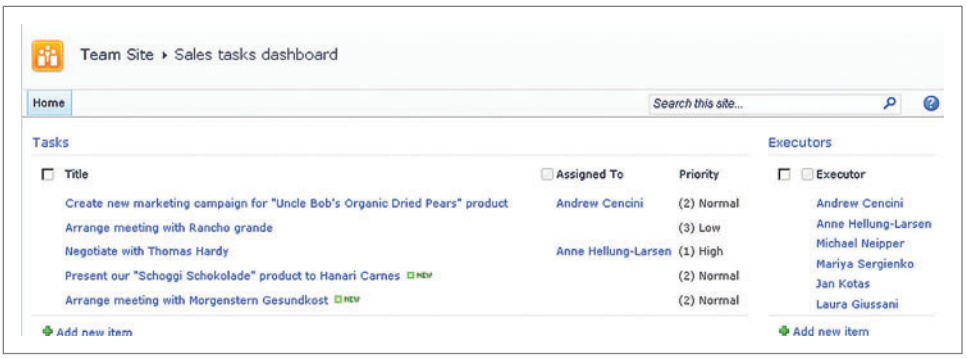


Figure 3 **Adding the Lists to the SharePoint Dashboard**

The Executors list should display only user names; thus it doesn't actually need the standard "Title" column. To hide this column, I go to list settings, enable management of content types, then go to the "Item" content type, click on the "Title" column and make the column hidden, as shown in **Figure 2**.

I filled these lists with sample data and then created a Web Part page for my dashboard, where I added these lists side-by-side (Tasks on the left and Executors on the right), as shown in **Figure 3**.

OK, now I have the lists and I have the data. Now it's time for the actual implementation of the drag-and-drop functionality.

SharePoint Designer

Microsoft SharePoint Designer is a completely free tool for rapid development of SharePoint applications. SharePoint 2010 is greatly improved as compared with SharePoint 2007, and now it's exceedingly useful, even for developers. The idea is that you can use the SharePoint Designer GUI to generate some really complex XSLT code and then just copy and paste the generated code into your Visual Studio project instead of writing the typo-prone and not always well-documented XML/XSLT by hand. I often use this trick in real-world projects and, I promise you, it saves plenty of time.

I open SharePoint Designer and navigate to the dashboard page I created earlier. I select a cell in the Assigned To column (right-click and choose Select | Cell). Now, the magic: in the status bar, I see the path to the XSL template (and to the corresponding HTML tag within this template) that's responsible for displaying this particular cell (see **Figure 4**).

This information can be very useful for determining which XSL template to override in order to change the cell markup. You can find the original code for the templates in the 14/TEMPLATE/LAYOUTS/XSL folder, and use it in your own XSLT files or in the <Xsl> tag of the XsltListViewWebPart.

But I don't need to deal with these huge and complicated XSLT files to achieve my goal. Instead, I can use the SharePoint Designer conditional formatting feature, which is designed to highlight certain rows or cells with special formatting, based on particular conditions. You don't need any special skills to use this feature; the GUI makes it easy. But behind the scenes, it's all implemented with XSLT. Thus, SharePoint Designer includes a kind of ready-to-use graphical XSLT generator, and I'm going to use it now for my own needs.

I select a cell, click the Conditional Formatting button on the Ribbon and then select Format Column, as shown in **Figure 5**.

Next, I create an unlikely condition, ID equal to zero, as shown in **Figure 6**.

Then I click the Set Style button and select some random style (such as "text-decoration: underline"). I press OK and switch to the Code View tab, where I locate the generated code; it is, of course, inside the <Xsl> tag of the XsltListViewWebPart control.

XSL Transformations

Now I'm ready to modify the markup of "Assigned To" cells. The "Assigned To" column is the "data acceptor" where I'll drag executors, so I need to provide the "ondragover," "ondragenter," "ondragleave" and "ondrop" attributes, which will point to the corresponding JavaScript event handler functions.

The code generated by SharePoint Designer in the previous paragraph contains the XSL template with the following signature:

```
<xsl:template name="FieldRef_printTableCell_EcbAllowed.AssignedTo"
  match="FieldRef[@Name='AssignedTo']" mode="printTableCellEcbAllowed"
  ddwrt:dvt_mode="body" ddwrt:ghost="" xmlns:ddwrt2="urn:frontpage:internal">
```

As you might know, XSL templates can call each other, either by name or by condition. The first type of call is performed using the "xsl:call-template" element, and it's very similar to a function call—such as what you'd use in C#, for example.

The second option is preferable and much more flexible: by using the "xsl:apply-templates" element, you can specify the mode and the parameter (which is selected using XPath so it can actually contain many elements), without specifying any particular

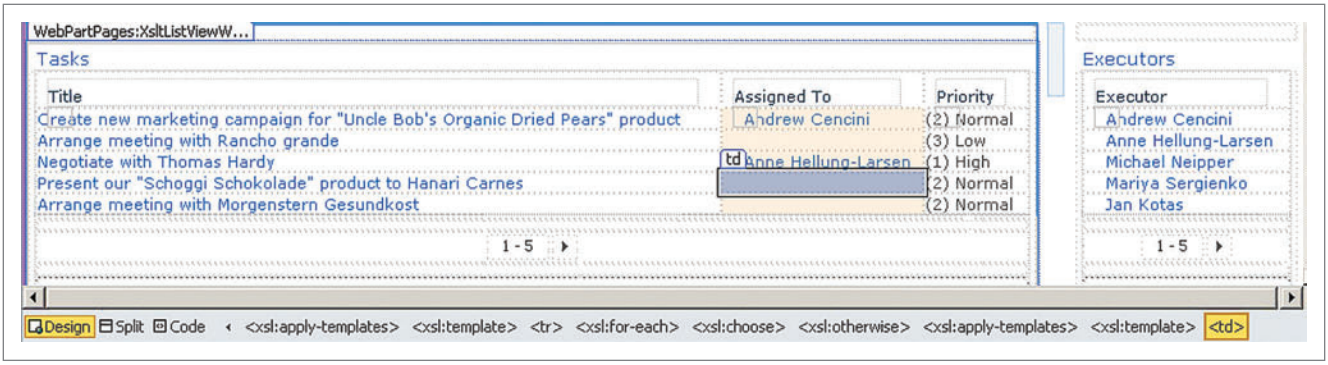


Figure 4 **The Path to the Current XSL Template in SharePoint Designer**

template name. For each parameter element, the corresponding template will be matched using the "match" attribute. You can think of this approach as something similar to overloads in C#.

As you can see in the preceding code, this template will match "FieldRef" elements, where the Name attribute is equal to "AssignedTo." Also, the "mode" attribute of the corresponding xsl:apply-template call must be equal to "printTableCellEcbAllowed." So this template is essentially an overload for the standard function that displays fields' values. And this overload will match only the "Assigned To" field values.

Now let's take a look at what's inside this template, as shown in **Figure 7** (some code was removed for clarity).

As you see, the template contains two xsl:param elements, one <td> element, several xsl:attribute elements and an xsl:apply-templates element, which will cause some lower-level templates to be applied.

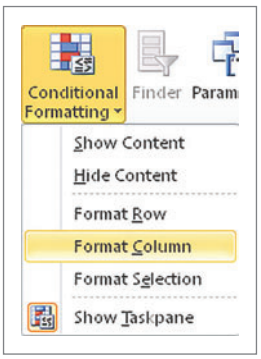


Figure 5 **Setting Conditional Formatting in SharePoint Designer**

> SharePoint Designer 2010 is greatly improved compared with SharePoint Designer 2007, and now it's exceedingly useful, even for developers.

To achieve my drag-and-drop goal, I just add the drag-and-drop attributes to the <td> element, like so:

```
<td ondragover="return UserDragOver(event, this)" ondragenter=
  "return UserDragOver(event, this)" ondragleave=
  "UserDragLeave(event, this)" ondrop="UserDrop(event, this)">
```

Pretty simple, isn't it?

Alternatively, if you have jQuery deployed in your SharePoint environment, you might consider attaching JavaScript event handlers using the jQuery .on method.

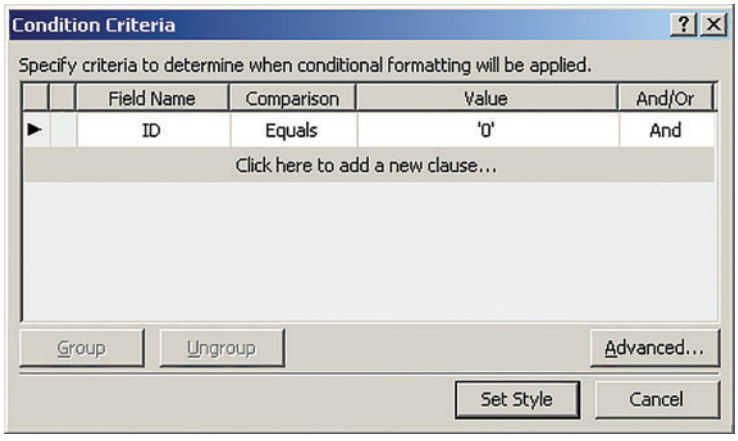


Figure 6 **The Conditional Formatting Dialog in SharePoint Designer**

The markup for the Assigned To column is ready (I'll write the handlers a bit later). Now it's time to customize the Executors list.

I switch back to the Design View tab, select a cell in the Executors column and repeat the conditional formatting trick for generating the XSLT code. Then I add the onstartdrag attribute to ensure that the drag and drop can properly start (I don't need the draggable attribute here, because "Person or Group" field values are rendered as links and, according to the specification, links have the draggable attribute set to "true" by default):

```
<td ondragstart="UserDragStart(event, this)">
```

Cool. But how will I track the data? How do I determine which executor is being dragged? Obviously, I need his login name or, better, his ID. Parsing the ID from inside the TD element is unreasonably complicated, in my opinion.

The point here is that in XSLT, for any field of type Person or Group, the user ID is available and can be easily retrieved with a simple XPath query.

In this query, I need to point to the current element's values. The current element is usually referenced as the $thisNode parameter in all standard XsltListViewWebPart templates. To retrieve the user's ID, you point to the attribute of the $thisNode parameter, with name equal to the name of the Person or Group column, with ".id" concatenated to its end.

So here's my query:

```
<td ondragstart="UserDragStart(event, {$thisNode/Executor.id}, this)">
```

The curly brackets are used for including the XPath expression right in the attribute value.

The markup is ready and can actually be used right away, but it would probably be a good idea to work with this code a little more to make it more reusable.

Making Templates Reusable

You might have noticed that the templates are tightly bound to specific column names and that these templates are intended only for particular lists. But it's actually very simple to modify these templates so you can reuse them for other lists with other column names.

To start, if you examine the template signature I presented earlier, you'll see the following attribute:

```
match="FieldRef[@Name='AssignedTo']"
```

Obviously, this binds the template to the Assigned To column. To make this template a bit broader-based, you can replace the name binding with a type binding, so that any Person or Group column will match. So be it! Here's the code:

```
match="FieldRef[@Type='User']"
```

The same modification should be applied to the second template, where the FieldRef element is matched to the "Executor" field internal name.

Now, because I can have any column of type Person or Group and any list, I need to pass some additional information to my JavaScript handlers. When drag and drop is performed, I need to update the value of the Assigned

Figure 7 **Inside the XSL Template**

```
<xsl:template match="FieldRef[@Name='AssignedTo']"
mode="printTableCellEcbAllowed" ...>
  <xsl:param name="thisNode" select="."/>
  <xsl:param name="class" />
  <td>
    <xsl:attribute name="style">
      <!-- ... -->
    </xsl:attribute>
    <xsl:if test="@ClassInfo='Menu' or @ListItemMenu='TRUE'">
      <xsl:attribute name="height">100%</xsl:attribute>
      <xsl:attribute name="onmouseover">OnChildItem(this)</xsl:attribute>
    </xsl:if>
    <xsl:attribute name="class">
      <!-- ... -->
    </xsl:attribute>

    <xsl:apply-templates select="." mode="PrintFieldWithECB">
      <xsl:with-param name="thisNode" select="$thisNode"/>
    </xsl:apply-templates>
  </td>
</xsl:template>
```

To column in the Tasks list, so I need to know the name of the column and GUID of the list.

As I mentioned previously, SharePoint XSLT has some standard parameters that are available globally. One such parameter is $List, which stores the current list's GUID. And the internal name of the field can be easily retrieved from the matched FieldRef element.

So, I'm going to pass the list GUID and the column internal name to the UserDrop handler, as follows (I'm omitting the "ondragenter," "ondragover" and "ondragleave" attributes for clarity):

```
<td ... ondrop="UserDrop(event, this, '{$List}', '{./@Name}'">
```

The "." points to the current FieldRef element (which was matched previously by the template).

SharePoint Designer includes a kind of ready-to-use graphical XSLT generator.

Next, I need to get rid of the "Executor" string in the id parameter in the Executors list XSLT using the following code:

```
<td ondragstart="UserDragStart(event, {$thisNode/@*[name()=
  concat(current()/@Name, '.id')]}, this)">
```

The templates are now ready and reusable, and now I'm going to write the corresponding JavaScript code to implement the handlers.

Writing JavaScript Handlers

Although there are four different handlers to write, most of them are primitive and they're all rather obvious.

Usually I'd recommend placing this code in a separate JavaScript file. And it generally would be a good idea to use Visual Studio to create it, as you'd get the benefit of IntelliSense there. In some circumstances, however, it's reasonable to put this code *inside* the XSLT, overriding the root template (match="/") for this purpose. This lets you use some XSLT variables and parameters inside your JavaScript code and it also means you don't have to be concerned about deploying JavaScript files.

So let's look at the code for the handlers—DragStart, DragEnter, DragOver, DragLeave and Drop.

In the UserDragStart handler, you need to initialize the data transfer. This means you need to store dragged data in a special HTML5 DataTransfer object, like this:

```
function UserDragStart(e, id, element) {
  e.dataTransfer.effectAllowed = 'copy';
  e.dataTransfer.setData('Text', id + '|' + element.innerHTML);
}
```

Note that the ID of the user is not the only part of the data that's transferred. I also added the inner HTML of the <td> element, to avoid having to refresh the page after dropping the data (see the UserDrop handler code in **Figure 8** for the details).

For the DragEnter and DragOver events in this case, the handlers are identical so I'm using a single function for them. You should indicate in these events that the user can drop his data here. According to the specification, for this purpose you should call the event.preventDefault method (if available), and then return false.

The DragEnter/DragOver handler is the perfect place to apply custom styles to the drag-and-drop placeholder to notify the user that he actually can drop his dragged data. To simplify the example, I'll use inline CSS styles, but in a real-world project I'd recommend using CSS classes (see the commented lines shown in **Figure 9**).

Obviously, in DragLeave I need to remove the previously applied styles, as shown in **Figure 9**.

During the Drop event in **Figure 8**, two important actions need to be performed:

Figure 8 **The Drop Event Handler**

```
function UserDrop(e, toElement, listGuid, columnName) {
  // Terminate the event processing
  if (e.stopPropagation)
      e.stopPropagation();
  // Prevent default browser action
  if (e.preventDefault)
      e.preventDefault();

  // Remove styles from the placeholder
  toElement.style.backgroundColor = '';
  //toElement.className = '';

  // iid attribute is attached to tr element by SharePoint
  // and contains ID of the current element
  var elementId = toElement.parentNode.getAttribute('iid').split(',')[1];

  // Transferred data
  var data = e.dataTransfer.getData('Text');
  var userId = data.split('|')[0];
  var userLinkHtml = data.split('|')[1];

  // Setting value of the field using SharePoint
  // EcmaScript Client Object Model
  var ctx = new SP.ClientContext.get_current();
  var list = ctx.get_web().get_lists().getById(listGuid);
  var item = list.getItemById(elementId);
  item.set_item(columnName, userId);
  item.update();

  // Asynchronous call
  ctx.executeQueryAsync(
    function () { toElement.innerHTML = userLinkHtml; },
    function (sender, args) { alert('Drag-and-drop failed.
      Message: ' + args.get_message()) }
    );

  return false;
}
```

1. Apply the changes. Using the SharePoint EcmaScript Client Object Model, set the field value to the transferred user ID.
2. Replace the inner HTML code in the current cell (TD) with code from the transferred one to make the changes appear without refreshing the page.

Now all handlers are implemented and you can deploy the JavaScript. Actually, you can do this in many different ways: customize the master page, use delegate control, create a custom action with Location set to "ScriptLink" or, as I mentioned earlier, include JavaScript right in the XSLT.

The simplest way here is to customize the master page file using SharePoint Designer, as it doesn't require any special skills. But because you're a developer, chances are you'd prefer to gather together all these JavaScript and XSLT customizations and create a deployable solution. OK, let's do it!

Building a Deployable Solution

To create a ready-to-use solution you can send to your customers, you need to perform some very simple steps:

1. Open Visual Studio and create an Empty SharePoint Project. Then select "deploy as a farm solution" in the project settings dialog.
2. Add a SharePoint "Layouts" Mapped Folder to your project, and add three new files to it: UserDragHandlers.js, UserDragProvider.xsl and UserDragConsumer.xsl.
3. Paste the previously created JavaScript code and the XSLT code (generated in SharePoint Designer) into corresponding files.
4. Add an Empty Element project item, open Elements.xml and paste the following code there:

```
<CustomAction ScriptSrc="/_layouts/SPDragAndDrop/
UserDragHandlers.js" Location="ScriptLink" Sequence="10" />
```

This will deploy the link to your JavaScript file to all site pages.

5. Finally, add an Event Receiver to Feature1 (which was created automatically when you added the Empty Element project item), uncomment the FeatureActivated method and paste the following code in it:

```
var web = properties.Feature.Parent as SPWeb;
SPList list;
SPView view;

list = web.Lists["Tasks"];
view = list.DefaultView;
view.XslLink = "../SPDragAndDrop/UserDragConsumer.xsl";
view.Update();

list = web.Lists["Executors"];
view = list.DefaultView;
view.XslLink = "../SPDragAndDrop/UserDragProvider.xsl";
view.Update();
```

And, yes, that's all—you're done! Pretty simple, isn't it? Now if you build and deploy the solution with the Tasks and Executors lists created previously, and then create a dashboard containing default views of each of them, your dashboard will sport a handy drag-and-drop feature.

Browser Support

By now, just about all of the major browsers except Opera support HTML5 drag and drop. I've tested Internet Explorer 9.0.8112.16421, Chrome 16.0.912.75 and Firefox 3.6.12, and they're all fully compatible with the described solution. I expect that some older versions of these browsers will work too. Actually, Safari should also work, but as of this writing, it has some strange bugs in its HTML5 drag-and-drop implementation that prevent the solution from working as expected.

Figure 9 **The DragOver and DragLeave Event Handlers**

```
function UserDragOver(e, toElement) {
  // Highlight the placeholder
  toElement.style.backgroundColor = '#efe';
  // toElement.className = 'userDragOver';

  // Denote the ability to drop
  if (e.preventDefault)
      e.preventDefault();
  e.dataTransfer.dropEffect = 'copy';
  return false;
}
function UserDragLeave(e, toElement) {
  // Remove styles
  toElement.style.backgroundColor = '';
  // toElement.className = '';
}
```

What's Next?

In my solution, neither the XSLT templates nor the JavaScript contain any list-specific code—no hardcoded GUIDs, names, titles or anything like that. So, these transformations could potentially be applied to any lists or document libraries, and you could add drag-and-drop support to any column of type Person or Group. Isn't that cool?

Obviously, other types of columns can be made draggable the same way, so you could, in fact, make a solution in which all cells in all SharePoint lists are draggable, and in dashboard pages users would then be able to drag cells between lists, which is also handy.

By now, just about all of the major browsers except Opera support HTML5 drag and drop.

A very interesting challenge might be implementing by-row drag and drop. The idea is essentially the same, but to get the proper template, you should use row conditional formatting. This could be used to link lists elements or, for example, to change their order. You could, for instance, make a "Recycle Bin" link from the quick-launch menu work as a drag-and-drop acceptor, and use it as a cool way to delete items from lists.

There's so much you can do. Just think, experiment and try—your SharePoint portal will become a much friendlier place. ■

Andrey Markeev *is a SharePoint Server MVP who works as a SharePoint developer at Softline Group in Russia. Markeev is one of the top 10 experts on SharePoint StackExchange (bit.ly/w9e4NP), the creator of several open source projects on CodePlex, and an active blogger and speaker. You can follow him on Twitter at twitter.com/amarkeev.*

Thanks to the following technical experts for reviewing this article: *Michael Nemtsev and Brandon Satrom*

TED NEWARD

Talk to Me, Part 2: ELIZA

When we last met, we built a simple system for responding to voice inputs over the phone using Tropo, a cloud-hosted voice-and-SMS service. We got as far as being able to respond to voice inputs and offer responses, hopefully bailing you out of hot water with your significant other for spending too much time on the Xbox over the holidays. For some of you, unfortunately, that might not have carried the day, and things are still tense between you. This is when, as a compassionate human being, I'd like to be able to offer my services as an amateur therapist. Regrettably, however, I don't scale very well and can't talk to each and every one of you. So, instead, let's examine an alternative. I speak, of course, of ELIZA.

> If you've never dabbled in even the simplest NLP, however, trying something like this can be a bit daunting.

ELIZA: A History

For those unfamiliar with ELIZA, "she" is a *chatterbot*—and one of the first and most recognizable steps toward artificial intelligence (AI). Written back in the '60s by Joseph Weizenbaum, in Lisp, ELIZA is a relatively simple (by today's standards) input-response processor that analyzes user input for "keys" and then generates human-like responses based on those keys. So, for example, if you said, "I am sad," to ELIZA, she might respond, "Why are you sad?" or "Does talking to me make you sad?" or even "Stop being sad!" In fact, the responses could be so authentic at times that, for a while, it was thought this might be a way to get a program to pass the Turing test.

Four decades later, however, we're still not having conversations with our computers the way Arthur C. Clarke imagined in "2001: A Space Odyssey," but that doesn't mean we should ignore "human-like" communication in our programs. We're starting to see natural language processing (NLP) subtly slip more and more into computerized systems, and when combined with speech-to-text recognizer engines, entirely new avenues of human-computer interaction open up. For example, even a simple ELIZA-like key-recognizer system can be helpful in trying to create human-assistance systems on Web sites or in routing customers to the right department within a large corporation without requiring

Figure 1 **The Feliza Knowledge Base**

```
let knowledgeBase =
  [
    ( "Bye",
      [ "So long! Thanks for chatting!";
        "Please come back soon, I enjoyed talking with you";
        "Eh, I didn't like you anyway" ] );
    ( "What is your name",
      [ "My name is Feliza";
        "You can call me Feliza";
        "Who's asking?" ] );
    ( "Hi",
      [ "Hi there";
        "Hello!";
        "Hi yourself" ] );
    ( "How are you",
      [ "I'm fine, how are you?";
        "Just peachy";
        "I've been better" ] );
    ( "Who are you",
      [ "I'm an artificial intelligence";
        "I'm a collection of silicon chips";
        "That is a very good question" ] );
    ( "Are you intelligent",
      [ "But of course!";
        "What a stupid question!";
        "That depends on who's asking." ] );
    ( "Are you real",
      [ "Does that question really matter all that much?";
        "Do I seem real to you?";
        "Are you?" ] );
    ( "Open the pod bay doors",
      [ "Um... No.";
        "My name isn't HAL, you dork.";
        "I don't know... That didn't work so well last time." ] );
  ]
let unknownResponses =
  [ "I'm sorry, could you repeat that again?";
    "Wait, what?";
    "Huh?" ]

let randomResponse list =
  let listLength list = (List.toArray list).Length
  List.nth list (rand.Next(listLength list))

let cleanInput (incoming : string) =
  incoming.
    Replace(".", "").
    Replace(",", "").
    Replace("?", "").
    Replace("!", "").
    ToLower()

let lookup =
  (List.tryFind
    (fun (it : string * string list) ->
      (fst it).Equals(cleanInput input))
    knowledgeBase)
randomResponse (if Option.isSome lookup then
                    (snd (Option.get lookup))
                else
                    unknownResponses)
```

Figure 2 **A Catchall Rule for Responding**

```
let processingRules =
  [
    // ...
    // Catchall rule for when nothing else matches before
    // this point; consider this the wildcard case. This
    // must always be the last case considered!
    (
      (fun (it : string) ->
        Some(randomResponse
          [
          "That didn't make sense.";
          "You cut out for a second there. What did you say?";
          "Wait--the Seahawks are about to... Never mind. They lost.";
          "I'm sorry, could you repeat that again?";
          "Wait, what?";
          "Huh?"
          ]))
    )
  ]
List.head (List.choose (fun (it) -> it (cleanInput input)) processingRules)
```

the highly frustrating, "Press 1 for Customer Service, press 2 for Human Resources, press 3 for …" call tree.

If you've never dabbled in even the simplest NLP, however, trying something like this can be a bit daunting. Fortunately, we can get some good results from even some very basic attempts. There's a great tutorial at bit.ly/uzBSM9, which serves as the inspiration for what we're about to do next, which is to write an ELIZA implementation in F#. The choice of F# here is twofold: first, as homage to the use of Lisp in the original ELIZA, because both are functional languages, and second, because I haven't done a column sample in F# in a while. Naturally, we'll call her F#-Eliza, or Feliza for short (because that sounds more exotic), and implement her as an F# library so she can be embedded in a variety of different programs.

Feliza: Version 0

The interface to Feliza should be short, sweet, straightforward—and hide a whole bunch of complexity. From the classic Gang-of-Four patterns catalog, this is the Façade pattern (""), and F# makes it easy to create a Façade through the use of its "module" functionality:

```
module Feliza

open System

let respond input =
  "Hi, I'm Feliza"
```

Using Feliza in a console-mode program, for example, would then be as easy as this:

```
open Feliza
open System

let main =
  Console.WriteLine("Hello!")
  while true do
    Console.Write("> ")
      let input = Console.ReadLine()
      let responseText = respond input
      Console.WriteLine(responseText)

      if (input.ToLower().Equals("bye")) then
        Environment.Exit(0)
  ()
```

This, then, forms our test bed. It also makes it easier to embed Feliza in other environments if we can stick to this über-simple API.

As we get going on building some working implementations, by the way, remember that Feliza isn't intended to be a general-purpose NLP engine—that takes a lot more work than I have room for in this column. In fact, Microsoft Research has an entire division dedicated to NLP (see research.microsoft.com/groups/nlp for more information on what they're investigating). And take careful note that I'm not one of them.

Feliza: Version 1

The easiest way to get to a "working" version 1 of Feliza is to create a simple list of possible responses and choose randomly among them:

```
let respond input =
  let rand = new Random()

  let responseBase =
    [| "I heard you!";
      "Hmm. I'm not sure I know what you mean.";
      "Continue, I'm listening...";
      "Very interesting.";
      "Tell me more..."
    |]
  responseBase.[rand.Next(responseBase.Length - 1)]
```

Working with an array here is not idiomatically "F#-ish," but it does make it easier to select randomly from the possible responses. It's not a particularly exciting conversation, though it may seem familiar to anyone who has ever tried to talk to a programmer who's trying to write code at the time. Still, for a little while, it might actually feel like a real conversation. We can do better, though.

> The interface to Feliza should be short, sweet, straightforward—and hide a whole bunch of complexity.

Feliza: Version 2

A next obvious implementation is to create a "knowledge base" of canned responses to particular inputs from the user. This is easily modeled in F# using tuples, with the first element in the tuple being the input phrase to which we wish to respond, and the second element being the response. Or, to be more human about it (and avoid obvious repetitions in responses), we can make the second element a list of possible responses, and randomly choose one from that list, as shown in **Figure 1**.

Figure 3 **The Function Construct**

```
(function
  | "How are you?" ->
    Some(randomResponse
      [
      "I'm fine, how are you?";
      "Just peachy";
      "I've been better"
      ])
  | "open the pod bay doors" ->
    Some(randomResponse
      [
      "Um ... No.";
      "My name isn't HAL, you dork.";
      "I don't know ... That didn't work so well last time."
          ])
  | _ -> None
);
```

Figure 4 **Tying Reponses to Keywords**

```
(fun (it : string) ->
  if it.Contains("hate") || it.Contains("despise") then
    Some(randomResponse
      [ "Why do you feel so strongly about this?";
        "Filled with hate you are, young one.";
        "Has this always bothered you so much?" ])
  else
    None
);
(fun (it : string) ->
  if it.StartsWith("what is your") then
    let subject =
      it.Substring(it.IndexOf("what is your") +
                   "what is your".Length).Trim()
    match subject with
      | "name" ->
        Some(randomResponse
          [ "Feliza."; "Feliza. What's yours?";
            "Names are labels. Why are they so important to you?" ])
      | "age" ->
        Some(randomResponse
          [ "Way too young for you, old man."; "Pervert!";
            "I was born on December 6th, 2011" ])
      | "quest" ->
        Some("To find the Holy Grail!")
      | "favorite color" ->
        Some("It's sort of green but more dimensions")
      | _ ->
        Some("Enough about me. What's yours?")
  else
    None
);
```

This version does some simple cleanup of the input and seeks a match on the first part of the list of tuples (the "knowledge base"), then selects a response randomly from the list. If no "key phrase" is found in the knowledge base, an "unknown" response is generated, again selected randomly from a list. (Granted, the cleanup is done in a particularly inefficient manner, but when we're talking about human communication, delays aren't a problem. In fact, some chatterbot implementations deliberately slow down the responses and print them character-by-character, to mimic someone typing on a keyboard.)

> F# veterans will note that F# "active patterns" would be a perfect fit for some of this.

Obviously, if this were to be used in any kind of real-world scenario, because the input phrase has to be an exact match to trigger the response, we would need a much, much larger knowledge base, incorporating every possible permutation of human speech. Ugh—not a scalable solution. What's more, we really lose a lot when Feliza doesn't respond to user input in a more meaningful way. One of the original strengths of ELIZA was that if you said, "I like potatoes," she could respond with, "Are potatoes important to you?"—making the conversation much more "personalized."

Feliza: Version 3

This version gets to be a bit more complicated, but it also offers more flexibility and power. Essentially, we turn the exact-match algorithm into a flexible one by converting the list of tuples into a list of functions that are each evaluated and given a chance to create a response. This opens up a huge list of options for how Feliza can interact with the input, and how she can pick out words from the input to generate particular responses.

It begins with a simple list of "processing rules," at the bottom of which will be a catchall response indicating she didn't know how to respond, the moral equivalent of the "unknownResponses" list from the previous version, as shown in **Figure 2**.

The core of this version is in the last line—the List.choose function takes each processingRule and executes it against the input, and if the processingRule returns a Some value, that value gets added to a list returned to the caller. So now we can add new rules, have each one return a value, and then either take the first one (as shown in **Figure 2**, by using List.head) or even randomly select one. In a future version, we might yield a Some value that's both a text response and a "weight" of appropriateness, to help selecting which is the right response.

Writing new rules becomes easier now. We can have a rule that just keys off of input, using F# pattern matching to make it easier to match:

```
(fun (it : string) ->
  match it with
  | "Hi" | "Howdy" | "Greetings" ->
    Some(randomResponse
      [
      "Hello there yourself!";
      "Greetings and salutations!";
      "Who goes there?"
      ])
  | _ -> None
);
```

Or we can use the shorthand "function" construct to do the same, as shown in **Figure 3**.

Most of the time, though, Feliza won't be getting those canned phrases, so we'd rather she take her cue from keywords in the user's input, as **Figure 4** specifies.

F# veterans will note that F# "active patterns" would be a perfect fit for some of this; those who aren't as familiar with the F# active patterns construct (or with the F# pattern-matching syntax in general) can find out more from Jessica Kerr's excellent two-part series on the subject at bit.ly/ys4jto and bit.ly/ABQkSN.

Next: Connecting to Feliza

Feliza is great, but without an input channel that reaches beyond the keyboard, she doesn't go very far. Feliza wants to help a lot more people than just those who are sitting in front of the keyboard—she wants to be accessible to anyone with a cell phone or Internet connection, and in the next installment, we'll "hook her up" to do exactly that.

Happy coding! ■

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, 2010). 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: *Matthew Podwysocki*

Does your Team do more than just track bugs?

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

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

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

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

Native Smart Card Login Support including Government and DOD

New in Team 2.11

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

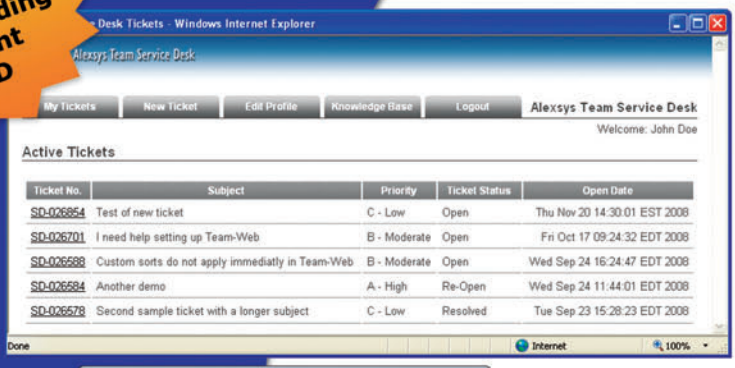
Service Desk Features

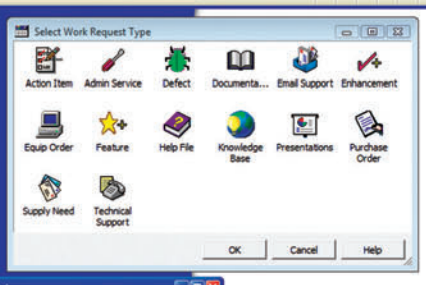
- Fully Secure
- Unlimited Users Self Registered or Active Directory
- Integrated into Your Web Site
- Fast/AJAX Dynamic Content
- Unlimited Service Desks
- Visual Service Desk Builder

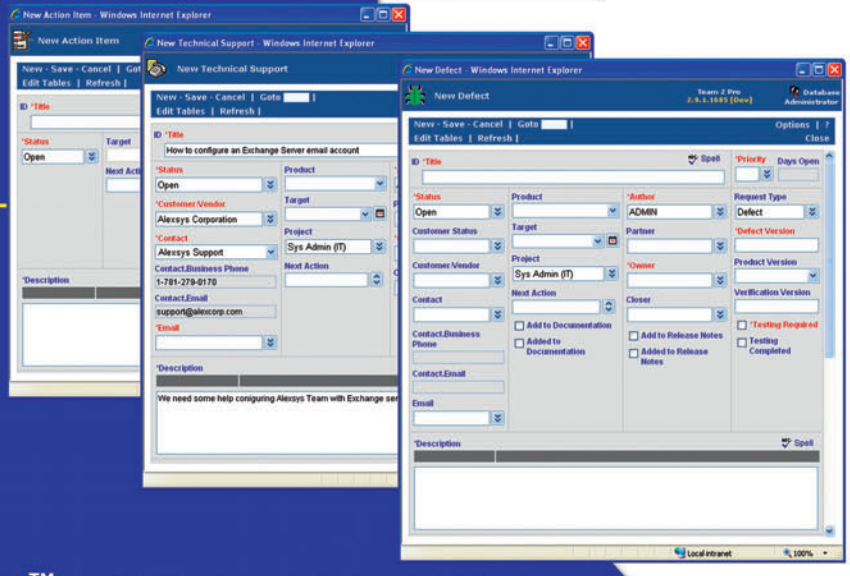
Team 2 Features

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

ZDNet Editors' Pick







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

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

JOHN PAPA

Knockout's Built-in Bindings for HTML and JavaScript

Knockout brings a rich data-binding implementation to HTML5 and JavaScript development. Once you grasp the concept of observables, the easiest way to slide into development with Knockout is to understand the variety of built-in bindings it offers. Knockout's built-in bindings are the simplest way to tap its data-binding features and add robust data binding to many aspects of your HTML5 apps. The previous column introduced Knockout, covered its various types of observables and explored the control of flow built-in bindings. This time I'll delve further into Knockout's built-in bindings. The code samples, which you can download from code.msdn.microsoft.com/mag201203ClientInsight, demonstrate how to use the various built-in bindings and explain the scenarios in which you might want to use them.

You can download the latest version of Knockout (currently 2.0.0) from bit.ly/scmtAi and reference it in your project, or you can use the NuGet Package Manager Visual Studio extension (available at bit.ly/dUeqlu) to download Knockout.

> Knockout brings a rich data-binding implementation to HTML5 and JavaScript development.

What Are the Built-in Bindings in Knockout?

At its most basic level, data binding requires a binding source (such as a JavaScript object) and a target to bind to (an HTML element, for example). The binding source is frequently called a view model. The target element may have several properties, so it's important to know what target property to bind to, as well. For example, if you want to bind your view model firstName property to an input tag's text, you'd want to bind to the Knockout value binding. In this case, Knockout identifies the target property through one of its built-in bindings: value. The Knockout built-in bindings allow you to bind to properties as well as methods of your view model. Knockout includes many built-in bindings that bind view model properties to target elements, as I'll discuss in this article.

Code download available at code.msdn.microsoft.com/mag201203ClientInsight.

Figure 1 **View Model with Properties, Nested Children and Methods**

```
my.showroomViewModel = {
  id: ko.observable("123"),
  salePrice: ko.observable(1995),
  profit: ko.observable(-7250),
  rating: ko.observable(4),
  isInStock: ko.observable(true),
  model: {
         code: ko.observable("314ce"),
         name: ko.observable("Taylor 314 ce")
  },
  colors: ko.observableArray([
         { key: "BR", name: "brown" },
         { key: "BU", name: "blue" },
         { key: "BK", name: "black"}]),
  selectedColor: ko.observable(""),
  selectedColorsForDropdown: ko.observableArray([]),
  selectedColorForRadio: ko.observableArray(),
  allowEditing: ko.observable(true),
  isReadonly: ko.observable(true),
    onSalesFloor: ko.observable(true),
    qty: ko.observable(7),
  photoUrl: ko.observable("/images/314ce.png"),
  url: ko.observable("http://johnpapa.net"),
  details: ko.observable("<strong><em>This guitar rocks!</em></strong>"),
  checkboxHasFocus: ko.observable(false),
  textboxHasFocus: ko.observable(false),
  buttonHasFocus: ko.observable(false),
  userInput: ko.observable(""),
  setFocusToCheckbox: function () {
         this.checkboxHasFocus(true);
  },
  displayValue: function () {
         if (this.userInput().length > 0) {
                window.alert("You entered: " + this.userInput());
         }
  },
  detailsAreVisible: ko.observable(false),
  showDetails: function () {
         this.detailsAreVisible(true);
  },
  hideDetails: function () {
         this.detailsAreVisible(false);
  },
  useUniqueName: ko.observable(true)
};
ko.applyBindings(my.showroomViewModel);
```

The syntax for using the built-in bindings is to include the Knockout binding name and the view model property pairs inside of the data-bind property of an HTML element. If you want to data bind to more than one property in the HTML element, simply separate the bindings by a comma using this syntax:

```
data-bind="built-in-binding:viewmodel-property1, another-built-in-binding:viewmodel-property2"
```

Following this pattern, you could bind the input element's value to the view model salePrice property, like so:

```
<input type="text" data-bind="value:salePrice " />
```

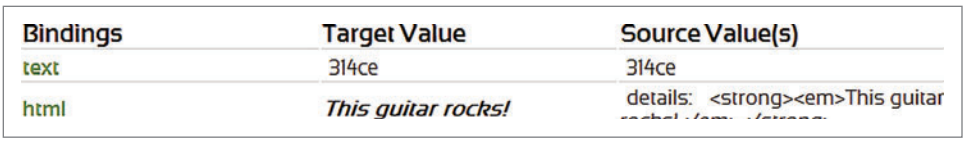


Figure 2 **The Knockout Text and HTML Bindings**

Knockout's built-in bindings allow you to handle most binding scenarios, but if you encounter a specialized binding scenario that isn't covered, you can create custom bindings with Knockout, too. I'll cover custom bindings in a future article.

Fundamental Bindings: text and html

Let's dive in by exploring the built-in bindings Knockout provides. **Figure 1** shows the view model that all of the examples in this article will use for their built-in bindings. The sample data is for a guitar, but it's just to demonstrate the bindings.

Perhaps the most common binding is the text binding. When Knockout sees a text binding, it sets the innerText property for Internet Explorer, or the equivalent property in other browsers. When the text binding is used, any previous text will be overwritten. The text binding is often used to display values in a span or div. This example binds the view model's model.code property to a span:

```
<span data-bind="text: model.code"></span>
```

The html binding isn't used as often, but it's very handy for rendering HTML content in your view model. In the following example, the contents of the html property are rendered, making the text bold italics:

```
<tr>
  <td><div class="caption">html</div></td>
  <td><div data-bind="html: details"></div></td>
  <td><span>details: </span><span data-bind="text: details"></span></td>
</tr>
```

You can see the results of these examples in **Figure 2** (and all examples by running the 04-builtin-bindings.html page in the sample code). All examples show the built-in binding, the target and the source values from the view model.

Value Binding

Data binding is arguably most useful for apps that are very interactive, so it makes sense that most of the built-in bindings in Knockout help bind to input and form elements, such as textboxes, checkboxes and dropdown lists. Let's explore these built-in bindings by first demonstrating the versatility of Knockout's value binding.

The value binding works with many of the HTML input types to bind a view model property directly to the value of an HTML input element, such as a textbox checkbox or radio button. The following example shows the view model model.code property being bound to a textbox. The property is defined using Knockout's observable function, which makes it notify the target when the source value changes:

```
<td><input type="text" data-bind="value: model.
code"/></td>
<td><span>model.code: </span><span data-bind="text:
model.code"></span></td>
```

If a user changes the value in the textbox, the new value is sent from the target (the textbox) to the source (the view model model.code property) when the user tabs away from the textbox. However, you could also use a special Knockout binding to tell Knockout to update the target to the source on every keystroke. In the next example, the textbox value is bound to the view model salePrice property and the valueUpdate binding is bound to afterkeydown. valueUpdate is a parameter for the value binding that helps you define when the value binding should be updated. Here the code is telling Knockout to update the source after every key down press (you can try this sample by running the sample code; the results are shown in **Figure 3**):

```
<td><input type="text" data-bind="value: salePrice, valueUpdate:
'afterkeydown'"/></td>
<td><span>salePrice: </span><span data-bind="text: salePrice"></span></td>
```

> Perhaps the most common binding is the text binding.

Binding Checkboxes and Radio Buttons

Checkboxes can be data bound to Knockout's *checked* binding. The checked binding should be bound to a property or expression that evaluates to true or false. Because the view model properties are defined as observables, the checkbox is updated when the source property changes. Likewise, when a user checks or unchecks the checkbox, the value is updated in the view model property. The following example shows the checkbox being bound to the isInStock property (from the view model in **Figure 1**; the results are shown in **Figure 4**):

```
<td><input type="checkbox" data-bind="checked: isInStock"/></td>
<td><span>isInStock: </span><span data-bind="text: isInStock"></span></td>
```

You can also use the checked binding to select a radio button from a group of radio buttons. The following example shows a set of radio buttons whose values are all hardcoded to two-letter codes representing a color; when a user selects a color via the radio button, the checked property is set and the view model selectedColorForRadio property is updated to the two-letter value:

```
<td>
  <input type="radio" value="BR" data-bind=
    "checked:  selectedColorForRadio" /><span>brown</span>
  <input type="radio" value="BU" data-bind=
    "checked: selectedColorForRadio" /><span>blue</span>
  <input type="radio" value="BK" data-bind=
    "checked: selectedColorForRadio" /><span>black</span>
</td>
<td><span>selectedColorForRadio: </span><span data-bind=
  "text: selectedColorForRadio"></span></td>
```

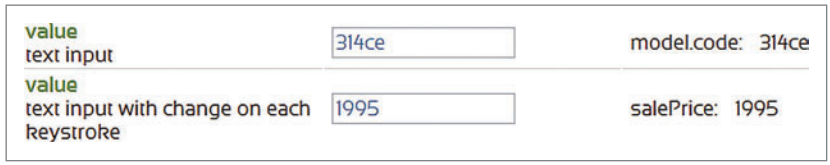


Figure 3 **The Value Binding to Textboxes**

Figure 4 **The Checked Binding**

Figure 5 **The Checked and Value Bindings Used in Radio Buttons**

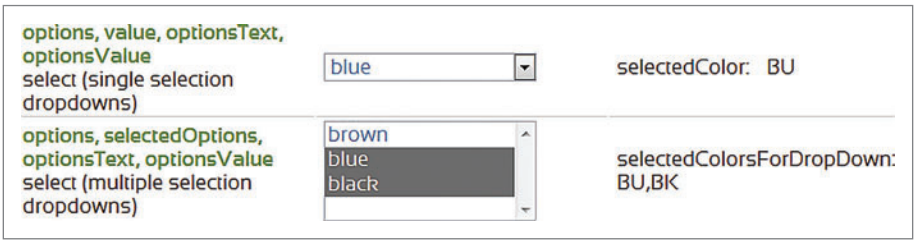


Figure 6 **Binding to Dropdown Lists**

While this works perfectly well, I find it more useful to data bind the list of colors to a series of radio buttons, which you can do by combining the three built-in bindings: value, checked and foreach. The view model in **Figure 1** has a colors property, which is an array of objects containing each color's name and a key value. The foreach binding in the next example loops through the colors array property, setting each radio button's value binding to the color's key property and a span's text binding to the color's name property:

```
<td>
  <div data-bind="foreach: colors">
    <input type="radio" data-bind=
      "value:key, checked: $parent.selectedColorForRadio" />
       <span data-bind="text: name"></span>
  </div>
</td>
<td><span>selectedColorForRadio: </span>
  <span data-bind="text: selectedColorForRadio"></span></td>
```

The radio button's checked binding is set to the view model selectedColorForRadio property using the $parent function. However, the binding can't simply be bound directly to that property because the foreach binding changed the context from the view model to the colors property. To properly bind to the view model's property, the code needs to refer back the context's parent (in this case the view model itself). The Knockout $parent function tells Knockout to refer one level up the context hierarchy, which data binds the checked binding to the view model selectedColorForRadio property. (There are many useful functions and tips like this that I'll explore in future articles.) The results of this example are shown in **Figure 5**.

Binding Dropdown Lists

Dropdown lists have several important properties to load a list of items, display a value, use a different key value and store the user's selection. Knockout provides a built-in binding for each of these.

The options binding identifies a list of values to display, usually from an array property on the view model. The example in this section sets the options binding to the view model colors property. Sometimes you want to display one value in the dropdown list but use another value when a user selects an item from the list. Knockout's built-in optionsText and optionsValue bindings help. The optionsText binding is set to the string name of the property to display in the dropdown list, from the options binding. The optionsValue binding is set to the string name of the property to bind to for the selected value of the item in the dropdown list. In this example, the colors array contains objects with a name and key property, of which the name will be used for the optionsText and the key for the optionsValue. The value binding is set to the view model selectedColor property, where the user's selection will be stored:

```
<td>
  <div class="caption">options, value, optionsText, optionsValue</div>
  <div>select (single selection dropdowns)</div>
</td>
<td><select data-bind="options: colors, value: selectedColor,
   optionsText: 'name', optionsValue: 'key'" ></select></td>
<td><span>selectedColor: </span><span data-bind="text: selectedColor"></span></td>
```

If you want to allow multiple selections from a dropdown list, you first add the multiple property for the HTML select element. Then you replace Knockout's selectedOption (singular) binding with its selectedOptions (plural) binding:

```
<td>
  <div class="caption">options, selectedOptions, optionsText, optionsValue</div>
  <div>select (multiple selection dropdowns)</div>
</td>
<td><select data-bind="options: colors,selectedOptions: selectedColorsForDropdown,
  optionsText: 'name', optionsValue: 'key'" multiple="multiple" ></select></td>
<td><span>selectedColorsForDropDown: </span><span data-bind=
  "text: selectedColorsForDropdown"></span></td>
```

Because the HTML *select* element is allowing multiple selections, the view model selectedColorsForDropdown property (which is set to the selectedOptions built-in binding) will contain a comma-delimited list of values for the selections.

Figure 6 shows the results of selecting both the blue and black colors. Notice that the dropdown lists display the name of the color (blue and black), but use the key (BU and BK) as the selected values.

Enabling and Disabling Input Elements

Knockout provides built-in bindings to enable and disable input elements. The enable binding will enable the input element if the property it's bound to evaluates to true, and will disable the element if it evaluates to false. The disable binding does the exact opposite:

```
<td>
  <input type="checkbox" data-bind="checked: allowEditing"/>
  <input type="text" data-bind="enable: allowEditing, value:salePrice" />
</td>
<td><span>allowEditing: </span><span data-bind="text: allowEditing"></span></td>
```

This sample code demonstrates that the checkbox value is data bound to the view model allowEditing property, which is also bound to the textbox enable binding. So when the checkbox is checked, the textbox is enabled; when it's unchecked, the textbox is disabled.

In contrast, the next sample demonstrates how the checkbox checked binding is bound to the view model isReadonly property and the textbox

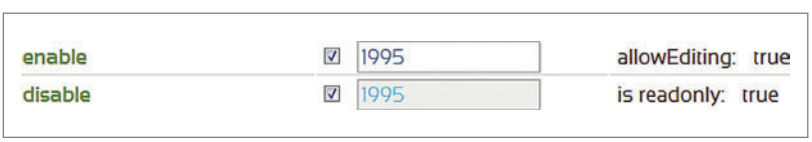


Figure 7 **Bindings for Enabling and Disabling Elements**

Figure 8 **Setting the Hasfocus Binding**

```
<td>
  <input type="checkbox" data-bind="hasfocus: checkboxHasFocus"/>
  <input type="text" data-bind="hasfocus: textboxHasFocus"/>
  <button data-bind="click: setFocusToCheckbox, hasfocus:buttonHasFocus">
    set focus to checkbox</button>
  <br/>
  <span data-bind="visible: checkboxHasFocus">checkbox has focus</span>
  <span data-bind="visible: textboxHasFocus">textbox has focus</span>
  <span data-bind="visible: buttonHasFocus">button has focus</span>
</td>
<td>
  <span>checkboxHasFocus: </span><span data-bind="text: checkboxHasFocus">
    checkbox has focus</span>
  <br/>
  <span>textboxHasFocus: </span><span data-bind="text: textboxHasFocus">
    textbox has focus</span>
  <br/>
  <span>buttonHasFocus: </span><span data-bind="text: buttonHasFocus">
    button has focus</span>
</td>
```

disable binding is set to the isReadonly property. So when the checkbox is checked, the textbox is disabled (the results of both samples can be seen in **Figure 7**):

```
<td>
  <input type="checkbox" data-bind="checked: isReadonly"/>
  <input type="text" data-bind="disable: isReadonly, value:salePrice" />
</td>
<td><span>is readonly: </span><span data-bind="text: isReadonly"></span></td>
```

Binding the Focus

Knockout has a built-in binding named hasfocus that determines and sets which element has the focus. The hasfocus binding is handy when you want the focus to be set to a specific element on a form. If multiple elements have the hasfocus binding with values that evaluate to true, the focus will be set to the element that had its hasfocus set most recently. You can set the hasfocus binding to the keyword true to move focus directly to an element. Or you can bind it to a view model property, as shown in the sample code in **Figure 8**.

This code sets the hasfocus binding appropriately for a checkbox, textbox and a button element to three different view model properties. When the focus is set to one of these HTML elements, the corresponding hasfocus binding sets the view model property to true for that element (and the others to false). You can try this example with the downloadable code, or see the results in **Figure 9**, where a user has placed the focus in the textbox.

Binding the Visibility

Knockout's visible binding should be bound to a property that evaluates to true or false. This binding will set the element's display style to visible if true (either true or a non-null value) or none if false (false, 0, undefined or null).

The next sample shows the checkbox checked binding and the textbox visible binding both set to the view model onSalesFloor property. When the checkbox is checked, the onSalesFloor property is set to true and the textbox becomes visible. When the checkbox is unchecked, the onSalesFloor property is set to false and the textbox is no longer visible (see **Figure 10**):

```
<td>
  <input type="checkbox" data-bind="checked: onSalesFloor"/>
  <input type="text" data-bind="visible: onSalesFloor, value:qty" />
</td>
<td>
  <span>onSalesFloor: </span><span data-bind="text: onSalesFloor"></span>
</td>
```

Knockout's visible binding should be bound to a property that evaluates to true or false.

Event Bindings

Knockout supports binding to any event through its *event* built-in binding, but it also has two special built-in bindings for click and submit. The *click* binding should be used on an element when you want to bind the click event to a method in a view model. It's most often used with a button, input or an *a* element, but can be used with any HTML element.

The following code sets the button click binding to the displayValue method on the view model; in **Figure 1**, you can see that the displayValue method on the view model simply displays the view model userInput property (which is bound to the textbox) with an alert:

```
<td>
  <input type="text" data-bind="value: userInput"/>
  <button data-bind="click: displayValue">display value</button>
</td>
<td>
  <span>userInput: </span><span data-bind="text: userInput"></span>
</td>
```

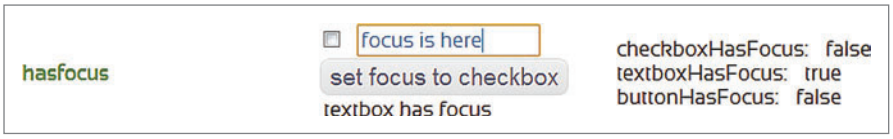


Figure 9 **Binding for Setting the Focus**

Figure 10 **Binding for Visibility**

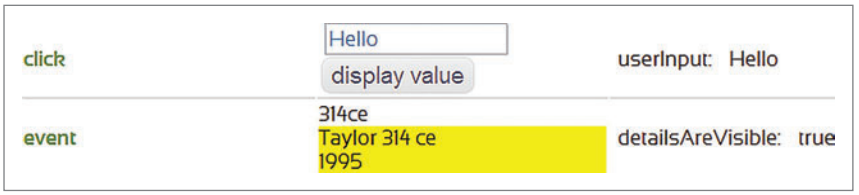


Figure 11 **The Click and Event Bindings**

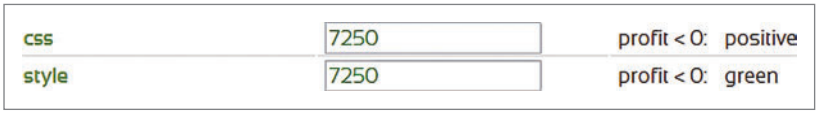


Figure 12 **Style Bindings**

When you want to bind a view model method to an event other than click, you can use Knockout's event binding. Because the *click* binding is the most-used binding for events, it's simply a shortcut to the event binding.

Knockout's event binding allows you to bind to any event. To use event binding, you pass an object literal containing name value pairs for the event name and the view model method, separated by commas. The following sample code sets Knockout's built-in event binding so that the mouseover and mouseout events are bound to the showDetails and hideDetails methods on the view model. These methods set the view model observable property detailsAreVisible to true or false, accordingly:

```
<td>
  <div data-bind="text:model.code, event: {mouseover: showDetails,
    mouseout: hideDetails}"></div>
  <div data-bind="visible: detailsAreVisible" style="background-color: yellow">
    <div data-bind="text:model.name"></div>
    <div data-bind="text:salePrice"></div>
  </div>
</td>
<td>
  <span>detailsAreVisible: </span><span data-bind="text: detailsAreVisible"></span>
</td>
```

The second div sets the visible binding to the view model detailsAreVisible property, so when the user moves the mouse over the first div, the second div becomes visible. When the mouse is moved away from the first div, the second div is no longer visible. The results are shown in **Figure 11**. The submit binding (not shown in **Figure 11**) accepts any input gesture that will submit an HTML form.

> The attr built-in binding allows you to data bind any attribute to a view model property.

Style Bindings

You can bind styles with Knockout using the css and the style built-in bindings. The css binding can be set to one or more valid css class names. The following sample shows that the textbox has its value binding set to the view model profit property and its css binding set to an object literal. The object literal contains one or more css class names to apply and a corresponding expression that should evaluate to true or false:

```
<td>
  <input data-bind="value:profit, css: {negative: profit() < 0,
    positive: !(profit() < 0), }"/>
</td>
<td>
  <span>profit < 0: </span><span data-bind="text: profit() < 0 ?
    'negative' : 'positive'"></span>
</td>
```

For example, if the profit property evaluates to less than 0, the css class named *negative* will be applied. Similarly, the second expression is evaluated and if it's true, the css class named *positive* will be applied.

While I recommend using css classes whenever possible, at times you might want to set a specific style as well. Knockout supports this with its *style* built-in binding. In the following example, the textbox color changes to red if the profit is less than 0, and to green if the profit is greater than 0 (the results for both the css and style bindings are shown in **Figure 12**):

```
<td>
  <input data-bind="value:profit, style: {color: profit() < 0 ? 'red' :
    'green'}"></input>
</td>
<td>
  <span>profit < 0: </span><span data-bind="text: profit() < 0 ? 'red' :
    'green'"></span>
</td>
```

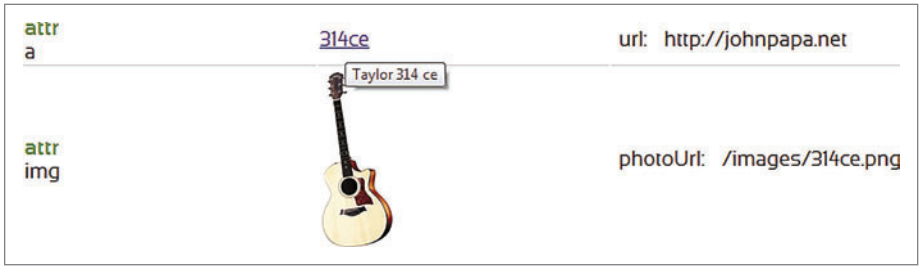


Figure 13 **Binding to Element Attributes**

Binding to Other HTML Attributes

While Knockout has many built-in bindings, you will surely encounter some situations for which none exist. For these, Knockout offers the attr built-in binding, which allows you to data bind any attribute to a view model property. This is very useful in many common scenarios, such as binding the href and title of the a element:

```
<td>
  <a data-bind="attr: {href: url, title: model.name}, text:model.code"></a>
</td>
<td><span>url: </span><span data-bind="text: url"></span></td>
```

Another common use for the attr binding is to make the img element bind its src attribute to the view model photoUrl property (you can see the results of both of these samples in **Figure 13**):

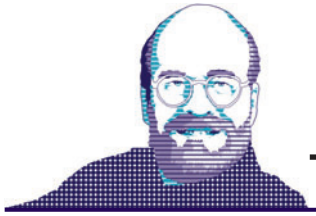
```
<td>
  <img data-bind="attr: {src: photoUrl, alt: model.code}" class="photoThumbnail"/>
</td>
<td><span>photoUrl: </span><span data-bind="text: photoUrl"></span></td>
```

Wrapping Up

This article explored many of the built-in bindings that Knockout offers. There are a few others, most notably the template binding, which I'll cover in a future article. In any case, the concepts are the same. Determine the binding property you want to use on the target element and the view model member to which you want to bind it. Once you grasp Knockout's observables and its variety of built-in bindings, you have the fundamental building blocks to create robust Web apps using the Model View ViewModel, or MVVM, pattern. ■

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 conferences such as BUILD, MIX, Professional Developers Conference, Tech·Ed, Visual Studio Live! and DevConnections. 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*



CHARLES PETZOLD

Streaming Audio in Windows Phone

Whether running on the desktop, the Web or in your hand, computer programs sometimes need to play sounds or music. Most often, this audio will be entirely encoded in MP3 or WMA files. The big advantage to this approach is that the OS itself usually knows how to decode and play these files. The application can then focus on the relatively easier job of providing a UI for pausing, restarting and perhaps navigating among tracks.

But life isn't always so convenient. Sometimes a program needs to play an audio file in a format not supported by the OS, or even to generate audio data dynamically, perhaps to implement electronic music synthesis.

In the parlance of Silverlight and Windows Phone, this process is known as audio "streaming." At run time the application provides a stream of bytes that comprise the audio data. This occurs through a class derived from MediaStreamSource, which feeds the audio data to the OS audio player on demand. Windows Phone OS 7.5 can stream audio in the background, and I'll show you how to do it.

The essential first step in dynamically generating audio data in a Windows Phone program is deriving from the abstract class MediaStreamSource.

Deriving from MediaStreamSource

The essential first step in dynamically generating audio data in a Windows Phone program is deriving from the abstract class MediaStreamSource. The code involved is rather messy in spots so instead of writing the code from scratch, you'll probably want to copy somebody else's.

The SimpleAudioStreaming project in the downloadable source code for this article shows one possible approach. This project contains a MediaStreamSource derivative named Sine440AudioStreamSource that simply generates a sine wave at 440 Hz. This is the frequency corresponding to the A above middle C that's commonly used as a tuning standard.

Code download available at code.msdn.microsoft.com/mag201203TouchAndGo.

MediaStreamSource has six abstract methods that a derivative needs to override, but only two of them are crucial. The first is OpenMediaPlayer, in which you need to create a couple Dictionary objects and a List object, as well as define the type of audio data your class provides and various parameters describing this data. The audio parameters are represented as fields of a Win32 WAVEFORMATEX structure with all the multibyte numbers in little-endian format (most significant byte first) and converted to a string.

I've never used MediaStreamSource for anything other than audio in the pulse-code modulation (PCM) format, which is used for most uncompressed audio, including CDs and the Windows WAV file format. PCM audio involves constant-size samples at a constant rate called the sample rate. For CD-quality sound, you'll use 16 bits per sample and a sample rate of 44,100 Hz. You can choose either one channel for monaural sound or two channels for stereo.

The Sine440AudioStreamSource class hardcodes a single channel and a 16-bit sample size but allows the sample rate to be specified as a constructor argument.

Internally, the audio pipeline maintains a buffer of audio data, the size of which is specified by the AudioBufferLength property

Figure 1 **The GetSampleAsync Method in Sine440AudioStreamSource**

```
protected override void GetSampleAsync(MediaStreamType mediaStreamType)
{
  // Reset MemoryStream object
  memoryStream.Seek(0, SeekOrigin.Begin);

  for (int sample = 0; sample < BufferSamples; sample++)
  {
    short amplitude = (short)(short.MaxValue * Math.Sin(angle));
    memoryStream.WriteByte((byte)(amplitude & 0xFF));
    memoryStream.WriteByte((byte)(amplitude >> 8));
    angle = (angle + angleIncrement) % (2 * Math.PI);
  }

  // Send out the sample
  ReportGetSampleCompleted(new MediaStreamSample(mediaStreamDescription,
    memoryStream,
    0,
    BufferSize,
    timestamp,
    mediaSampleAttributes));
  // Prepare for next sample
  timestamp += BufferSamples * 10000000L / sampleRate;
}
```

of MediaStreamSource. The default setting is 1,000 ms, but you can set it as low as 15 ms. To keep this buffer full, calls are made to the GetSampleAsync method of your MediaStreamSource derivative. Your job is to shovel a bunch of audio data into a MemoryStream and call ReportGetSampleCompleted.

The Sine440AudioStreamSource class is hardcoded to provide 4,096 samples per call. With a sample rate of 44,100, that's not quite one-tenth second of audio per call. Playing around with this value and the AudioBufferLength property is necessary if you're implementing a user-controlled synthesizer that must respond quickly to user input. For minimum latency you'll want to keep the buffer sizes small but not too small such that gaps in the playback result.

Figure 1 shows the Sine440AudioStreamSource implementation of the GetSampleAsync override. Within the loop, a 16-bit sine value is obtained from a call to the Math.Sin method scaled to the size of the following short:

```
short amplitude = (short)(short.MaxValue * Math.Sin(angle));
```

That amplitude is then split into 2 bytes and stored in the MemoryStream, low byte first. For stereo, each sample requires two 16-bit values, alternating between left and right.

Other calculations are possible. You can switch from a sine wave to a sawtooth wave by defining amplitude like this:

```
short amplitude = (short)(short.MaxValue * angle / Math.PI + short.MinValue);
```

In both cases, a variable named "angle" ranges from 0 to 2π radians (or 360 degrees) so it references a single cycle of a particular waveform. After each sample, angle is increased by angleIncrement, a variable calculated earlier in the class based on the frequency of the sample rate and the frequency of the waveform to be generated, which here is hardcoded as 440 Hz:

```
angleIncrement = 2 * Math.PI * 440 / sampleRate;
```

Notice that as the frequency of the generated waveform approaches half the sample rate, angleIncrement approaches π or 180 degrees. At half the sample rate, the generated waveform is based on just two samples per cycle, and the result is a square wave rather than a smooth sine curve. However, all the harmonics in this square wave are above half the sample rate.

Also notice that you can't generate frequencies greater than half the sample rate. If you try, you'll actually generate "aliases" that are below half the sample rate. Half the sample rate is known as the Nyquist frequency, named after Harry Nyquist, an engineer who worked for AT&T when he published an early paper on information theory in 1928 that laid the foundations for audio sampling technology.

For CD audio a sample rate of 44,100 Hz was chosen partially because half of 44,100 is greater than the upper limit of human hearing, commonly regarded as 20,000 Hz.

Aside from the Sine440AudioStreamSource class, the remainder of the SimpleAudioStreaming project is fairly easy: The MainPage.xaml file contains a MediaElement named mediaElement, and the MainPage OnNavigatedTo override calls SetSource on this object with an instance of the MediaStreamSource derivative:

```
mediaElement.SetSource(new Sine440AudioStreamSource(44100));
```

I originally had this call in the program's constructor, but I discovered that the music playback couldn't be resumed if you navigated away from the program and then back without the program being tombstoned.

The SetSource method of MediaElement is the same method you call if you want MediaElement to play a music file referenced with a Stream object. Calling SetSource would normally start the sound playing, but this particular MediaElement has its AutoPlay property set to false, so a call to Play is also necessary. The program includes Play and Pause buttons in its application bar that perform these operations.

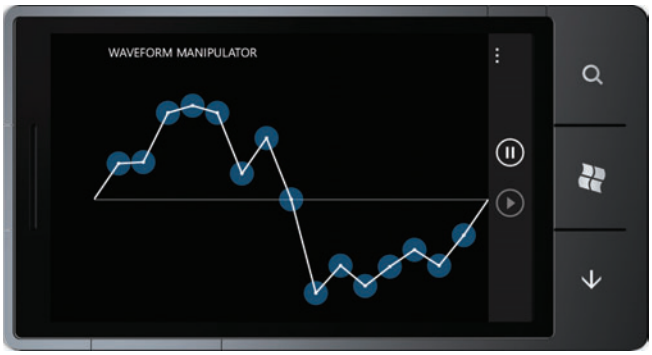


Figure 2 **The WaveformManipulator Program**

While the program is running, you can control the volume from the phone's Universal Volume Control (UVC), but if you terminate the program or navigate away from it, the sound stops.

Moving to the Background

It's also possible to use this same MediaStreamSource derivative to play sounds or music in the background. Background music continues to play on the phone if you navigate away from the program or even terminate it. For background audio, you can use the phone's UVC not only to control the volume but also to pause and restart the audio and (if applicable) jump ahead or back to other tracks.

While the program is running, you can control the volume from the phone's Universal Volume Control (UVC), but if you terminate the program or navigate away from it, the sound stops.

You'll be happy to discover that much of what you learned in last month's installment of this column (msdn.microsoft.com/magazine/hh781030) continues to apply to background audio streaming. In that column I described how to play music files in the background: You create a library project that contains a class that derives from AudioPlayerAgent. Visual Studio will generate this class for you if you add a new project of type Windows Phone Audio Playback Agent.

The application must have a reference to the DLL containing the AudioPlayerAgent but doesn't access this class directly. Instead, the application accesses the BackgroundAudioPlayer class to set an initial AudioTrack object and to call Play and Pause. You'll recall that the AudioTrack class has a constructor that lets you specify a

track title, an artist and an album name, as well as specify which buttons should be enabled on the UVC.

Typically the first argument to the AudioTrack constructor is a Uri object that indicates the source of the music file you wish to play. If you want this AudioTrack instance to play streaming audio rather than a music file, you'll set this first constructor argument to null. In that case BackgroundAudioPlayer will look for a class that derives from AudioStreamingAgent in a DLL referenced by the program. You can create such a DLL in Visual Studio by adding a project of type Windows Phone Audio Streaming Agent.

The SimpleBackgroundAudioStreaming solution in the downloadable code for this article shows how this is done. The solution contains the application project and two library projects, one named AudioPlaybackAgent containing an AudioPlayer class that derives from AudioPlayerAgent, and the other named AudioStreamAgent containing an AudioTrackStreamer class that derives from AudioStreamingAgent. The application project contains references to both these library projects, but it does not attempt to access the actual classes. For reasons I discussed in my previous column, doing so is futile.

Let me emphasize again that the application must contain references to the background agent DLLs. It's easy to omit these references because you won't see any errors, but the music won't play.

Much of the logic in SimpleBackgroundAudioStreaming is the same as for a program that plays music files in the background, except that whenever BackgroundAudioPlayer tries to play a Track with a null Uri object, the OnBeginStreaming method in the AudioStreamingAgent derivative will be called. Here's the exceptionally simple way in which I handle that call:

```
protected override void OnBeginStreaming(AudioTrack track, AudioStreamer streamer)
{
  streamer.SetSource(new Sine440AudioStreamSource(44100));
}
```

That's it! It's the same Sine440AudioStreamSource class I described earlier, but now included in the AudioStreamAgent project.

Although the SimpleBackgroundAudioStreaming program creates only one track, you can have multiple track objects, and you can mix AudioTrack objects that reference music files and those that use streaming. Notice that the AudioTrack object is an argument to the OnBeginStreaming override, so you can use that information to customize the particular MediaStreamSource you want to use for that track. To let you provide more information, AudioTrack has a Tag property that you can set to any string you want.

Building a Synthesizer

A steady sine curve at 440 Hz can be pretty boring, so let's build an electronic music synthesizer. I've put together a very rudimentary synthesizer consisting of 12 classes and two interfaces in the Petzold.MusicSynthesis library project in the SynthesizerDemos solution. (Some of these classes are similar to code I wrote for Silverlight 3 that appeared in my blog in July 2007—accessible from charlespetzold.com.)

At the center of this synthesizer is a MediaStreamSource derivative named DynamicPcmStreamSource that has a property named SampleProvider, defined like so:

```
public IStereoSampleProvider SampleProvider { get; set; }
```

The IStereoSampleProvider interface is simple:

```
public interface IStereoSampleProvider
{
  AudioSample GetNextSample();
}
```

AudioSample has two public fields of type short named Left and Right. In its GetSampleAsync method, DynamicPcmStreamSource calls the GetNextSample method to obtain a pair of 16-bit samples:

```
AudioSample audioSample = SampleProvider.GetNextSample();
```

One class that implements IStereoSampleProvider is named Mixer. Mixer has an Inputs property that's a collection of objects of type MixerInput. Each MixerInput has an Input property of type IMonoSampleProvider, defined like so:

```
public interface IMonoSampleProvider
{
  short GetNextSample();
}
```

One class that implements IMonoSampleProvider is named SteadyNoteDurationPlayer, which is an abstract class that can play a series of notes of the same duration at a particular tempo. It has a property named Oscillator that also implements IMonoSampleProvider to generate the actual waveforms. Two classes derive from SteadyNoteDurationPlayer: Sequencer, which plays a series of notes repetitively; and Rambler, which plays a random stream of notes. I use these two classes in two different applications in the SynthesizerDemos solution, one that only runs in the foreground and another that plays music in the background.

The foreground-only application is called WaveformManipulator,

Figure 3 Synthesizer Initialization Code in WaveformManipulator

```
// Initialize Waveformer control
for (int i = 0; i < waveformer.Points.Count; i++)
{
  double angle = (i + 1) * 2 * Math.PI / (waveformer.Points.Count + 1);
  waveformer.Points[i] = new Point(angle, Math.Sin(angle));
}

// Create two Sequencers with slightly different tempi
Sequencer sequencer1 = new Sequencer(SAMPLE_RATE)
{
  Oscillator = new VariableWaveformOscillator(SAMPLE_RATE)
  {
    Points = waveformer.Points
  },
  Tempo = 480
};

Sequencer sequencer2 = new Sequencer(SAMPLE_RATE)
{
  Oscillator = new VariableWaveformOscillator(SAMPLE_RATE)
  {
    Points = waveformer.Points
  },
  Tempo = 470
};

// Set the same Pitch objects in the Sequencer objects
Pitch[] pitches =
{
  ...
};

foreach (Pitch pitch in pitches)
{
  sequencer1.Pitches.Add(pitch);
  sequencer2.Pitches.Add(pitch);
}

// Create Mixer and MixerInput objects
mixer = new Mixer();
mixer.Inputs.Add(new MixerInput(sequencer1) { Space = -0.5 });
mixer.Inputs.Add(new MixerInput(sequencer2) { Space = 0.5 });
```

Figure 4 **The Synthesizer Setup for PentatonicRambler**

```
protected override void OnBeginStreaming(AudioTrack track, AudioStreamer streamer)
{
  // Create a Rambler
  Rambler rambler = new Rambler(SAMPLE_RATE,
  new Pitch(Note.Csharp, 4), // Start
  new Pitch(Note.Csharp, 2), // Minimum
  new Pitch(Note.Csharp, 6)) // Maximum
  {
    Oscillator = new AlmostSquareWave(SAMPLE_RATE),
    Tempo = 480
  };

  // Set allowable note values
  rambler.Notes.Add(Note.Csharp);
  rambler.Notes.Add(Note.Dsharp);
  rambler.Notes.Add(Note.Fsharp);
  rambler.Notes.Add(Note.Gsharp);
  rambler.Notes.Add(Note.Asharp);

  // Create Mixer and MixerInput objects
  Mixer mixer = new Mixer();
  mixer.Inputs.Add(new MixerInput(rambler));

  DynamicPcmStreamSource audioStreamSource =
    new DynamicPcmStreamSource(SAMPLE_RATE);
  audioStreamSource.SampleProvider = mixer;
  streamer.SetSource(audioStreamSource);
}
```

and it features a control that lets you interactively define a waveform used for playing back the music, as shown in **Figure 2**.

The points defining the waveform are transferred to an Oscillator derivative named VariableWaveformOscillator. As you move the round touch-points up and down, you'll notice about a one-second delay before you actually hear a change in the music's timbre. This is a result of the default 1,000 ms buffer size defined by MediaStreamSource.

The WaveformManipulator program uses two Sequencer objects loaded with the same series of notes, which are E minor, A minor, D minor and G major arpeggios. The tempi for the two Sequencer objects are slightly different, however, so they drift out of synchronization, first with a type of reverb or echo effect, and then more like counterpoint. (This is a form of "process music" inspired by the early work of American composer Steve Reich.) The initialization code from MainPage.xaml.cs that "wires up" the synthesizer components is shown in **Figure 3**.

The Mixer, DynamicPcmStreamSource and MediaElement objects are connected together in the OnNavigatedTo override:

```
DynamicPcmStreamSource dynamicPcmStreamSource =
  new DynamicPcmStreamSource(SAMPLE_RATE);
dynamicPcmStreamSource.SampleProvider = mixer;
mediaElement.SetSource(dynamicPcmStreamSource);
```

Because WaveformManipulator uses MediaElement for playing the music, it only plays when the program is running in the foreground.

Background Limitations

I gave a lot of thought to making a version of WaveformManipulator that played the music in the background using BackgroundAudioPlayer. Obviously you would only be able to manipulate the waveform when the program was in the foreground, but I couldn't get past an obstacle that I discussed in last month's column: The background agent DLLs your program supplies to handle background processing run in a different task than the program itself, and the only way I can see that these two tasks can exchange arbitrary data is through isolated storage.

I decided not to pursue this job, partially because I had a better idea for a program that played streaming audio in the background. This was a program that would play a random tune, but with notes that would change slightly when the Accelerometer registered a change in phone orientation. Shaking the phone would create a new tune entirely.

This project got as far as my attempt to add a reference to the Microsoft.Devices.Sensors assembly to the AudioStreamAgent project. That move invoked a message box with a red X and the message, "An attempt has been made to add a reference unsupported by a background agent." Apparently background agents can't use the accelerometer. So much for that program idea!

Instead, I wrote a program called PentatonicRambler, which uses background streaming to play a never-ending melody in a pentatonic scale consisting of only the five black notes of the piano. The notes are randomly chosen by a synthesizer component called Rambler, which restricts each successive note to either one step up or one step down from the previous note. The lack of large jumps makes the resultant stream of notes sound more like a composed (or improvised) melody rather than a purely random one.

As you move the round touch-points up and down, you'll notice about a one-second delay before you actually hear a change in the music's timbre.

Figure 4 shows the OnBeginStreaming override in the AudioStreamingAgent derivative.

I would have preferred to define the assemblage of the synthesizer components in the program itself, and then transfer this setup to the background agent, but given the process isolation between the program task and the background agent tasks, that would require a bit of work. The synthesizer setup would have to be defined entirely in a text string (perhaps XML-based) and then passed from the program to the AudioStreamingAgent derivative through the Tag property of AudioTrack.

Meanwhile, my wish list for future enhancements to Windows Phone includes a facility that lets programs communicate with the background agents they invoke. ■

CHARLES PETZOLD *is a longtime contributor to* MSDN Magazine. *His Web site is charlespetzold.com.*

THANKS to the following technical experts for reviewing this article: *Eric Bie, Mark Hopkins and Chris Pearson*

Touch, Not the Mouse

Microsoft is making a huge amount of noise about touch control of computers. As usual, some of it is justified and some is completely wrong. Perhaps this is a fundamental human trait, throwing any new technology at every problem in sight to see what sticks. But with just a bit of careful thought, and a pause from the mass hysteria, you could save many hours wasted in blind alleys.

Touch is an excellent way to control phones and tablets, which don't have space for keyboards or mice. I bought a non-touch Amazon Kindle years ago, which at the time I liked very much. But I haven't used it since I got my smartphone—the Kindle reader app is just too darn sweet. Tap anywhere on the right side to page forward, the left to page back, the menu button to see other choices on which I tap to select. I can't stand using the old Kindle's hard buttons to page, or the joystick control to select from menus, or holding the extra weight and seeing the wasted space of the seldom-used alphabetic keyboard. Even with a smaller screen, the touch-enabled phone app blows away the non-touch-dedicated device. Fortunately, Santa brought me a new Kindle Touch, which I'm having fun with.

On the other hand, I've been trying to use touchscreens on a PC since around 1981. They don't work well there, because PCs solve different problems than do mobile devices. PCs are used for producing content as much as for consuming content, so users need a keyboard to enter that content. Even the worst keyboard typist uses two fingers, one on each hand. On a PC with a vertical monitor touchscreen, a typist can typically use only one hand, cutting the data input rate by at least 50 percent and, often, far more.

The touch demo programs show kids finger painting on their touchscreens. Hoo-[expletive]-ray. Show me an artist over the age of six who still finger paints. Tablet for fun, yes. PC for work, no.

A PC mouse has single-pixel resolution, and it's easy to handle—just slide your hand to the side and grasp it. To touch a PC screen, you have to lift your entire arm, cantilevered out from your shoulder. This takes much more muscle effort, and the pointing is far less precise—line-of-text resolution at best. Do 10 arm lifts right now, and tell me how you'd like to do that all day, in return for less-precise pointing. As long as you need a PC keyboard, a mouse is the best solution to the pointing problem.

But even on a mobile device, touch isn't the magic bullet. Touch is excellent for selecting among alternatives presented on the screen, but the small keys on a phone's virtual keyboard are slow and error-prone for entering arbitrary data, such as navigation addresses. I recently drove my younger daughter to a gymnastics meet at "Allard Center YMCA, Goffstown, New Hampshire." That would have taken dozens of keystrokes, even with auto-complete, and especially with keying errors due to the small size. With voice recognition, I just spoke it into the microphone and bang! I was on the road. (She placed third all-around in her age group. Good going, Lucy.)

Well, that didn't take long: In January's column ("Lowering Higher Education"), I wrote that university education would shift from a classroom-based model to a Web-based model, hammering those invested in the former. As an example, I cited Stanford's class on Artificial Intelligence, taught by Peter Norvig and Sebastian Thrun, which the school offered free to anyone via the Web. It attracted 58,000 students from around the world.

Less than a month after that column ran, Thrun announced that he's leaving his tenured position to join startup online university Udacity (**udacity.com**). "Having done [the Web AI course], I can't teach at Stanford again," Thrun said in an MSNBC.com article (**tinyurl.com/7qw2gq7**). "You can take the blue pill and go back to your classroom and lecture to your 20 students, but I've taken the red pill and I've seen Wonderland."

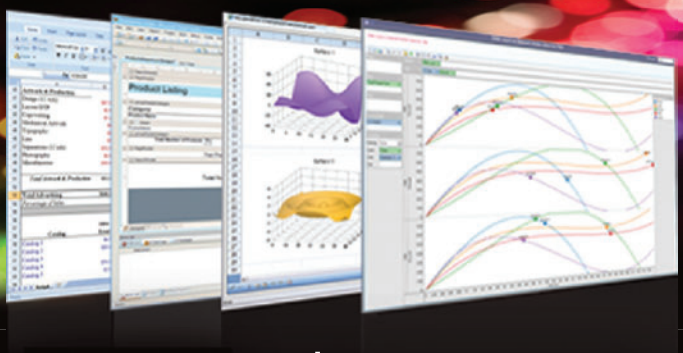
The real win is the combination of touch and voice, using each for what it does best. Arriving at an airport recently for a teaching gig, I got in my rental car, pulled out my phone and tapped Navigate. Then I simply said "Marriott" and the phone presented me with a list of all the nearby Marriotts in order of distance from me. I tapped the one I wanted, and the phone started guiding me there. It doesn't get much sweeter than that.

As Donald Norman wrote in "The Design of Everyday Things" (Doubleday Business, 1990), about encountering objects that are easy to use: "… stop and examine it: the ease of use did not come about by accident. Someone designed the object carefully and well."

Even I, a hardened cynic who's worked on these things for decades, could only shake my head in wonder and say "[expletive] magic." ■

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.*

GrapeCity PowerTools



PS PowerSuite

Enterprise-Ready Reporting, Spreadsheets, Grids, Data Analysis and Charting Components for .NET Applications

GrapeCity PowerSuite

At the core of every business application is data. PowerSuite provides developers with the complete suite of .NET controls to add effective data interaction, decision support and business intelligence capabilities to their .NET applications.

PowerSuite is:

- **ActiveReports** – a fast, flexible and robust reporting engine
- **Spread** – a full-featured Excel® platform providing a complete and familiar spreadsheet experience
- **ActiveAnalysis** – an interactive data analysis control providing pivoting, drill-down and drill-up
- **MultiRow** – an innovative grid component allowing complete data layout customization
- **ActiveChart** – a chart control providing enhanced graphical presentation of data, and more

PS PowerSuite > AR ActiveReports | SP Spread NET | AA ActiveAnalysis | MR MultiRow | AC ActiveChart

GCPowerTools.com/PowerSuite

PS PowerSuite

Standardize on PowerSuite, the premier .NET control suite for building world-class Business Applications

GCPowerTools.com • GvTv.GCPowerTools.com