

msdn[®] magazine

DISTRIBUTED APPS

AppFabric Service Bus Discovery
Juval Lowy 30

Runtime Data Sharing Through an Enterprise Distributed Cache
Iqbal Khan 42

Building a Real-Time Transit Application Using the Bing Map App SDK
Luan Nguyen 50

Connected Devices Using the .NET Micro Framework
Colin Miller 60

PLUS:

Getting Started with Windows Phone Development Tools
Joshua Partlow 70

Scalable Multithreaded Programming with Thread Pools
Ron Fosner 80

COLUMNS

CUTTING EDGE

Action Filters in ASP.NET MVC
Dino Esposito page 6

DATA POINTS

Entity Framework Preview: Code First, ObjectSet and DbContext
Julie Lerman page 14

CLR INSIDE OUT

New Features and Improved Performance in Silverlight 4
Justin Van Patten and Andrew Pardoe page 20

FORECAST: CLOUDY

Performance-Based Scaling in Windows Azure
Joseph Fultz page 86

THE WORKING PROGRAMMER

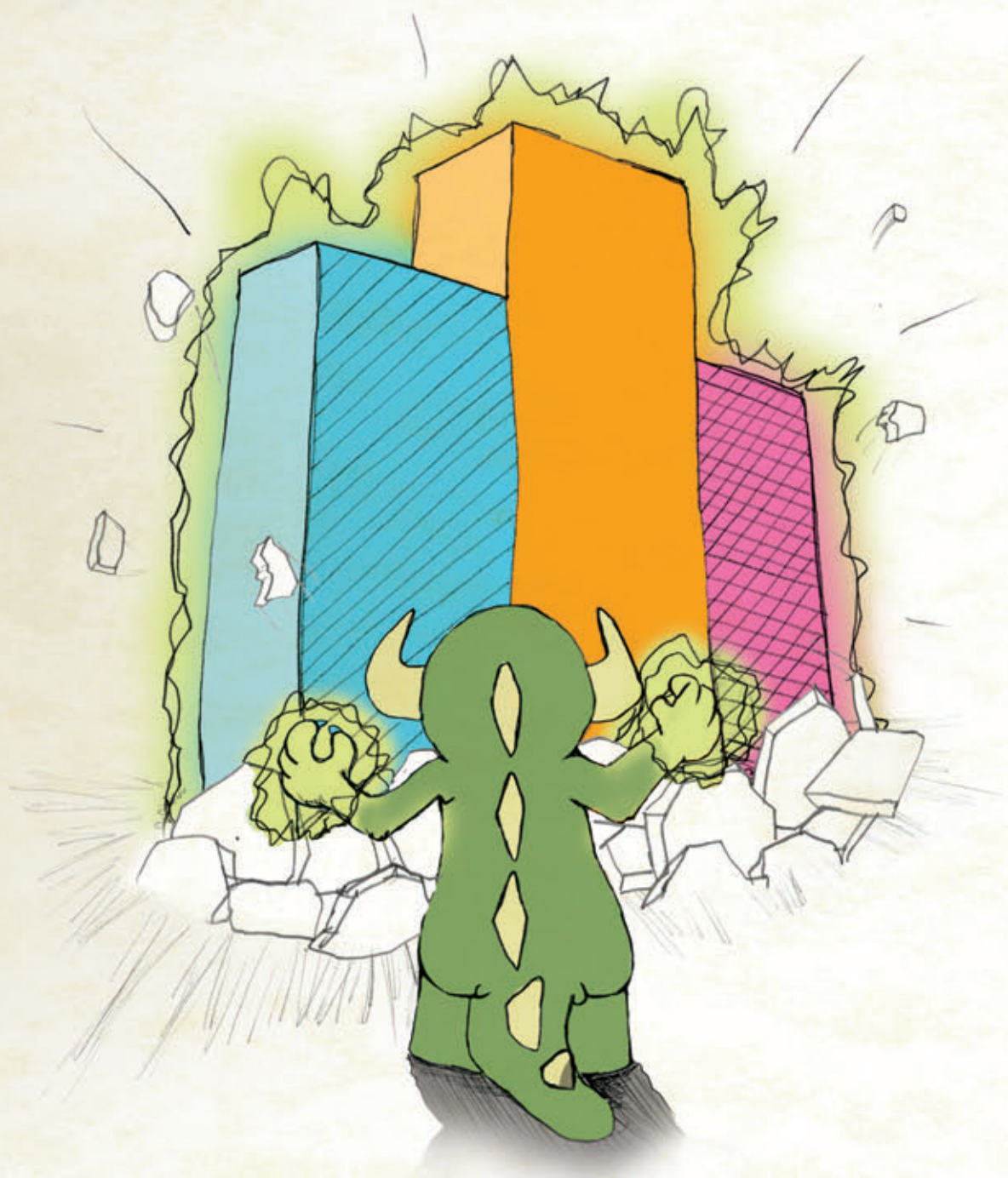
Multiparadigmatic .NET, Part 2
Ted Neward page 91

UI FRONTIERS

Multi-Touch Inertia
Charles Petzold page 95

DON'T GET ME STARTED

Devs and Designers Should Be Friends
David Platt page 100



BREAK OUT OF THE BOX

Sure, Visual Studio 2010 has a lot of great functionality—we're excited that it's only making our User Interface components even better! We're here to help you go beyond what Visual Studio 2010 gives you so you can create Killer Apps quickly, easily and without breaking a sweat! Go to infragistics.com/beyondthebox today to expand your toolbox with the fastest, best-performing and most powerful UI controls available. You'll be surprised by your own strength!

Infragistics Sales 800 231 8588
Infragistics Europe Sales +44 (0) 800 298 9055
Infragistics India +91-80-6785-1111
twitter.com/infragistics



dtSearch®

Instantly Search Terabytes of Text



- ◆ 25+ full-text and fielded data search options

- ◆ Built-in file parsers and converters **highlight hits** in popular file types

- ◆ Spider supports static and dynamic web data; **highlights hits** with links, formatting and images intact

- ◆ API supports C++, .NET, Java, SQL, etc. .NET Spider API. Includes 64-bit (Win/Linux)

- ◆ Fully-functional evaluations available

Content extraction only licenses also available

"Bottom line: dtSearch manages a terabyte of text in a single index and returns results in less than a second" — **InfoWorld**

dtSearch "covers all data sources ... powerful Web-based engines" — **eWEEK**

"Lightning fast ... performance was unmatched by any other product" — **Redmond Magazine**

For hundreds more reviews, and hundreds of developer case studies, see www.dtSearch.com

1-800-IT-FINDS • www.dtSearch.com

The Smart Choice for Text Retrieval® since 1991



msdn®

magazine

OCTOBER 2010 VOLUME 25 NUMBER 10

LUCINDA ROWLEY Director

DIEGO DAGUM Editorial Director/mmeditor@microsoft.com

KERI GRASSL Site Manager

KEITH WARD Editor in Chief/mmeditor@microsoft.com

TERRENCE DORSEY Technical Editor

DAVID RAMEL Features Editor

WENDY GONCHAR Managing Editor

KATRINA CARRASCO Associate Managing Editor

SCOTT SHULTZ Creative Director

JOSHUA GOULD Art Director

ALAN TAO Senior Graphic Designer

CONTRIBUTING EDITORS K. Scott Allen, Dino Esposito, Julie Lerman, Juval Lowy, Dr. James McCaffrey, Ted Neward, Charles Petzold, David S. Platt

Redmond Media Group

Henry Allain President, Redmond Media Group

Matt Morollo Vice President, Publishing

Doug Barney Vice President, Editorial Director

Michele Imgrund Director, Marketing

Tracy Cook Online Marketing Director

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

Matt Morollo VP, Publishing

Chris Kourtoglou Regional Sales Manager

William Smith National Accounts Director

Danna Vedder Microsoft Account Manager

Jenny Hernandez-Asandas Director Print Production

Serena Barnes Production Coordinator/msdnadproduction@1105media.com

1105 MEDIA

Neal Vitale President & Chief Executive Officer

Richard Vitale Senior Vice President & Chief Financial Officer

Michael J. Valenti Executive Vice President

Abraham M. Langer Senior Vice President, Audience Development & Digital Media

Christopher M. Coates Vice President, Finance & Administration

Erik A. Lindgren Vice President, Information Technology & Application Development

Carmel McDonagh Vice President, Attendee Marketing

David F. Myers Vice President, Event Operations

Jeffrey S. Klein Chairman of the Board

MSDN Magazine (ISSN 1528-4859) is published monthly by 1105 Media, Inc., 9201 Oakdale Avenue, Ste. 101, Chatsworth, CA 91311. Periodicals postage paid at Chatsworth, CA 91311-9998, and at additional mailing offices. Annual subscription rates payable in US funds are: U.S. \$35.00, International \$60.00. Annual digital subscription rates payable in U.S. funds are: U.S. \$25.00, International \$25.00. Single copies/back issues: U.S. \$10, all others \$12. Send orders with payment to: MSDN Magazine, P.O. Box 3167, Carol Stream, IL 60132, email MSDNmag@1105service.com or call (847) 763-9560. POSTMASTER: Send address changes to MSDN Magazine, P.O. Box 2166, Skokie, IL 60076. Canada Publications Mail Agreement No: 40612608. Return Undeliverable Canadian Addresses to Circulation Dept. or IMS/NJ. Attn: Returns, 310 Paterson Plank Road, Carlstadt, NJ 07072.

Printed in the U.S.A. Reproductions in whole or part prohibited except by written permission. Mail requests to "Permissions Editor," c/o MSDN Magazine, 16261 Laguna Canyon Road, Ste. 130, Irvine, CA 92618.

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

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

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

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

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

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

Microsoft



Printed in the USA

Your best source for software development tools!

programmer's paradise®



LEADTOOLS Recognition SDK by LEAD Technologies

Develop desktop and server document imaging and ECM applications that require high-speed multi-threaded forms recognition and processing, OCR, ICR, OMR, and barcode technology.

- Supports text, OMR, image, and 1D/2D barcode fields
- Recognize machine print and constrained handwritten text
- Auto-registration and clean-up to improve recognition results
- Latin character set support is included. Arabic and Asian support is available

Paradise #
LO5 26301A01

\$3,214.99

programmers.com/LEAD

STOP OVERBUYING SOFTWARE TODAY!

Eliminate Wasteful Software License Spend:

- Control your software licensing costs
- Stop paying for licenses you're not using
- Reduce your license spend by \$300+ per desktop user



FREE 30-DAY PROOF OF CONCEPT

Learn more:

programmers.com/eliminate-wasteful-license-spend



"Pragma SSH for Windows" Best SSH/SFTP/SCP Servers and Clients for Windows

by Pragma Systems

Get all in one easy to use high performance package. FIPS Certified and Certified for Windows.

- Certified for Windows Server 2008R2
- Compatible with Windows 7
- High-performance servers with centralized management
- Active Directory & GSSAPI authentication
- Supports over 1000 sessions
- Hyper-V and PowerShell support
- Runs in Windows 2008R2/2008/2003/7/Vista/XP/2000

Certified for Windows 7/2008R2

Paradise #
P35 04201A01

\$550.99

programmers.com/pragma

InstallShield Professional for Windows 2011

by Flexera Software

If your software targets Windows®, InstallShield® is your solution. It makes it easy to author high-quality reliable Windows installer (MSI) and InstallScript installations and App-V™ virtual packages for Windows platforms, including Windows 7. InstallShield, the industry standard for MSI installations, also supports the latest Microsoft technologies including Visual Studio 2010, .NET Framework 4.0, IIS7.0, SQL Server 2008 SP1, and Windows Server 2008 R2 and Windows Installer 5, keeping your customers happy and your support costs down.

NEW RELEASE!



Upd from any
Active IS Pro +
IS Pro Silver Mtn
Paradise #
I21H02401B01

\$1,384.99

programmers.com/flexera

VMware vSphere Essentials Kit Bundle

vSphere Essentials provides an all-in-one solution for small offices to virtualize three physical servers for consolidating and managing applications to reduce hardware and operating costs with a low up-front investment. vSphere Essentials includes:

- VMware ESXi and VMware ESX (deployment-time choice)
- VMware vStorage VMFS
- Four-way virtual SMP
- VMware vCenter Server Agent
- VMware vStorage APIs/VCB
- VMware vCenter Update Manager
- VMware vCenter Server for Essentials



for 3 hosts
Paradise #
V55 85101A02

\$446.99

programmers.com/vSphere

BUILD ON VMWARE ESXi AND VSPHERE

for Centralized Management, Continuous Application Availability, and Maximum Operational Efficiency in Your Virtualized Datacenter.

Programmer's Paradise invites you to take advantage of this webinar series sponsored by our TechXtend solutions division.

FREE VIRTUALIZATION WEBINAR SERIES: REGISTER TODAY! TechXtend.com/Webinars



ActiveReports 6 by GrapeCity

Integrate Business Intelligence/Reporting/Data Analysis into your .NET applications using the NEW ActiveReports 6.

- Fast and Flexible reporting engine
- Data Visualization and Layout Controls such as Chart, Barcode and Table Cross Section Controls
- Wide range of Export and Preview formats including Windows Forms Viewer, Web Viewer, Adobe Flash and PDF
- Royalty-Free Licensing for Web and Windows applications

NEW VERSION 6!

Professional Ed.
Paradise #
D03 04301A01

\$1,310.99

programmers.com/grapecity

Microsoft Visual Studio Professional 2010

by Microsoft

Microsoft Visual Studio 2010 Professional with MSDN Essentials Subscription is an integrated environment that simplifies creating, debugging and deploying applications. Unleash your creativity and bring your vision to life with powerful design surfaces and innovative collaboration methods for developers and designers. Work within a personalized environment, targeting a growing number of platforms, including Microsoft SharePoint and cloud applications and accelerate the coding process by using your existing skills. Integrated support for Test-First Development and new debugging tools let you find and fix bugs quickly and easily to ensure high quality solutions.



with MSDN
Paradise #
M47 40201A02

\$1,060.99

programmers.com/microsoft

TX Text Control 15.1

Word Processing Components

TX Text Control is royalty-free, robust and powerful word processing software in reusable component form.

- .NET WinForms control for VB.NET and C#
- ActiveX for VB6, Delphi, VBScript/HTML, ASP
- File formats DOCX, DOC, RTF, HTML, XML, TXT
- PDF and PDF/A export, PDF text import
- Tables, headers & footers, text frames, bullets, structured numbered lists, multiple undo/redo, sections, merge fields, columns
- Ready-to-use toolbars and dialog boxes

New Service Pack!



Professional Edition
Paradise #
T79 02101A02

\$1,220.99

Download a demo today.

programmers.com/theimagingsource

FREE WEBINAR SERIES: MAXIMIZING DATA QUALITY FOR VALUE AND ROI



Data is a company's greatest asset. Enterprises that can harness the power of their data will be strategically positioned for the next business evolution. But too often businesses get bogged down in defining a Data Management process, awaiting some "magic bullet", while the scope of their task grows larger and their data quality erodes. Regardless of how your eventual data management solution is implemented, there are processes that need to occur now to facilitate that process. In this new series, with a mixture of slides, demonstrations and Q&A sessions, we will discuss how to use your existing Data Modeling assets to build the foundations of strong data quality.

REGISTER TODAY! programmers.com/CA

CA ERwin® Data Modeler r7.3 – Product Plus 1 Year Enterprise Maintenance

CA ERwin Data Modeler is a data modeling solution that enables you to create and maintain databases, data warehouses and enterprise data resource models. These models help you visualize data structures so that you can effectively organize, manage and moderate data complexities, database technologies and the deployment environment.

Paradise #
P26 04201E01

\$3,931.99

programmers.com/ca



NEW! Intel® Parallel Studio 2011 by Intel

A comprehensive, all-in-one toolkit for Microsoft Visual Studio® C/C++ developers, Intel® Parallel Studio 2011 simplifies the analysis, compiling, debugging, error-checking, and tuning of your serial and threaded apps.

With Intel Parallel Studio, get everything you need to optimize legacy serial code, exploit multicore, and scale for manycore.

NEW RELEASE!



Single User DVD
Paradise #
I23 63101E03

\$753.99

programmers.com/intel

866-719-1528

programmersparadise.com

Prices subject to change. Not responsible for typographical errors.



A Few of My Favorite App Things

Windows Phone 7 will be out possibly as early as the month you read this—October—although no firm release date had been announced as of this writing. Obviously, Windows Phone 7 represents a lot of risk for Microsoft—and a lot of potential.

There's no getting around the fact that Microsoft is late to the smartphone game, with the iPhone and Android gobbling up market share. But, in my opinion, it's not too late—not by a long shot. Right now the aforementioned competitors are the main game in town: HP hasn't given any clues as to what will happen with the Palm Pre phone since it bought Palm, and the BlackBerry has no juice at all. That means there's absolutely room for another entry, and Microsoft is working hard to make sure Windows Phone 7 becomes that third major player.

To do that, it will all be about—say it with me (and skip a little bit, if you're feeling brave)—developers, developers, developers, developers! Previews of the UI are encouraging, and I love the fact that Microsoft hasn't copied what Apple and Google have done with their interfaces, coming up with something completely different.

But at the end of the day, folks buy smartphones because of the apps they can put on them. It will be crucial for the developer community to have a heaping helping of apps ready to go at launch, and many more soon after. If those apps aren't outstanding—and they need to be, given the competition's head start—it will make the hill that much higher to climb. So, putting on my consumer hat, here's what I think are the key features of any Windows Phone 7 app, in order of importance:

1. **Price.** Studies continue to show that the lowest-priced and free apps are the most popular. That's no revelation, but worth keeping on the front burner of your go-to-market strategy. It's tempting, when you've put in six months of development on something, to assume that customers will want to pony up for your brilliant app. The reality is, however, that you'll have more competition in your space—whatever your space is—than you've probably ever had before. That means making a calculation on whether a competitive price will bring in enough downloads to offset the lower price. I urge you to consider this point very, very

carefully; it may be the thing that makes or breaks your success in the Windows Phone 7 market.

2. **Stability.** No matter how entertaining your game is, how useful your password-management app is, or how revolutionary your guitar-tuner app, if it's buggy from the beginning, it's likely to be unused. And worse, the negative comments on the Microsoft app store (whatever form that store may take) could be fatal. Version 1 apps built with the attitude of “we'll fix it with version 2” may not make it to version 2.

I have a GPS app on my smartphone (the manufacturer shall remain anonymous—I'll be getting Windows Phone 7 as soon as it's out) that works about half the time. When it does work, it's awesome. But it's my least-favorite app, by a mile, because I never know if I'll be able to use it. Again, you'll probably have many competitors in your space; even if their apps are inferior to yours in terms of features, they'll get the sales because they just work.

3. **Complexity, or lack thereof.** Never, ever forget that these apps will be on tiny screens, with users pushing fingers around. A UI that tries to cram in too much is doomed. Simplicity wins the day here. Ask yourself if every menu bar, button and icon must be there. If not, be brutal and delete. You're not going to get—nor do users expect—the kind of UI choices and options available with a typical desktop app. In fact, they'll be expecting the opposite: a clean, spare, easy-to-navigate UI that involves as little reading as possible.
4. **Get it to market early.** You have an opportunity to get your app the kind of attention and publicity that's simply not available with apps from the “other guys.” This is a brand-new market, and now's the time to plant your flag. I'm not saying to rush development and invalidate point No. 2, but instead to move resources around to give your Windows Phone 7 development team the ability to create a polished app that's ready to go when the phone launches.

So get to work. I can't wait to see what you come up with.

Tell me about your app at mmeditor@microsoft.com.

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

© 2010 Microsoft Corporation. All rights reserved.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, you are not permitted to reproduce, store, or introduce into a retrieval system *MSDN Magazine* or any part of *MSDN Magazine*. If you have purchased or have otherwise properly acquired a copy of *MSDN Magazine* in paper format, you are permitted to physically transfer this paper copy in unmodified form. Otherwise, you are not permitted to transmit copies of *MSDN Magazine* (or any part of *MSDN Magazine*) in any form or by any means without the express written permission of Microsoft Corporation.

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

MSDN Magazine is published by 1105 Media, Inc. 1105 Media, Inc. is an independent company not affiliated with Microsoft Corporation. Microsoft Corporation is solely responsible for the editorial contents of this magazine. The recommendations and technical guidelines in *MSDN Magazine* are based on specific environments and configurations. These recommendations or guidelines may not apply to dissimilar configurations. Microsoft Corporation does not make any representation or warranty, express or implied, with respect to any code or other information herein and disclaims any liability whatsoever for any use of such code or other information. *MSDN Magazine*, *MSDN*, and Microsoft logos are used by 1105 Media, Inc. under license from owner.



OnTime Now!

Cloud-based Scrum Tool for Developers

In just seconds, get your team started with the world's most advanced agile project management solution. See why *OnTime Now* is so popular with agile dev teams!

\$79 / month for 5 users • Free 30-day Team Trial



axosoft

Software for Software Development



Local Scrum Tool for Developers

Ship software on time with the locally-installed, award-winning project management tool for agile development teams. *MS-SQL* server back-end.

Starts @ \$395 for teams of 5 • Free for 1 user



NEW!

Subversion for Visual Studio

It's every developer's dream come true when it comes to using *Subversion* with *Microsoft Visual Studio*: fast, keyboard-friendly and stays out of the way!

Just \$49 / user

www.axosoft.com

(800) 653-0024 • (480) 362-1900



Action Filters in ASP.NET MVC

The biggest challenge many software architects face today is how to design and implement an application that can meet all version 1 requirements plus all others that can show up afterward. Maintainability has been one of the fundamental attributes of software design since the first draft of the ISO/IEC 9126 paper, back in 1991. (The paper provides a formal description of software quality and breaks it down into a set of characteristics and sub-characteristics, one of which is maintainability. A PDF version of the paper can be obtained at iso.org.)

The ability to serve a customer's present and future needs certainly isn't a new requirement for any piece of software. However, what many Web applications require today is a subtle and short-term form of maintainability. Many times customers don't want new functions or a different implementation of an existing feature. They just want you to add, replace, configure or remove small pieces of functionality. A typical example is when Web sites with large audiences launch specific advertising campaigns. The overall behavior of the site doesn't change, but extra actions must be performed along with existing actions. Moreover, these changes usually aren't persistent. They must be removed after a few weeks to then be reincorporated a few months later, be configured differently and so forth. You need the ability to program any required functionality by composing small pieces together; you need to track dependencies without greatly impacting the source code; and you need to move toward an aspect-oriented design of software. These are some of the primary reasons behind the rapid adoption of Inversion of Control (IoC) frameworks in many enterprise projects.

So what's this article all about? It isn't meant to be a boring lecture on how software is changing today. Instead, it's an in-depth exploration of a powerful feature of ASP.NET MVC controllers that can notably help you in the building of aspect-oriented Web solutions: ASP.NET MVC action filters.

What's an Action Filter, Anyway?

An action filter is an attribute that, when attached to a controller class or a controller method, provides a declarative means to attach some behavior to a requested action. By writing an action filter, you can hook up the execution pipeline of an action method and adapt it to your needs. In this way, you can also take out of the controller class any logic that doesn't strictly belong to the controller. In doing so, you make this particular behavior reusable and, more importantly, optional. Action filters are ideal for implementing crosscutting concerns that affect the life of your controllers.

ASP.NET MVC comes with a few predefined action filters such as `HandleError`, `Authorize` and `OutputCache`. You use `HandleError` to trap exceptions thrown during the execution of methods on the target controller class. The programming interface of the `HandleError` attribute lets you indicate the error view to associate with a given exception type.

The `Authorize` attribute blocks the execution of a method for unauthorized users. It doesn't distinguish, however, between users who just haven't logged in yet and logged users lacking enough permissions to perform a given action. In the configuration of the attribute, you can specify any roles required to execute a given action.

The `OutputCache` attribute caches the response of the controller's method for the specified duration and requested list of parameters.

An action filter class implements several interfaces. The full list of interfaces is presented in **Figure 1**.

In the most common scenarios, you're mostly concerned with `IActionFilter` and `IResultFilter`. Let's get to know them more closely. Here's the definition of the `IActionFilter` interface:

```
public interface IActionFilter
{
    void OnActionExecuted(ActionExecutedContext filterContext);
    void OnActionExecuting(ActionExecutingContext filterContext);
}
```

You implement the `OnActionExecuting` method to execute code before the controller's action executes; you implement `OnActionExecuted` to post-process the controller state a method has determined. The context objects provide a lot of runtime information. Here's the signature of `ActionExecutingContext`:

```
public class ActionExecutingContext : ControllerContext
{
    public ActionDescriptor ActionDescriptor { get; set; }
    public ActionResult Result { get; set; }
    public IDictionary<string, object> ActionParameters { get; set; }
}
```

The action descriptor, in particular, provides information about the action method, such as its name, controller, parameters, attributes and additional filters. The signature of `ActionExecutedContext` is only a bit different, as shown here:

```
public class ActionExecutedContext : ControllerContext
{
    public ActionDescriptor ActionDescriptor { get; set; }
    public ActionResult Result { get; set; }
    public bool Canceled { get; set; }
    public Exception Exception { get; set; }
    public bool ExceptionHandled { get; set; }
}
```

In addition to a reference to the action description and action result, the class supplies information about an exception that could've occurred and offers two Boolean flags that deserve more attention.

The New WebUI Studio 2010 Is Here.

Introducing 220 industry-leading controls for ASP.NET, Silverlight, and WPF with innovative features and rich user experiences for rapid business application development.

WebUI Studio for Silverlight

Introducing ClientUI, a state-of-the-art suite of 180 MVVM-ready UI controls with amazing user experience. Unlock the new world of RIA with powerful deep link navigation, fluid drag-drop, multiple window interface with task bar, and more.

WebUI Studio for WPF

Take your desktop apps to the next level with our 60 rich UI controls, from stunning 3D cover flow with VirtualFlow™ technology, fluid drag-drop, tool bars, menus, callout, dropdown and split buttons, combo box, accordion, to sophisticated windowing and dock navigation control and more.

WebUI Studio for ASP.NET

Featuring the award-winning WebGrid Enterprise, the new WebEssentials with 8 sleek Web 2.0 widgets, the world's fastest WebScheduler, and 30 more controls, WebUI Studio for ASP.NET lets you ship your ultimate business web apps ahead of the competition.



WebUI Studio®
The World's Premium UI Suite for .NET™

VS 2010 Blend 4

Download your free trial with full support at:
www.webuistudio.net/try

24-Hour Hotline

 1-310-914-0158


 +6221-2555-6680

Figure 1 Interfaces for an Action Filter

Filter Interfaces	Description
IActionFilter	Methods in the interface are invoked before and after the execution of the controller method.
IAuthorizationFilter	Methods in the interface are invoked before the controller method executes.
IExceptionFilter	Methods in the interface are invoked whenever an exception is thrown during the execution of the controller method.
IResultFilter	Methods in the interface are invoked before and after the processing of the action result.

The `ExceptionHandled` flag indicates that your action filter is given a chance to mark an occurred exception as handled. The `Canceled` flag concerns the `Result` property of the `ActionExecutingContext` class.

Note that the `Result` property on the `ActionExecutingContext` class exists only to move the burden of generating any action response from the controller method to the list of registered action filters. If any action filters assign a value to the `Result` property, the target method on the controller class will never be invoked. In doing this, you bypass the target method, moving the burden of producing a response entirely to action filters. However, if multiple action filters were registered for a controller method, then they'd share the action result. When a filter sets the action result, all subsequent filters in the chain will receive the `ActionExecuteContext` object with the property `Canceled` set to true. Whether you set `Canceled` programmatically in the action-executed step, or set the `Result` property in the action-executing step, the target method will never be run.

Writing Action Filters

As mentioned, when it comes to writing custom filters, most of the time you're interested in filters that pre- and post-process the action result and filters that run before and after the execution of the regular controller method. Instead of implementing interfaces natively, an action filter class is commonly derived from `ActionFilterAttribute`:

```
public abstract class ActionFilterAttribute :
    FilterAttribute, IActionFilter, IResultFilter
{
    public virtual void OnActionExecuted(
        ActionExecutedContext filterContext);
    public virtual void OnActionExecuting(
        ActionExecutingContext filterContext);
    public virtual void OnResultExecuted(
        ResultExecutedContext filterContext);
    public virtual void OnResultExecuting(
        ResultExecutingContext filterContext);
}
```

You override `OnActionExecuted` to add some custom code to the execution of the method. You override `OnActionExecuting` as a precondition to the execution of the target method. Finally, you override `OnResultExecuting` and `OnResultExecuted` to place your code around the internal step that governs the generation of the method response.

Figure 2 shows a sample action filter that adds compression programmatically to the response of the method it's applied to.

In ASP.NET, compression is commonly achieved by registering an HTTP module that intercepts any requests and compresses their responses. In the alternative, you can also enable compression at the IIS level. ASP.NET MVC supports both scenarios well and also offers a third option: controlling compression on a per-method

basis. In this way, you can control the level of compression for a specific URL without the need of writing, registering and maintaining an HTTP module.

As you can see in **Figure 2**, the action filter overrides the `OnActionExecuting` method. This may sound a bit weird at first, as you might expect compression to be a crosscutting concern you take care of just before returning some response. Compression is implemented through the `Filter` property of the intrinsic `HttpResponse`. Any response being worked out by the runtime environment is returned to the client browser through the `HttpResponse` object. Subsequently, any custom streams mounted on the default output stream through the `Filter` property can alter the output being sent. Thus, in the course of the `OnActionExecuting` method, you just set up additional streams on top of the default output stream.

When it comes to HTTP compression, however, the most difficult part is taking into careful account the preferences of the browser. The browser sends its compression preferences through the `Accept-Encoding` header. The content of the header indicates that the browser is able to handle only certain encodings—usually `gzip` and `deflate`. To be a good citizen, your action filter must try to

Figure 2 A Sample Action Filter for Compressing the Method Response

```
public class CompressAttribute : ActionFilterAttribute
{
    public override void OnActionExecuting(
        ActionExecutingContext filterContext)
    {
        // Analyze the list of acceptable encodings
        var preferred = GetPreferredEncoding(
            filterContext.HttpContext.Request);

        // Compress the response accordingly
        var response = filterContext.HttpContext.Response;
        response.AppendHeader("Content-encoding", preferred.ToString());

        if (preferredEncoding == CompressionScheme.Gzip)
        {
            response.Filter = new GZipStream(
                response.Filter, CompressionMode.Compress);
        }

        if (preferredEncoding == CompressionScheme.Deflate)
        {
            response.Filter = new DeflateStream(
                response.Filter, CompressionMode.Compress);
        }

        return;
    }

    static CompressionScheme GetPreferredEncoding(
        HttpRequestBase request)
    {
        var acceptableEncoding = request.Headers["Accept-Encoding"];
        acceptableEncoding = SortEncodings(acceptableEncoding);

        // Get the preferred encoding format
        if (acceptableEncoding.Contains("gzip"))
            return CompressionScheme.Gzip;
        if (acceptableEncoding.Contains("deflate"))
            return CompressionScheme.Deflate;

        return CompressionScheme.Identity;
    }

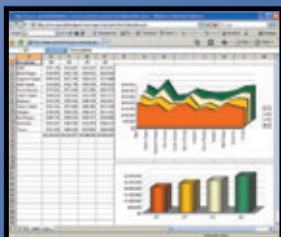
    static String SortEncodings(string header)
    {
        // Omitted for brevity
    }
}
```


Microsoft Chose SpreadsheetGear...

"After carefully evaluating SpreadsheetGear, Excel Services, and other 3rd party options, we ultimately chose SpreadsheetGear for .NET because it is the best fit for MSN Money."

Chris Donohue, MSN Money Program Manager.

SpreadsheetGear 2010



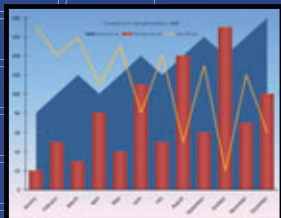
ASP.NET Excel Reporting

Easily create richly formatted Excel reports without Excel using the new generation of spreadsheet technology built from the ground up for scalability and reliability.



Excel Compatible Windows Forms Control

Add powerful Excel compatible viewing, editing, formatting, calculating, charting and printing to your Windows Forms applications with the easy to use WorkbookView control.



Create Dashboards from Excel Charts and Ranges

You and your users can design dashboards, reports, charts, and models in Excel rather than hard to learn developer tools and you can easily deploy them with one line of code.

Download the FREE fully functional 30-Day evaluation of SpreadsheetGear 2010 today at

www.SpreadsheetGear.com.



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

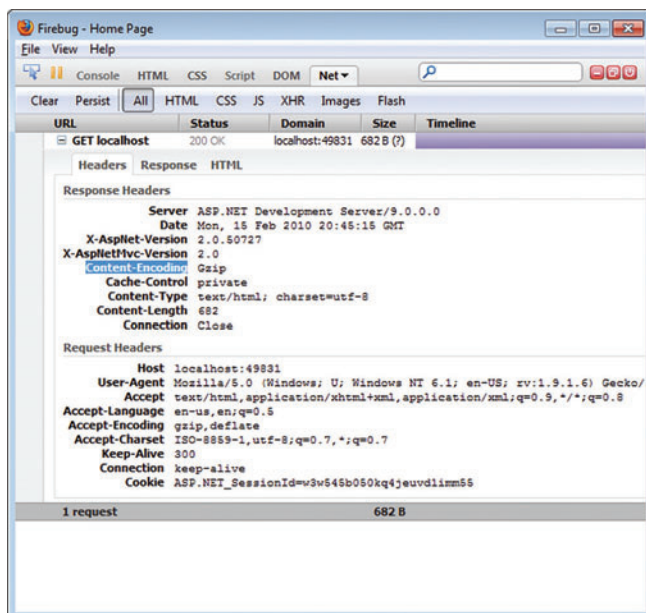


Figure 3 A Compressed Response Obtained via the Compress Attribute

figure out exactly what the browser can handle. This can be a tricky task. The role of the Accept-Encoding header is fully explained in RFC 2616 (see w3.org/Protocols/rfc2616/rfc2616-sec14.html). In brief, the content of the Accept-Encoding header can contain a *q* parameter that's meant to assign a priority to an acceptable value. For example, consider that all the following strings are acceptable values for an encoding, but even though gzip is apparently the first choice in all of them, only in the first one is it the favorite choice:

```
gzip, deflate
gzip;q=.7,deflate
gzip;q=.5,deflate;q=.5,identity
```

A compression filter should take this into account, like the filter in **Figure 2** does. These details should reinforce the idea that when you write an action filter, you're interfering with the handling of the request. Subsequently, anything you do should be consistent with the expectation of the client browser.

Applying an Action Filter

As mentioned, an action filter is just an attribute that can be applied to individual methods as well as to the whole parent class. Here's how you would set it up:

```
[Compress]
public ActionResult List()
{
    // Some code here
    ...
}
```

If the attribute class contains some public properties, you can assign values to them declaratively using the familiar notation of attributes:

```
[Compress(Level=1)]
public ActionResult List()
{
    ...
}
```

Figure 3 shows the compressed response as reported by Firebug.

An attribute is still a static way of configuring a method. This means that you need a second compile step to apply further changes.

However, action filters expressed in the form of attributes provide a key benefit: They keep crosscutting concerns out of the core action method.

A Broader Look at Action Filters

To measure the real power of action filters, think of applications that need a lot of customization work over time and applications that require adaptation when installed for different customers.

Imagine, for example, a Web site that at some point launches an advertising campaign based on reward points a registered user may gain if he performs some standard action within a site (buying goods, answering questions, chatting, blogging and so on). As a developer, you probably need some code that runs after the regular method of executing a transaction, posting a comment or starting a chat has run. Unfortunately, this code is come-and-go and should hardly be part of the original implementation of the core action method. With action filters, you can create distinct components for each required scenario and, for example, arrange an action filter for adding reward points. Next, you attach the reward filter to any methods where the post action is required; then you recompile and go.

```
[Reward(Points=100)]
public ActionResult Post(String post)
{
    // Core logic for posting to a blog
    ...
}
```

As mentioned, attributes are static and require an additional compile step. While this may not be desirable in all cases (say, in sites with highly volatile features), it's much better than nothing. At the minimum, you gain the ability to update Web solutions quickly and with a low impact on existing features, which is always good to keep regression failures under strict control.

Dynamic Loading

This article demonstrated action filters in the context of controller action methods. I demonstrated the canonical approach of writing filters as attributes that you use to decorate action methods statically. There's an underlying question, however: Can you load action filters dynamically?

The ASP.NET MVC framework is a well-written (and large) piece of code, so it exposes a number of interfaces and overridable methods with which you can customize nearly every aspect of it. Happily, this trend is going to be reinforced by the upcoming Model-View-Controller (MVC) 3. According to the public roadmap, one of the team's objectives for MVC 3 is enabling dependency injection at all levels. The answer to the previous question about dynamic loading, therefore, lies in the dependency-injection abilities of the MVC framework. A possible winning strategy is customizing the action invoker to gain access to the list of filters before the execution of a method. Because the list of filters looks like a plain read/write collection object, it shouldn't be that hard to populate it dynamically. But this is good fodder for a new column. ■

DINO ESPOSITO is the author of "Programming ASP.NET MVC" from Microsoft Press (2010) and has coauthored "Microsoft .NET: Architecting Applications for the Enterprise" (Microsoft Press, 2008). Based in Italy, Esposito is a frequent speaker at industry events worldwide. You can join his blog at weblogs.asp.net/despos.

THANKS to the following technical expert for reviewing this article:
Scott Hanselman

version 17

LEADTOOLS®

The World Leader in Imaging Development SDKs



Silverlight, .NET, WPF, WCF, WF, C API, C++ Class Lib, COM & more!

Develop your application with the same robust imaging technologies used by **Microsoft, HP, Sony, Canon, Kodak, GE, Siemens, the US Air Force and Veterans Affairs Hospitals.**

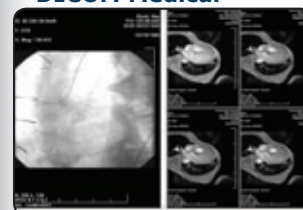
LEADTOOLS provides developers easy access to decades of expertise in color, grayscale, document, medical, vector and multimedia imaging development. Install LEADTOOLS to eliminate months of research and programming time while maintaining high levels of quality, performance and functionality.

- Silverlight:** 100% pure Silverlight 3 and 4 Imaging SDK.
- Image Formats & Compression:** Supports 150+ image formats and compressions including TIFF, EXIF, PDF, JPEG2000, JBIG2 and CCITT G3/G4.
- Display Controls:** ActiveX, COM, Win Forms, Web Forms, WPF and Silverlight.
- Image Processing:** 200+ filters, transforms, color conversion and drawing functions supporting region of interest and extended grayscale data.
- OCR/ICR/OMR:** Full page or zonal recognition for multithreaded 32 and 64 bit development with PDF, PDF/A, XPS, DOC, XML and Text output.
- Forms Recognition & Processing:** Automatically identify and classify forms and extract user filled data.
- Barcode:** Auto-detect, read and write 1D and 2D barcodes for multithreaded 32 & 64 bit development.
- Document Cleanup/Preprocessing:** Auto-deskew, despeckle, hole punch, line and border removal, inverted text correction and more for optimum results in OCR and Barcode recognition.
- PDF & PDF/A:** Read, write and view searchable PDF with text, images, bookmarks and annotations.
- Annotations:** Interactive UI for document mark-up, redaction and image measurement (including support for DICOM annotations).
- Medical Web Viewer Framework:** Plug-in enabled framework to quickly build high-quality, full-featured, web-based medical image delivery and viewer applications.
- PACS Workstation Framework:** Set of .NET PACS components that can be used to build a full featured PACS Workstation application.
- Medical Image Viewer:** High level display control with built-in tools for image mark-up, window level, measurement, zoom/pan, cine, and LUT manipulation.
- DICOM:** Full support for all IOD classes and modalities defined in the DICOM standard (including Encapsulated PDF/CDA and Raw Data).
- PACS Communications:** Full support for DICOM messaging and secure communication enabling quick implementation of any DICOM SCU and SCP services.
- 3D:** Construct 3D volumes from 2D DICOM medical images and visualize with a variety of methods including MIP, MinIP, MRP, VRT and SSD.
- Scanning:** TWAIN & WIA (32 & 64-bit), auto-detect optimum driver settings for high speed scanning.
- DVD:** Play, create, convert and burn DVD images.
- MPEG Transport Stream:** With DVR for UDP and TCP/IP streams & auto-live support.
- Multimedia:** Capture, play, stream and convert MPEG, AVI, WMV, MP4, MP3, OGG, ISO, DVD and more.

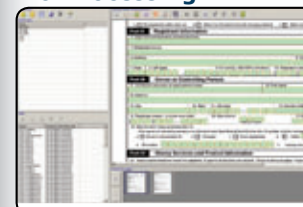
Document



DICOM Medical



Form Recognition & Processing



Barcode



Multimedia



Vector



Free 60 Day Evaluation!
www.leadtools.com/msdn
(800) 637-1840

POWERFUL DATA ACCESS. POINT-AND-CLICK SIMPLICITY.

Presenting Telerik OpenAccess ORM – The easiest way to build your data layer

Powerful Visual Designer

Visualize mapped classes and create a working data layer in minutes with the help of the Visual Designer for OpenAccess ORM.

Forward and Reverse Mapping capabilities

Easily map tables to classes from an existing database or let OpenAccess ORM automatically persist the application classes that you already have.

Saves development time

Create and maintain OpenAccess ORM data layers with minimal effort thanks to tight Visual Studio integration and a rich variety of intuitive wizards.

Full LINQ support

Take full advantage of LINQ for intuitive object querying when building your application with one of the 16 supported databases.

Automatic web service layer

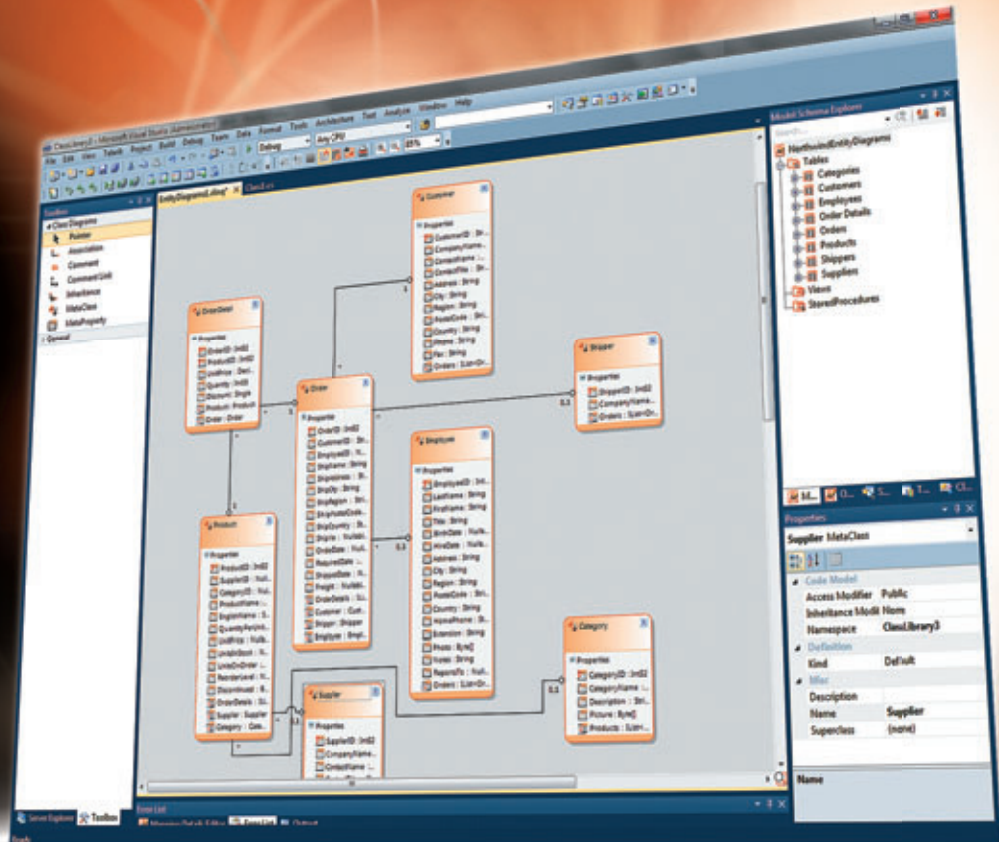
Automatically create a complete Plain WCF, Wcf Data or Wcf Ria web service layer for your OpenAccess ORM project using the built-in Telerik Data Services Wizard.

Silverlight support

Enjoy seamless integration with Silverlight applications with the supported plain WCF services, Data Services, RIA Services technologies.

Optimized for complex scenarios

Tackle even the most complex scenarios with an array of built-in features such as distributed caching, configurable fetch plans, support for n-tier and disconnected applications, support for stored procedures, and many more.



Developer Productivity Tools | Automated Testing Tools | Team Productivity Tools | Web CMS

www.telerik.com/orc

 **telerik**
deliver more than expected



Entity Framework Preview: Code First, ObjectSet and DbContext

While the Entity Framework (EF) 4 was still in beta, the development team began work on another way to use it. We had the database-first way of creating a model (by reverse engineering a database into an Entity Data Model) and the new EF 4 model-first feature (define an Entity Data Model, then create the database from it). In addition, the team decided to create something called “code first.”

With code first you begin with code—not the database and not the model. In fact, with code first there’s no visual model and no XML describing that model. You simply create the classes for your application domain and code first will enable them to participate in the EF. You can use a context, write and execute LINQ to Entities queries and take advantage of EF change-tracking. But there’s no need to deal with a model in a designer. When you aren’t building highly architected apps, you can almost forget that the EF is even there, handling all of your database interaction and change-tracking for you.

An ObjectSet is a generic
collection-like representation
of a set of entity types.

Domain-driven development (DDD) fans seem to love code first. In fact, it was a number of DDD gurus who helped the EF team understand what code first could do for DDD programmers. Check out Danny Simmons’ blog post about the Data Programmability Advisory Council for more about that (blogs.msdn.com/b/dsimmons/archive/2008/06/03/dp-advisory-council.aspx).

But code first wasn’t ready for inclusion in the Microsoft .NET Framework 4 and Visual Studio 2010. In fact, it’s still evolving and Microsoft is providing developers access to the bits to play with and provide feedback through the Entity Framework Feature CTP (EF Feature CTP). The fourth iteration of this community technology preview (CTP4) was released in mid-July 2010. The changes in this iteration were significant.

In addition to including the code-first assembly in the CTP4, Microsoft has been working on some great new ideas for the next iteration of the EF. Those are also included in CTP4 so that people can play with them and, again, provide feedback. More importantly, code first takes advantage of some of these new features.

You can find a number of detailed walkthroughs on this CTP in blog posts by the EF team (tinyurl.com/297xm3j), Scott Guthrie (tinyurl.com/29r3qkb) and Scott Hanselman (tinyurl.com/253dl2g).

In this column, I want to focus on some of the particular features and improvements in the CTP4 that have generated an incredible amount of buzz around the EF because they simplify developing with the EF—and the .NET Framework in general—so dramatically.

Improvements to Core API

One of the noteworthy improvements that the CTP4 adds to the EF runtime is a lightweight version of two workhorse classes,ObjectContext and ObjectSet. ObjectContext enables querying, change-tracking and saving back to the database. ObjectSet encapsulates sets of like objects.

The new classes, DbContext and DbSet, have the same essential functions, but don’t expose you to the entire feature set of ObjectContext and ObjectSet. Not every coding scenario demands access to all of the features in the more robust classes. DbContext and DbSet are not derived from ObjectContext and ObjectSet, but DbContext provides easy access to the full-featured version through a property: DbContext.ObjectContext.

ObjectSet Basics and the Simplified DbSet

An ObjectSet is a generic collection-like representation of a set of entity types. For example, you can have an ObjectSet<Customer> called Customers that’s a property of a context class. LINQ to Entities queries are written against ObjectSets. Here’s a query against the Customers ObjectSet:

```
from customer in context.Customers
where customer.LastName=="Lerman"
select customer
```

ObjectSet has an internal constructor and does not have a parameterless constructor. The way to create an ObjectSet is through the ObjectContext.CreateObjectSet method. In a typical context class, the Customers property would be defined as:

```
public ObjectSet< Customer> Customers {
    get {
        return _customers ?? (_customers =
            CreateObjectSet< Customer >(" Customers"));
    }
}

private ObjectSet< Customer > _customers;
```

This article discusses a prerelease version of the Entity Framework. All information is subject to change.

Blazing-Fast **Grid Controls**

Optimized for ASP.NET



Award-Winning Development Tools by DevExpress



Learn more and *download* your free evaluation copy today
Visit **DEVEXPRESS.COM/ASPXGRID**
or **CALL US (818) 844-3383**

DevExpress
Download • Compare • Decide!

PRESENTATION CONTROLS | REPORTING CONTROLS
IDE PRODUCTIVITY TOOLS | BUSINESS APP FRAMEWORKS
All trademarks and registered trademarks are the property of their respective owners.

Now you can write queries against Customers and manipulate the set, adding, attaching and deleting objects as necessary.

DbSet is much simpler to work with. DbSet isn't publicly constructible (similar to ObjectSet) but it will auto-create DbSets and assign them to any properties (with public setters) you declare on your derived DbContext.

The ObjectSet collection methods are geared toward the EF terminology—AddObject, Attach and DeleteObject:

```
context.Customers.AddObject(myCust)
```

DbSet simply uses Add, Attach and Remove, which is more in line with other collection method names, so you don't have to spend time figuring out "how to say that in the EF," like so:

```
context.Customers.Add(MyCust)
```

My favorite feature of DbSet is that it lets you work more easily with derived types.

The use of the Remove method instead of DeleteObject also more clearly describes what the method is doing. It removes an item from a collection. DeleteObject suggests that the data will be deleted from the database, which has caused confusion for many developers.

By far my favorite feature of DbSet is that it lets you work more easily with derived types. ObjectSet wraps base types and all of the types that inherit from it. If you have a base entity, Contact and a Customer entity that derives from it, every time you want to work with customers, you'll have to start with the Contacts ObjectSet. For example, to query for customers, you write `context.Contacts.OfType<Customer>`. That's not only confusing, it's also definitely not easily discoverable. DbSet lets you wrap the derived types, so now you can create a property Customers that returns `DbSet<Customer>` and enables interaction with that directly rather than through the Contacts set.

DbContext—a Streamlined Context

DbContext exposes the most commonly used features of theObjectContext and provides some additional ones that are truly beneficial. My two favorite features so far of DbContext are the generic Set property and the OnModelCreating method.

All of those ObjectSets I've referred to so far are explicit properties of an ObjectContext instance. For example, say you have a model called PatientHistory that has three entities in it: Patient, Address and OfficeVisit. You'll have a class, PatientHistoryEntities, which inherits ObjectContext. This class contains a Patients property that's an `ObjectSet<Patient>` as well as an Addresses property and an OfficeVisits property. If you want to write dynamic code using generics, you must call `context.CreateObjectSet<T>` where T is one of your entity types. Again, this is just not discoverable.

DbContext has a simpler method called Set that lets you simply call `context.Set<T>`, which will return a `DbSet<T>`. It may only look like 12 less letters, but to my coding brain, using the property feels right, whereas calling a factory method doesn't. You can also use derived entities with this property.

Another DbContext member is OnModelCreating, which is useful in code first. Any additional configurations you want to apply to your domain model can be defined in OnModelCreating just before the EF builds the in-memory metadata based on the domain classes. This is a big improvement over the previous versions of code first. You'll see more about this further on.

Code First Gets Smarter and Easier

Code first was first presented to developers as part of the EF Feature CTP1 in June 2009 with the name "code only." The basic premise behind this variation of using the EF was that developers simply want to define their domain classes and not bother with a physical model. However, the EF runtime depends on that model's XML to coerce queries against the model into database queries and then the query results from the database back into objects that are described by the model. Without that metadata, the EF can't do its job. But the metadata does not need to be in a physical file. The EF reads those XML files once during the application process, creates strongly typed metadata objects based on that XML, and then does all of that interaction with the in-memory XML.

Code first creates in-memory metadata objects, too. But instead of creating it by reading XML files, it infers the metadata from the domain classes (see Figure 1). It uses convention to do this and then provides a means by which you can add additional configurations to further refine the model.

The new code first ModelBuilder class builds the in-memory model.

Another important job of code first is to use the metadata to create a database schema and even the database itself. Code first has provided these features since its earliest public version.

Here's an example of where you'd need a configuration to overcome some invalid assumptions. I have a class called ConferenceTrack with an identity property called TrackId. Code-first convention looks for "Id" or class name + "Id" as an identity property to be used for an entity's EntityKey and a database table's primary key. But TrackId doesn't fit this pattern, so I have to tell the EF that this is my identity key.

The new code first ModelBuilder class builds the in-memory model based on the classes described earlier. You can further define configurations using ModelBuilder. I'm able to specify that the

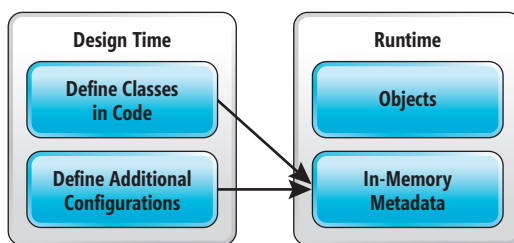


Figure 1 Code First Builds the Entity Data Model Metadata at Run Time

Full-Featured Reporting Controls

Optimized for ASP.NET



Award-Winning Development Tools by DevExpress



Learn more and *download* your free evaluation copy today
Visit **DEVEXPRESS.COM/REPORTS**
or **CALL US (818) 844-3383**

DevExpress
Download • Compare • Decide!

PRESENTATION CONTROLS | REPORTING CONTROLS
IDE PRODUCTIVITY TOOLS | BUSINESS APP FRAMEWORKS
All trademarks and registered trademarks are the property of their respective owners.

ConferenceTrack entity should use its TrackId property as its key with the following ModelBuilder configuration:

```
modelBuilder.Entity<ConferenceTrack>().HasKey(
    ct => ct.TrackId);
```

ModelBuilder will now take this additional information into account as it's creating the in-memory model and working out the database schema.

Applying Configurations More Logically

This is where DbContext.OnModelCreating comes in so nicely. You can place the configurations in this method so that they'll be applied as the model is being created by the DbContext:

```
protected override void OnModelCreating(
    ModelBuilder modelBuilder) {
    modelBuilder.Entity<ConferenceTrack>().HasKey(
        ct => ct.TrackId);
}
```

Another new feature added in the CTP4 is an alternative way to apply configurations through attributes in the classes. This technique is called data annotations. The same configuration can be achieved by applying the Key annotation directly to the TrackId property in the ConferenceTrack class:

```
[Key]
public int TrackId { get; set; }
```

Definitely simpler, however, my personal preference is to use the programmatic configurations so that the classes don't have to have any EF-specific knowledge in them.

Using this approach also means DbContext will take care of caching the model so constructing further DbContexts doesn't incur model discovery cost again.

Relationships Get Easier

One of the most noteworthy improvements to code first is it's much smarter about making assumptions from the classes. While there are many improvements to these conventions, I find the enhanced relationship conventions to have affected my code the most.

Code first can correctly interpret the intent of most relationships defined in classes.

Even though relationships are defined in your classes, in previous CTPs it was necessary to provide configuration information to define these relationships in the model. In addition, the configurations were neither pretty nor logical. Now code first can correctly interpret the intent of most relationships defined in classes. In cases where you need to tweak the model with some configurations, the syntax is much simpler.

My domain classes have a number of relationships defined through properties. ConferenceTrack for example, has this one-to-many relationship:

```
public ICollection<Session> Sessions { get; set; }
```

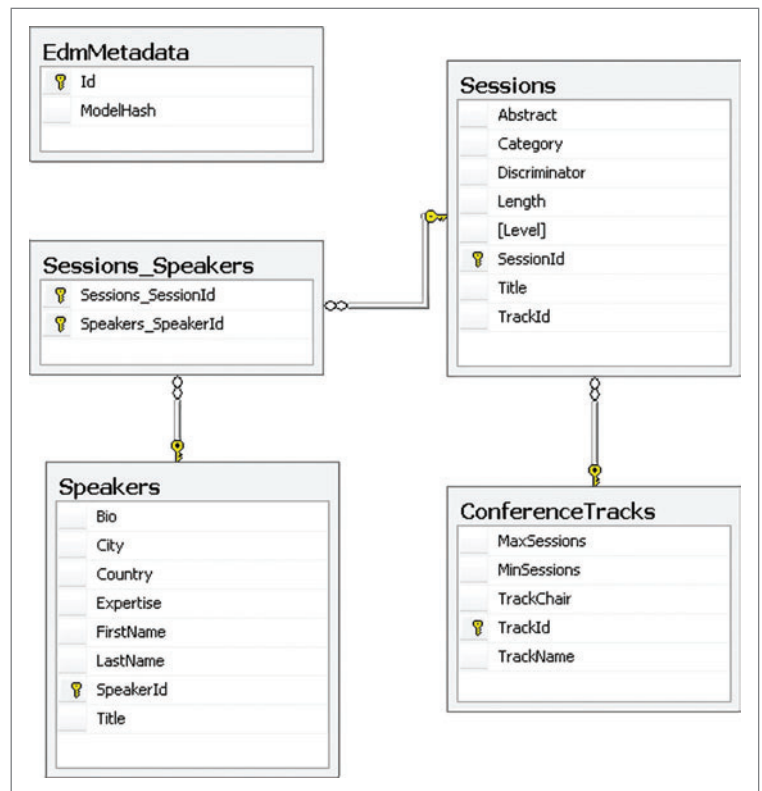


Figure 2 Model Building-Created Database Structure

Session has the converse relationship as well as a many-to-many:

```
public ConferenceTrack ConferenceTrack { get; set; }
public ICollection<Speaker> Speakers { get; set; }
```

Using my classes and a single configuration (to define the TrackId as the key for conferences), the model builder created the database with all of the relationships and inheritances shown in Figure 2.

Notice the Sessions_Speakers table created for the many-to-many relationship. My domain has a Workshop class that inherits from Session. By default, code first assumes Table Per Hierarchy inheritance, so it created a discriminator column in the Sessions table. I can use a configuration to change its name to IsWorkshop or even to specify that it should create a Table per Type inheritance instead.

Planning Ahead for These New Features

These compelling new features that you have early access to in CTP4 might be a source of frustration for developers who are saying "I want it now!" It's true that this is just a preview and you can't put it in production today, but certainly play with these bits if you're able to and provide your feedback to the EF team to help make it even better. You can begin planning ahead for how you'll use code first and the other new EF features in your upcoming applications. ■

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. Lerman blogs at thedatafarm.com/blog and is the author of the highly acclaimed book, "Programming Entity Framework" (O'Reilly Media, 2009). Follow her on Twitter.com: julielerman.

THANKS to the following technical expert for reviewing this article:
Rowan Miller

Powerhouse Analytics Controls

Optimized for ASP.NET



Award-Winning Development Tools by DevExpress



Learn more and *download* your free evaluation copy today
Visit **DEVEXPRESS.COM/ANALYTICS**
or **CALL US (818) 844-3383**

DevExpress
Download • Compare • Decide!

PRESENTATION CONTROLS | REPORTING CONTROLS
IDE PRODUCTIVITY TOOLS | BUSINESS APP FRAMEWORKS
All trademarks and registered trademarks are the property of their respective owners.

New Features and Improved Performance in Silverlight 4

One of the biggest changes in Silverlight 4 was moving to a new version of the CLR for the core execution engine. Every release of the .NET Framework has used the same CLR as its core, from the Microsoft .NET Framework 2.0 through the .NET Framework 3.5 SP1. The .NET Framework 4 made some changes—even some very large changes, such as factoring out the easy-to-download Client Profile and decreasing startup time by optimizing layout of native binaries—but we’ve always been restricted by the high compatibility bar imposed by being an in-place update.

With the .NET Framework 4 release, we were able to make major changes to the CLR itself, while still remaining highly compatible with previous versions. Silverlight 4 uses the new CLR as the basis for its CoreCLR and brings all of its improvements from the desktop to the Web. Some of the most notable runtime improvements are a change in the default garbage collector (GC) behavior and the fact that we no longer just-in-time (JIT) compile Silverlight Framework binaries every time a Silverlight program executes. In the base classes we’ve made improvements throughout, including enhancements to isolated storage and changes in System.IO that enable direct access to the file system from Silverlight applications running with elevated permissions.

Let’s start with a little background on how the CoreCLR GC works.

Generational GC

CoreCLR uses the same GC as the desktop CLR. It’s a generational GC, which means that its operations are based upon the heuristic that the most recently allocated objects are the most likely to be garbage at the next collection time. This heuristic is apparent in small scopes: Function locals are not program-reachable immediately after the function returns. This heuristic is generally applicable for larger scopes as well: Programs usually hold some global state in objects that live the length of the program’s execution.

Objects are usually allocated in the youngest generation—what we refer to as Generation 0—and are promoted during garbage collections (if they survive the collection) to older generations until they reach the maximum generation (Generation 2 in the current CLR GC implementation).

We have another generation in the CLR GC called the Large Object Heap (LOH). Large objects—currently defined as objects greater than 85,000 bytes—are allocated directly into the LOH. This heap is collected at the same time as Generation 2.

Without a generational GC, the GC needs to inspect the entire heap in order to know what memory is reachable and what memory is

garbage before collecting the unused memory. With a generational GC we don’t need to look at the whole heap for every collection. Because the duration of a collection is directly related to the size of the generations being collected, the GC is optimized to only collect Generation 2 (and the LOH) less frequently. Collections are almost instant on small heaps and take longer as the heaps grow—Generation 0 collections can take as little as tens of microseconds.

CoreCLR uses the same garbage collector as the desktop CLR.

For most programs, Generation 2 and the LOH are much larger than Generation 0 and Generation 1, so inspecting all the memory on these heaps takes longer. Keep in mind that the GC always collects Generation 0 when it collects Generation 1, and collects all of the heaps when it collects Generation 2. This is why a Generation 2 collection is called a *full* collection. For more details on the performance of the different heap collections, see the October 2009 CLR Inside Out column at msdn.microsoft.com/magazine/ee309515.

Concurrent GC

The straightforward algorithm to perform a garbage collection is to have the execution engine pause all program threads while the GC does its work. We refer to this kind of a collection as a *blocking* collection. These allow the GC to move non-pinned memory around—for example, to move it from one generation to the next or to compact sparse memory segments—without the program knowing that anything has changed. If memory were to move while program threads are executing, it would look to the program like memory had been corrupted.

But some of the work of a garbage collection doesn’t change memory. Since the first version of the CLR, we’ve provided a GC mode that does *concurrent* collections. These are collections that do most of the work of a full GC without having to pause program threads for the whole duration of that collection.

There are a number of things the GC can do without changing any state that’s visible to the program, for example, the GC can find all program-reachable memory. Program threads can continue to execute while the GC inspects the heaps. Before doing the actual collection, the GC just needs to discover what has changed while

Elegant Scheduling Controls

Optimized for ASP.NET



Award-Winning Development Tools by DevExpress



Learn more and *download* your free evaluation copy today
Visit **DEVEXPRESS.COM/SCHEDULING**
or **CALL US (818) 844-3383**

DevExpress
Download • Compare • Decide!

PRESENTATION CONTROLS | REPORTING CONTROLS
IDE PRODUCTIVITY TOOLS | BUSINESS APP FRAMEWORKS
All trademarks and registered trademarks are the property of their respective owners.

it was inspecting memory—for example, if the program allocated a new object, it needs to be marked as reachable. At the end of this, the GC asks the execution engine to block all threads—just as in a non-concurrent GC—and proceeds to finish all the reachable memory at that point.

The impact of background GC on program latency is significant.

Background GC

Concurrent GC has always provided a great experience in most scenarios, but there's one scenario that we improved greatly. Remember that memory is allocated in the youngest generation or on the LOH. Generations 0 and 1 are located on a single segment—we call it the *ephemeral* segment because it holds short-lived objects. When the ephemeral segment fills up, the program can no longer create new objects because there's no room for them on the ephemeral segment. The GC needs to do a collection on the ephemeral segment to free some space and allow allocation to continue.

The problem with concurrent GC is that it can't do either of these things while a concurrent collection is taking place. The GC thread can't move any memory around while program threads are running (so it can't promote older objects to Generation 2) and because there's already a GC in progress it can't start an ephemeral collection. But the GC needs to free some memory on the ephemeral segment before the program can continue. It's pretty much stuck; the program threads have to be paused, not because concurrent GC is changing program-visible state, but because the program can't allocate. If a concurrent collection finds that the ephemeral segment is full once it's found all reachable memory, it will pause all threads and do a blocking compaction.

This problem explains the motivation behind the development of background GC. It works like the concurrent GC in that the GC always does most of the work of a full collection on its own thread in the background. The main difference is that it allows an ephemeral collection to take place while the full collection gathers data. This means programs can continue to execute when the ephemeral segment fills. The GC just does an ephemeral collection and everything goes on as expected.

The impact of background GC on program latency is significant. When running the background GC, we observed far fewer pauses in program execution and those that remained were shorter in duration.

Background GC is the default mode for Silverlight 4 and is only enabled on Windows platforms because OS X lacks some of the OS support the GC needs to run in background or concurrent mode.

NGen Performance Improvements

The compilers for managed languages such as C# and Visual Basic don't directly produce code that can be executed on the user's

machine. These compilers produce an intermediate language called MSIL that's compiled to executable code at program execution using a JIT compiler.

Using MSIL has a lot of benefits, ranging from security to portability, but there are two tradeoffs with JIT-compiled code. First, a lot of .NET Framework code has to be compiled before your program's Main function can be compiled and executed. This means your user has to wait for the JIT before the program starts running. Second, any .NET Framework code that gets used has to be compiled for every Silverlight program executing on the user's machine.

NGen helps with both of these issues. NGen compiles the .NET Framework code at install time so that it's already compiled when your program starts executing. Code that's been compiled with NGen can often be shared across multiple programs so the working set on the user's machine is reduced when running two or more Silverlight programs. If you want to know more about how NGen improves startup time and working set, see the May 2006 CLR Inside Out column at msdn.microsoft.com/magazine/cc163610.

Code from the .NET Framework makes up a large portion of Silverlight programs, so not having NGen available in Silverlight 2 and 3 made a noticeable difference in startup time. The JIT compiler was taking far too long to optimize and compile the library code on the startup path of every program.

Our solution to this problem was to not have the JIT compiler optimize code generation in Silverlight 2 and 3. The code still needed to be compiled, but because the JIT compiler was producing simple code, it didn't take very long to compile. Compared to traditional desktop applications, most programs written for rich Internet application Web scenarios are small and don't run for very long. Even more importantly, they're usually interactive programs, meaning they spend most of their time waiting for the user's input. In the scenarios targeted by Silverlight 2 and 3, having quick startup time was far more important than generating optimized code.

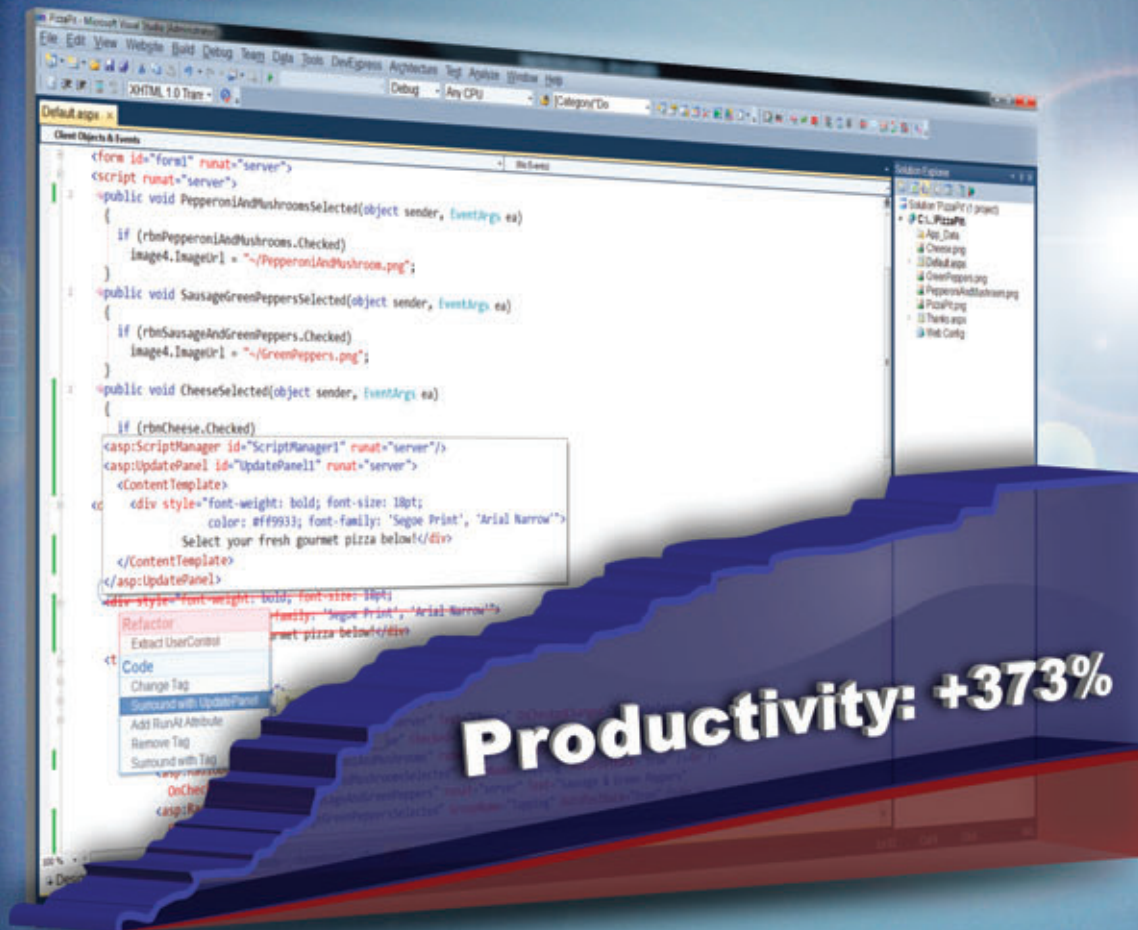
Silverlight 4 gives you both startup performance and optimized code.

As Silverlight Web apps have evolved, we've made changes to keep the experience positive. For example, we added support for installing and running Silverlight applications from the desktop in Silverlight 3. Normally, these applications are larger and do more work than the small, interactive applications found in the classic Web scenario. Silverlight itself has added a lot of computationally intensive functionality, such as support for touch input under Windows 7 and rich photo manipulation as you see on the Bing Maps Web site. All of these scenarios require that the code be optimized to perform efficiently.

Silverlight 4 gives you startup performance and optimized code. The JIT compiler now uses the same optimizations in Silverlight

Rocket Fuel for Visual Studio®

(Superior code, up to 3X faster)



See CodeRush in action: DevExpress.com/373



Learn more and *download* your free evaluation copy today
Visit **DEVEXPRESS.COM/CODERUSH**
or **CALL US (818) 844-3383**

DevExpress™
Download • Compare • Decide!

PRESENTATION CONTROLS | REPORTING CONTROLS
IDE PRODUCTIVITY TOOLS | BUSINESS APP FRAMEWORKS
All trademarks and registered trademarks are the property of their respective owners.

as it does for desktop .NET applications. We're able to do the optimizations because we've enabled NGen for the Silverlight .NET Framework assemblies. When you install Silverlight, we automatically compile all of the managed code that comes in the Silverlight runtime and save it on your hard disk. When a user executes your Silverlight program, it starts executing without having to wait for any of the Framework code to compile. Just as importantly, we now optimize the code in your Silverlight program so that your programs run faster, and we can share the Framework code between multiple Silverlight programs executing on the user's machine.

Isolated storage is a virtual file system that Silverlight applications can use.

Silverlight 4 creates native images for the .NET Framework assemblies during installation. There are some applications where startup performance is the only performance that matters. Think of Notepad as an example: it's important that it starts quickly, but once you start typing it doesn't matter how fast Notepad runs (provided it runs faster than you type). For this class of programs, the time it takes to JIT compile the application startup code may cause a performance decrease. Most applications will start up 400 ms to 700 ms more quickly in Silverlight 4 and will see up to a 60 percent performance improvement during execution.

The Base Class Library (BCL) is at the core of the managed APIs that are now supported by NGen in Silverlight 4. Let's take a look at what's new in the BCL.

New BCL Functionality

Many of the new BCL enhancements in Silverlight 4 are also new in the .NET Framework 4 and have already been written about in that context. I'll give you a brief overview of what we've included in Silverlight 4.

Code contracts provide a built-in way to express pre-conditions, post-conditions and object invariants in your Silverlight code. Code contracts can be used to better express assumptions in your code and can help find bugs early. There are many additional benefits to using code contracts. You can learn more in Melitta Andersen's August 2009 CLR Inside Out column at msdn.microsoft.com/magazine/ee236408, on the Code Contracts DevLabs site at msdn.microsoft.com/devlabs/dd491992, and on the BCL team blog at blogs.msdn.com/bclteam.

Tuples are most often used to return multiple values from a method. They're often used in functional languages such as F# and dynamic languages like IronPython, but are just as easy to use from Visual Basic and C#. You can learn more about the design of tuples in Matt Ellis' July 2009 CLR Inside Out column at msdn.microsoft.com/magazine/dd942829.

`Lazy<T>` provides an easy way to lazily initialize objects. Lazy initialization is a technique that applications can use to defer loading or initializing data until it's first needed.

The new `BigInteger` and `Complex` numeric data types are available in the Silverlight 4 SDK in `System.Numerics.dll`. `BigInteger` represents an arbitrary-precision integer and `Complex` represents a complex number with real and imaginary components.

`Enum`, `Guid` and `Version` now support `TryParse` like many of the other BCL data types, providing a more efficient way to create an instance from a string that doesn't throw exceptions on errors.

`Enum.HasFlag` is a new convenience method that can be used to easily check whether or not a flag is set on a `Flags` enum, without having to remember how to use the bitwise operators.

`String.IsNullOrEmpty` is a convenience method that checks whether or not a string is null, empty or contains only white space.

`String.Concat` and `Join` overloads now take an `IEnumerable<T>` parameter. These new overloads to `String.Concat` and `Join` enable concatenating any collection that implements `IEnumerable<T>` without the need to first convert the collection to an array.

`Stream.CopyTo` makes it easy to read from one stream and write the contents to another stream in one line of code.

In addition to these new features, we've also made some enhancements to isolated storage and enabled trusted Silverlight applications to directly access parts of the file system through `System.IO`.

Isolated Storage Enhancements

Isolated storage is a virtual file system that Silverlight applications can use to store data on the client. To learn more about isolated storage in Silverlight, refer to the March 2009 CLR Inside Out column at msdn.microsoft.com/magazine/dd458794.

The most notable improvement to isolated storage in Silverlight 4 is in the area of performance. Since the release of Silverlight 2, we've received a lot of feedback from developers regarding the performance of isolated storage. In Silverlight 3, we made some changes that significantly improved the performance of reading data out of isolated storage. In Silverlight 4, we've gone a step further and addressed the performance bottlenecks that developers were seeing when writing data to isolated storage. Overall, the performance of isolated storage is much improved in Silverlight 4.

We also heard from developers that there was no easy way to rename or copy files within isolated storage. In order to rename a file, you had to manually read from the original file, create and write to a new file, and then delete the original file. Renaming a directory could be accomplished in a similar manner, but requires even more lines of code, especially when the directory you want to rename contains subdirectories. This works, but is more code than you should have to write and isn't as efficient as telling the OS to simply rename the file or directory on disk.

In Silverlight 4, we've added new methods to the `IsolatedStorageFile` class that you can call to efficiently perform these operations in single line of code: `CopyFile`, `MoveFile` and `MoveDirectory`. We also added new methods that provide additional information about the files and directories within isolated storage: `GetCreationTime`, `GetLastAccessTime` and `GetLastWriteTime`.

Another new API we added in Silverlight 4 is `IsolatedStorageFile.IsEnabled`. Previously, the only way to determine whether isolated storage was enabled was to try using it and then catching the subsequent `IsolatedStorageException`, which is thrown if isolated

FREE ASP.NET Controls

(Yes, they are absolutely free)



Over **20 Free ASP.NET WebForms Controls** Including:

CALLBACK PANEL | MENU | CLOUD CONTROL | NAVBAR | NEWS TICKER
PAGER | POPUP CONTROL | UPLOAD CONTROL | DATA VIEW



Download and Start Using Today
Visit **DEVEXPRESS.COM/FREEASP**

DevExpress
Download • Compare • Decide!

PRESENTATION CONTROLS | REPORTING CONTROLS
IDE PRODUCTIVITY TOOLS | BUSINESS APP FRAMEWORKS

All trademarks and registered trademarks are the property of their respective owners.

storage is disabled. The new static `IsEnabled` property can be used to more easily determine whether or not isolated storage is enabled.

Many browsers such as Internet Explorer, Firefox, Chrome and Safari now support a private browsing mode where browsing history, cookies and other data aren't persisted. Silverlight 4 respects private browsing settings, preventing apps from accessing isolated storage and storing information on your local machine when the browser is in private mode. In such circumstances, the `IsEnabled` property will return false and any attempt to use isolated storage will result in an `IsolatedStorageException`, the same behavior as if isolated storage had been explicitly disabled by the user.

File System Access

Silverlight applications run in a partial-trust security sandbox. The security sandbox restricts access to the local machine and places a number of constraints on the application, preventing malicious code from causing harm. For example, partial-trust Silverlight applications cannot directly access the file system. If an app needs to store data on the client, its only option is to store data within isolated storage. Access to the broader file system can only be accomplished through the `OpenFileDialog` or `SaveFileDialog`.

Silverlight 4 allows
out-of-browser applications
to configure themselves to
run in elevated trust.

Silverlight 3 added the ability to install and run apps out-of-browser. This enables some interesting offline scenarios, but such apps still run within the same sandbox as apps running inside the browser. Silverlight 4 allows out-of-browser applications to configure themselves to run in elevated trust. Such trusted applications are able to bypass some of the restrictions of the sandbox after installation. For example, trusted applications can access user files, use networking without cross-domain access restrictions, bypass user consent and initiation requirements, and access native OS functionality.

When a user installs an application that requires elevated trust, the normal installation prompt is replaced with a warning that tells users that the application can access user data and should only be installed from trusted Web sites.

Trusted applications can use the APIs in `System.IO` to directly access the following user directories on the file system: `MyDocuments`, `MyMusic`, `MyPictures` and `MyVideos`. File operations outside of these directories are currently not allowed and will result in a `SecurityException`. Within these directories, all file operations are allowed, including reading and writing. For example, a trusted photo album application can directly access all files within the `MyPictures` directory. A trusted video-editing app can save a movie to the `MyVideos` directory.

It's important not to hardcode file system paths to these directories in your applications as the paths will be different depending on the underlying OS. File system paths are absolutely different between Windows and Mac OS X, but paths can also be different between versions of Windows. To work correctly across all platforms, `System.Environment.GetFolderPath` should be used to get the file system paths for these directories. The following code uses `Environment.GetFolderPath` to get the file system path to the `MyPictures` directory, finds all files within `MyPictures` (and subdirectories) that end with `.jpg` using the `System.Directory.EnumerateFiles` method, and adds each file path to a `ListBox`:

```
if (Application.Current.HasElevatedPermissions) {
    string myPictures = Environment.GetFolderPath(
        Environment.SpecialFolder.MyPictures);
    IEnumerable<string> files =
        Directory.EnumerateFiles(myPictures, "*.jpg",
            SearchOption.AllDirectories);
    foreach (string file in files) {
        listBox1.Items.Add(file);
    }
}
```

This code shows how to create a text file in the user's `MyDocuments` directory from a trusted application:

```
if (Application.Current.HasElevatedPermissions) {
    string myDocuments = Environment.GetFolderPath(
        Environment.SpecialFolder.MyDocuments);
    string filename = "hello.txt";
    string file = Path.Combine(myDocuments, filename);

    try {
        File.WriteAllText(file, "Hello World!");
    }
    catch {
        MessageBox.Show("An error occurred.");
    }
}
```

`System.IO.Path.Combine` is used to combine the path to `MyDocuments` with the name of the file, which will insert the appropriate directory separator character for the underlying platform between the two (Windows uses `\`, while Mac uses `/`). `File.WriteAllText` is used to create the file (or overwrite it if it already exists) and write the text "Hello World!" to the file.

Better Performance and More Capabilities

As you've seen, the new CLR in Silverlight 4 includes improvements in both the runtime and base classes. The new GC behavior, the fact that we now NGen the Silverlight Framework assemblies, and the isolated storage performance improvements mean your apps will start faster and run better on Silverlight 4. Enhancements to the BCL enable apps to do more with less code, and new capabilities, such as the ability for trusted applications to access the file system, facilitate compelling new app scenarios. ■

ANDREW PARDOE is a program manager for CLR at Microsoft. He works on many aspects of the execution engine for both the desktop and Silverlight runtimes. He can be reached at andrew.pardoe@microsoft.com.

JUSTIN VAN PATTEN is a program manager on the CLR team at Microsoft, where he works on the Base Class Libraries. You can reach him via the BCL team blog at blogs.msdn.com/bclteam.

THANKS to the following technical experts for reviewing this article:
Surupa Biswas, Vance Morrison and Maoni Stephens

We didn't invent the Internet...

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



applications

powered by 

connectivity

powered by 

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

Each day, as you click around the Web or use any connected application, chances are that directly or indirectly some bits are flowing through applications that use our components, on a server, on a device, or right on your desktop. It's your code and our code working together to move data, information, and business. We give you the most robust suite of components for adding Internet Communications, Security, and E-Business Connectivity to

any application, on any platform, anywhere, and you do the rest. Since 1994, we have had one goal: to provide the very best connectivity solutions for our professional developer customers. With more than 100,000 developers worldwide using our software and millions of installations in almost every Fortune 500 and Global 2000 company, our business is to connect business, one application at a time.

To learn more please visit our website →

www.nsoftware.com

Studio Enterprise has 8,310,799 lines of code and would cost

\$710,467,541

and take 10 Years to develop*

*Based on the Constructive Cost Model (COCOMO)
<http://en.wikipedia.org/wiki/COCOMO>

Do you develop for the web?

Our ASP.NET
AJAX &
Silverlight controls
would cost
\$159,009,712



Do you have mobile versions of your ASP.NET apps?

Building
our Mobile Web
controls would
cost
\$3,217,145



Only build desktop apps?

Our
WinForms
& WPF controls
would cost
\$233,189,875



Grids • Charting • Reporting
Scheduling • Menus & Toolbars
Ribbon • Data Input • Editors • PDF



Welcome to

DEVTOPIA

Studio Enterprise saves you serious time and money with tools for every platform, covering every need from world-famous grids to rich data visualization controls. With over 23 years of control development experience, we bring you Devtopia. Get quality, cost saving controls and a community of support to assist you along your dev path.

67
WinForms Controls

59
Silverlight Controls

46
ASP.NET AJAX Controls

40
WPF Controls

34
ActiveX Controls

29
iPhone & Mobile Controls

275
CONTROLS in
Studio Enterprise

Learning Made Easy.
Documentation Pages per Studio

3,186 pgs
iPhone/Mobile

7,758 pgs
ASP.NET
AJAX

8,111 pgs
WPF

7,981 pgs
WinForms

43,539 pgs
Studio Enterprise

Get developing right away.

2,791
WinForms

847
Silverlight

513
WPF

1,340
ActiveX

706
ASP.NET
AJAX

691
iPhone
& Mobile

Studio Enterprise Code Snippets
6,888

Code snippets per Studio

Check out the demo with source code

STUDIO
ENTERPRISE 2010 ^{New!} v2

DOWNLOAD TODAY @ COMPONENTONE.COM/DEVTOPIA



AppFabric Service Bus Discovery

Juval Lowy

In my [January 2010 article](#), “Discover a New WCF with Discovery” ([msdn.microsoft.com/magazine/ee335779](#)), I presented the valuable discovery facility of Windows Communication Foundation (WCF) 4. WCF Discovery is fundamentally an intranet-oriented technique, as there’s no way to broadcast address information across the Internet.

Yet the benefits of dynamic addresses and decoupling clients and services on the address axis would apply just as well to services that rely on the service bus to receive client calls.

Fortunately, you can use events relay binding to substitute User Datagram Protocol (UDP) multicast requests and provide for discovery and announcements. This enables you to combine

the benefit of easy deployment of discoverable services with the unhindered connectivity of the service bus. This article walks through a small framework I wrote to support discovery over the service bus—bringing it on par with the built-in support for discovery in WCF—along with my set of helper classes. It also serves as an example of rolling your own discovery mechanism.

AppFabric Service Bus Background

If you’re unfamiliar with the AppFabric Service Bus, you can read these past articles:

- “Working With the .NET Service Bus,” April 2009
[msdn.microsoft.com/magazine/dd569756](#)
- “Service Bus Buffers,” May 2010
[msdn.microsoft.com/magazine/ee336313](#)

Solution Architecture

For the built-in discovery of WCF, there are standard contracts for the discovery exchange. Unfortunately, these contracts are defined as internal. The first step in a custom discovery mechanism is to define the contracts used for discovery request and callbacks. I defined the `IServiceBusDiscovery` contract as follows:

```
[ServiceContract]
public interface IServiceBusDiscovery
{
    [OperationContract(IsOneWay = true)]
    void OnDiscoveryRequest(string contractName, string contractNamespace,
        Uri[] scopesToMatch, Uri replayAddress);
}
```

This article discusses:

- Architecture of the discovery mechanism
- The discovery host
- The discovery client
- Helper classes
- Announcements
- The Metadata Explorer tool

Technologies discussed:

Windows Communication Foundation, AppFabric Service Bus

Code download available at:

[code.msdn.microsoft.com/mag201010Discovery](#)

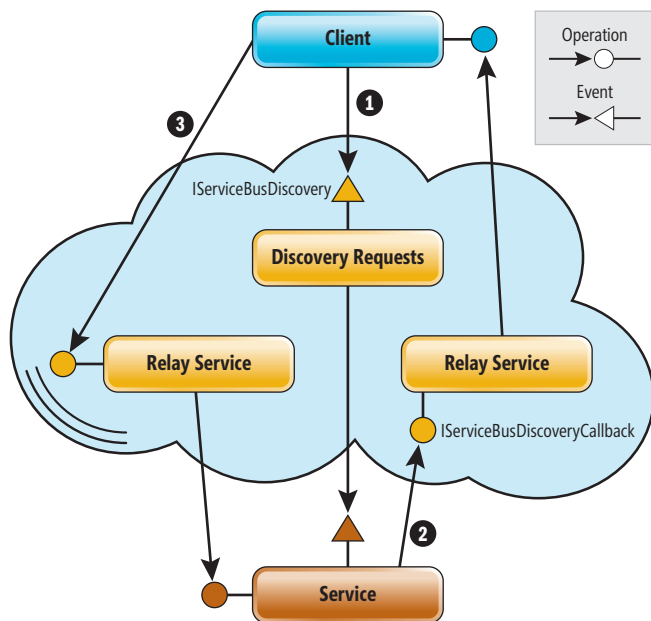


Figure 1 Discovery over the Service Bus

The single-operation `IServiceBusDiscovery` is supported by the discovery endpoint. `OnDiscoveryRequest` allows the clients to discover service endpoints that support a particular contract, as with regular WCF. The clients can also pass in an optional set of scopes to match.

Services should support the discovery endpoint over the events relay binding. A client fires requests at services that support the discovery endpoint, requesting the services call back to the client's provided reply address.

The services call back to the client using the `IServiceBusDiscoveryCallback`, defined as:

```
[ServiceContract]
public interface IServiceBusDiscoveryCallback
{
    [OperationContract(IsOneWay = true)]
    void DiscoveryResponse(Uri address, string contractName,
        string contractNamespace, Uri[] scopes);
}
```

The client provides an endpoint supporting `IServiceBusDiscoveryCallback` whose address is the `replyAddress` parameter of `OnDiscoveryRequest`. The binding used should be the one-way relay binding to approximate unicast as much as possible. **Figure 1** depicts the discovery sequence.

The first step in **Figure 1** is a client firing an event of discovery request at the discovery endpoint supporting `IServiceBusDiscovery`. Thanks to the events binding, this event is received by all discoverable services. If a service supports the requested contract, it calls back to the client through the service bus (step 2 in **Figure 1**). Once the client receives the service endpoint (or endpoints) addresses, it proceeds to call the service as with a regular service bus call (step 3 in **Figure 1**).

Discoverable Host

Obviously a lot of work is involved in supporting such a discovery mechanism, especially for the service. I was able to encapsulate that with my `DiscoverableServiceHost`, defined as:

```
public class DiscoverableServiceHost : ServiceHost, ...
{
    public const string DiscoveryPath = "DiscoveryRequests";

    protected string Namespace { get; }

    public Uri DiscoveryAddress { get; set; }

    public NetEventRelayBinding DiscoveryRequestBinding { get; set; }

    public NetOnewayRelayBinding DiscoveryResponseBinding { get; set; }

    public DiscoverableServiceHost(object singletonInstance,
        params Uri[] baseAddresses);
    public DiscoverableServiceHost(Type serviceType,
        params Uri[] baseAddresses);
}
```

Besides discovery, `DiscoverableServiceHost` always publishes the service endpoints to the service bus registry. To enable discovery, just as with regular WCF discovery, you must add a discovery behavior and a WCF discovery endpoint. This is deliberate, so as to both avoid adding yet another control switch and to have in one place a single consistent configuration where you turn on or off all modes of discovery.

You use `DiscoverableServiceHost` like any other service relying on the service bus:

```
Uri baseAddress =
    new Uri("sb://MyServiceNamespace.servicebus.windows.net/MyService/");

ServiceHost host = new DiscoverableServiceHost(typeof(MyService), baseAddress);

// Address is dynamic
host.AddServiceEndpoint(typeof(IMyContract), new NetTcpRelayBinding(),
    Guid.NewGuid().ToString());

// A host extension method to pass creds to behavior
host.SetServiceBusCredentials(...);

host.Open();
```

Note that when using discovery, the service address can be completely dynamic.

Figure 2 provides the partial implementation of pertinent elements of `DiscoverableServiceHost`.

The helper property `IsDiscoverable` of `DiscoverableServiceHost` returns true only if the service has a discovery behavior and at least one discovery endpoint. `DiscoverableServiceHost` overrides the `OnOpening` method of `ServiceHost`. If the service is to be discoverable, `OnOpening` calls the `EnableDiscovery` method.

`EnableDiscovery` is the heart of `DiscoverableServiceHost`. It creates an internal host for a private singleton class called `DiscoveryRequestService` (see **Figure 3**).

The constructor of `DiscoveryRequestService` accepts the service endpoints for which it needs to monitor discovery requests (these are basically the endpoints of `DiscoverableServiceHost`).

`EnableDiscovery` then adds to the host an endpoint implementing `IServiceBusDiscovery`, because `DiscoveryRequestService` actually responds to the discovery requests from the clients. The address of the discovery endpoint defaults to the URI "DiscoveryRequests" under the service namespace. However, you can change that before opening `DiscoverableServiceHost` to any other URI using the `DiscoveryAddress` property. Closing `DiscoverableServiceHost` also closes the host for the discovery endpoint.

Figure 3 lists the implementation of `DiscoveryRequestService`.

`OnDiscoveryRequest` first creates a proxy to call back the discovering client. The binding used is a plain `NetOnewayRelayBinding`, but you can control that by setting the `DiscoveryRelayBinding`.

sponseBinding property. Note that DiscoverableServiceHost has a corresponding property just for that purpose. OnDiscoveryRequest then iterates over the collection of endpoints provided to the constructor. For each endpoint, it checks that the contract matches the requested contract in the discovery request. If the contract matches, OnDiscoveryRequest looks up the scopes associated with the endpoint and verifies that those scopes match the optional scopes in the discovery request. Finally, OnDiscoveryRequest calls back the client with the address, contract and scope of the endpoint.

Discovery Client

For the client, I wrote the helper class ServiceBusDiscoveryClient, defined as:

```
public class ServiceBusDiscoveryClient : ClientBase<IServiceBusDiscovery>,...
{
    protected Uri ResponseAddress
    {get;}

    public ServiceBusDiscoveryClient(string serviceNamespace,string secret);

    public ServiceBusDiscoveryClient(string endpointName);
    public ServiceBusDiscoveryClient(NetOnewayRelayBinding binding,
        EndpointAddress address);

    public FindResponse Find(FindCriteria criteria);
}
```

Figure 2 Implementing DiscoverableServiceHost (Partial)

```
public class DiscoverableServiceHost : ServiceHost,...
{
    Uri m_DiscoveryAddress;
    ServiceHost m_DiscoveryHost;

    // Extracts the service namespace out of the endpoints or base addresses
    protected string Namespace
    {
        get
        { ... }
    }

    bool IsDiscoverable
    {
        get
        {
            if(Description.Behaviors.Find<ServiceDiscoveryBehavior>() != null)
            {
                return Description.Endpoints.Any(endpoint =>
                    endpoint is DiscoveryEndpoint);
            }
            return false;
        }
    }

    public Uri DiscoveryAddress
    {
        get
        {
            if(m_DiscoveryAddress == null)
            {
                m_DiscoveryAddress =
                    ServiceBusEnvironment.CreateServiceUri("sb",Namespace,DiscoveryPath);
            }
            return m_DiscoveryAddress;
        }
        set
        {
            m_DiscoveryAddress = value;
        }
    }

    public DiscoverableServiceHost(Type serviceType,params Uri[]
        baseAddresses) : base(serviceType,baseAddresses)
    {}
}
```

I modeled ServiceBusDiscoveryClient after the WCF DiscoveryClient, and it's used much the same way, as shown in Figure 4.

ServiceBusDiscoveryClient is a proxy for the IServiceBusDiscovery discovery events endpoint. Clients use it to fire the discovery request at the discoverable services. The discovery endpoint address defaults to "DiscoveryRequests," but you can specify a different address using any of the constructors that take an endpoint name or an endpoint address. It will use a plain instance of NetOnewayRelayBinding for the discovery endpoint, but you can specify a different binding using any of the constructors that take an endpoint name or a binding instance. ServiceBusDiscoveryClient supports cardinality and discovery timeouts, just like DiscoveryClient.

Figure 5 shows partial implementation of ServiceBusDiscoveryClient.

The Find method needs to have a way of receiving callbacks from the discovered services. To that end, every time it's called, Find opens and closes a host for an internal synchronized singleton class called DiscoveryResponseCallback. Find adds to the host an endpoint supporting IServiceBusDiscoveryCallback. The constructor of DiscoveryResponseCallback accepts a delegate of the type Action<Uri,Uri[]>. Every time a service responds back, the implementation of DiscoveryResponse invokes that delegate,

```
void EnableDiscovery()
{
    // Launch the service to monitor discovery requests
    DiscoveryRequestService discoveryService =
        new DiscoveryRequestService(Description.Endpoints.ToArray());

    discoveryService.DiscoveryResponseBinding = DiscoveryResponseBinding;

    m_DiscoveryHost = new ServiceHost(discoveryService);

    m_DiscoveryHost.AddServiceEndpoint(typeof(IServiceBusDiscovery),
        DiscoveryRequestBinding, DiscoveryAddress.AbsoluteUri);

    m_DiscoveryHost.Open();
}

protected override void OnOpening()
{
    if(IsDiscoverable)
    {
        EnableDiscovery();
    }

    base.OnOpening();
}

protected override void OnClosed()
{
    if(m_DiscoveryHost != null)
    {
        m_DiscoveryHost.Close();
    }

    base.OnClosed();
}

[ServiceBehavior(InstanceContextMode = InstanceContextMode.Single,...)]
class DiscoveryRequestService : IServiceBusDiscovery
{
    public DiscoveryRequestService(ServiceEndpoint[] endpoints);

    public NetOnewayRelayBinding DiscoveryResponseBinding
    {get;set;}
}
```


WE ARE SPREADSHEETS

Act now and save up to \$300!

 GrapeCity PowerTools

SPREAD⁵

Windows Forms & ASP.NET

Award-winning Microsoft® Excel® compatible
spreadsheet components for .NET and ASP.NET

- World's best-selling .NET spreadsheet technology
- Hundreds of Chart styles for data visualization
- Full featured Formula support, including most Excel functions
- Full support for native Microsoft Excel files and data import/export
- Spreadsheet Designers, Quick-start Wizard and Chart Wizards



GCPowerTools.com/ActNow



WE ARE
GRAPECITY
Excel  Report  Analyze

1-800-645-5913 / 1-919-460-4551

WE ARE

REPORTING

Act now and save up to \$300!

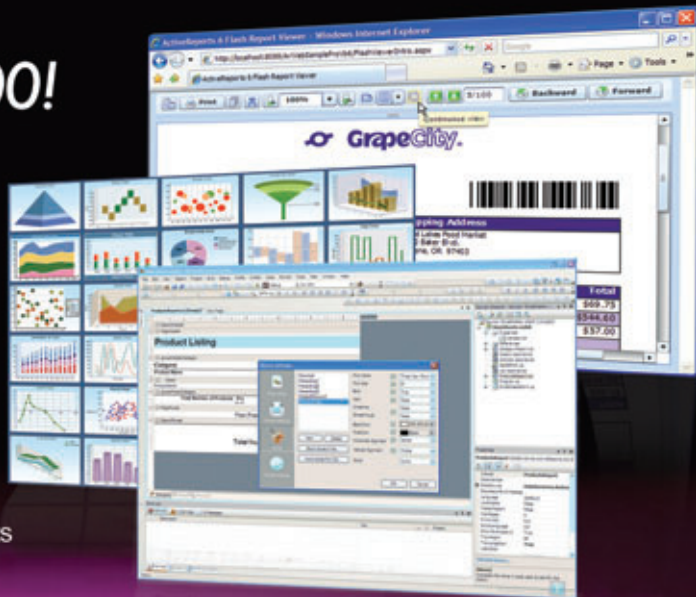
 GrapeCity PowerTools

ACTIVE REPORTS 6

Powerful Award-winning .NET reporting tool for Microsoft Visual Studio

- Unmatched customization, performance and quality
- Rich collection of report designing and rendering components
- Medium Trust environment support in ASP.NET
- Wide range of cross-platform export and preview formats
- Powerful PDF reporting with digital signatures, timestamps and multilanguage support

GCPowerTools.com/ActNow



WE ARE
GRAPECITY
Excel  Report  Analyze

1-800-645-5913 / 1-919-460-4551

Figure 3 The `DiscoveryRequestService` Class (Partial)

```
public class DiscoverableServiceHost : ServiceHost
{
    [ServiceBehavior(InstanceContextMode = InstanceContextMode.Single,
        UseSynchronizationContext = false)]
    class DiscoveryRequestService : IServiceBusDiscovery
    {
        readonly ServiceEndpoint[] Endpoints;

        public NetOneWayRelayBinding DiscoveryResponseBinding
        {get;set;}

        public DiscoveryRequestService(ServiceEndpoint[] endpoints)
        {
            Endpoints = endpoints;
        }

        void IServiceBusDiscovery.OnDiscoveryRequest(string contractName,
            string contractNamespace, Uri[] scopesToMatch, Uri responseAddress)
        {
            ChannelFactory<IServiceBusDiscoveryCallback> factory =
                new ChannelFactory<IServiceBusDiscoveryCallback>{
                    DiscoveryResponseBinding, new EndpointAddress(responseAddress)};

            IServiceBusDiscoveryCallback callback = factory.CreateChannel();

            foreach(ServiceEndpoint endpoint in Endpoints)
            {
                if(endpoint.Contract.Name == contractName &&
                    endpoint.Contract.Namespace == contractNamespace)
                {
                    Uri[] scopes = DiscoveryHelper.LookupScopes(endpoint);

                    if(scopesToMatch != null)
                    {
                        bool scopesMatched = true;
                        foreach(Uri scope in scopesToMatch)
                        {
                            if(scopes.Any(uri => uri.AbsoluteUri == scope.AbsoluteUri)
                                == false)
                                break;
                        }
                    }
                    if(scopesMatched == false)
                    {
                        continue;
                    }
                    callback.DiscoveryResponse(endpoint.Address.Uri, contractName,
                        contractNamespace, scopes);
                }
            }
            (callback as ICommunicationObject).Close();
        }
    }
}

public static class DiscoveryHelper
{
    static Uri[] LookupScopes(ServiceEndpoint endpoint)
    {
        Uri[] scopes = new Uri[0];
        EndpointDiscoveryBehavior behavior =
            endpoint.Behaviors.Find<EndpointDiscoveryBehavior>();
        if(behavior != null)
        {
            if(behavior.Scopes.Count > 0)
            {
                scopes = behavior.Scopes.ToArray();
            }
        }
        return scopes;
    }
    // More members
}
```

providing it with the discovered address and scope. The Find method uses a lambda expression to aggregate the responses in an instance of `FindResponse`. Unfortunately, there's no public constructor for `FindResponse`, so Find uses the `CreateFindResponse` method of `DiscoveryHelper`, which in turn uses reflection to instantiate it. Find also creates a waitable event handle. The lambda expression signals that handle when the cardinality is met. After calling `DiscoveryRequest`, Find waits for the handle to be signaled, or for the discovery duration to expire, and then it aborts the host to stop processing any discovery responses in progress.

More Client-Side Helper Classes

Although I wrote `ServiceBusDiscoveryClient` to be functionally identical to `DiscoveryClient`, it would benefit from a streamlined discovery experience offered by my `ServiceBusDiscoveryHelper`:

```
public static class ServiceBusDiscoveryHelper
{
    public static EndpointAddress DiscoverAddress<T>(
        string serviceNamespace, string secret, Uri scope = null);

    public static EndpointAddress[] DiscoverAddresses<T>(
        string serviceNamespace, string secret, Uri scope = null);

    public static Binding DiscoverBinding<T>(
        string serviceNamespace, string secret, Uri scope = null);
}
```

`DiscoverAddress<T>` discovers a service with a cardinality of one, `DiscoverAddresses<T>` discovers all available service endpoints (cardinality of all) and `DiscoverBinding<T>` uses the service metadata endpoint to discover the endpoint binding.

Much the same way, I defined the class `ServiceBusDiscoveryFactory`:

```
public static class ServiceBusDiscoveryFactory
{
    public static T CreateChannel<T>(string serviceNamespace, string secret,
        Uri scope = null) where T : class;

    public static T[] CreateChannels<T>(string serviceNamespace, string secret,
        Uri scope = null) where T : class;
}
```

`CreateChannel<T>` assumes cardinality of one, and it uses the metadata endpoint to obtain the service's address and binding used to create the proxy. `CreateChannels<T>` creates proxies to all discovered services, using all discovered metadata endpoints.

Figure 4 Using `ServiceBusDiscoveryClient`

```
string serviceNamespace = "...";
string secret = "...";

ServiceBusDiscoveryClient discoveryClient =
    new ServiceBusDiscoveryClient(serviceNamespace, secret);

FindCriteria criteria = new FindCriteria(typeof(IMyContract));
FindResponse discovered = discoveryClient.Find(criteria);
discoveryClient.Close();

EndpointAddress address = discovered.Endpoints[0].Address;
Binding binding = new NetTcpRelayBinding();
ChannelFactory<IMyContract> factory =
    new ChannelFactory<IMyContract>(binding, address);
// A channel factory extension method to pass creds to behavior
factory.SetServiceBusCredentials(secret);

IMyContract proxy = factory.CreateChannel();
proxy.MyMethod();
(proxy as ICommunicationObject).Close();
```

Figure 5 Implementing ServiceBusDiscoveryClient (Partial)

```

public class ServiceBusDiscoveryClient : ClientBase<IServiceBusDiscovery>
{
    protected Uri ResponseAddress
    {get;private set;}

    public ServiceBusDiscoveryClient(string endpointName) : base(endpointName)
    {
        string serviceNamespace =
            ServiceBusHelper.ExtractNamespace(Endpoint.Address.Uri);
        ResponseAddress = ServiceBusEnvironment.CreateServiceUri(
            "sb",serviceNamespace,"DiscoveryResponses/"+Guid.NewGuid());
    }

    public FindResponse Find(FindCriteria criteria)
    {
        string contractName = criteria.ContractTypeNames[0].Name;
        string contractNamespace = criteria.ContractTypeNames[0].Namespace;

        FindResponse response = DiscoveryHelper.CreateFindResponse();

        ManualResetEvent handle = new ManualResetEvent(false);

        Action<Uri,Uri[]> addEndpoint = (address,scopes)=>
        {
            EndpointDiscoveryMetadata metadata = new EndpointDiscoveryMetadata();
            metadata.Address = new EndpointAddress(address);
            if(scopes != null)
            {
                foreach(Uri scope in scopes)
                {
                    metadata.Scopes.Add(scope);
                }
            }
            response.Endpoints.Add(metadata);

            if(response.Endpoints.Count >= criteria.MaxResults)
            {
                handle.Set();
            }
        };

        DiscoveryResponseCallback callback =
            new DiscoveryResponseCallback(addEndpoint);

        ServiceHost host = new ServiceHost(callback);

        host.AddServiceEndpoint(typeof(IServiceBusDiscoveryCallback),
            Endpoint.Binding,ResponseAddress.AbsoluteUri);

        host.Open();

        try
        {
            DiscoveryRequest(criteria.ContractTypeNames[0].Name,
                criteria.ContractTypeNames[0].Namespace,
                criteria.Scopes.ToArray(),ResponseAddress);

            handle.WaitOne(criteria.Duration);
        }
        catch
        {}
        finally
        {
            host.Abort();
        }
        return response;
    }

    void DiscoveryRequest(string contractName,string contractNamespace,
        Uri[] scopesToMatch,Uri replayAddress)
    {
        Channel.OnDiscoveryRequest(contractName,contractNamespace,
            scopesToMatch, replayAddress);
    }

    [ServiceBehavior(InstanceContextMode = InstanceContextMode.Single,
        UseSynchronizationContext = false)]
    class DiscoveryResponseCallback : IServiceBusDiscoveryCallback
    {
        readonly Action<Uri,Uri[]> Action;

        public DiscoveryResponseCallback(Action<Uri,Uri[]> action)
        {
            Action = action;
        }

        public void DiscoveryResponse(Uri address,string contractName,
            string contractNamespace,Uri[] scopes)
        {
            Action(address,scopes);
        }
    }

    public static class DiscoveryHelper
    {
        internal static FindResponse CreateFindResponse()
        {
            Type type = typeof(FindResponse);

            ConstructorInfo constructor =
                type.GetConstructors(BindingFlags.Instance|BindingFlags.NonPublic)[0];

            return constructor.Invoke(null) as FindResponse;
        }
        // More members
    }
}

```

Announcements

To support announcements, you can again use the events relay binding to substitute UDP multicast. First, I defined the *IServiceBusAnnouncements* announcement contract:

```

[ServiceContract]
public interface IServiceBusAnnouncements
{
    [OperationContract(IsOneWay = true)]
    void OnHello(Uri address, string contractName,
        string contractNamespace, Uri[] scopes);

    [OperationContract(IsOneWay = true)]
    void OnBye(Uri address, string contractName,
        string contractNamespace, Uri[] scopes);
}

```

As shown in **Figure 6**, this time, it's up to the clients to expose an event binding endpoint and monitor the announcements.

The services will announce their availability (over the one-way relay binding) providing their address (step 1 in **Figure 6**), and the clients will proceed to invoke them (step 2 in **Figure 6**).

Service-Side Announcements

My *DiscoveryRequestService* supports announcements:

```

public class DiscoverableServiceHost : ServiceHost,...
{
    public const string AnnouncementsPath = "AvailabilityAnnouncements";

    public Uri AnnouncementsAddress
    {get;set;}

    public NetOnewayRelayBinding AnnouncementsBinding
    {get;set;}

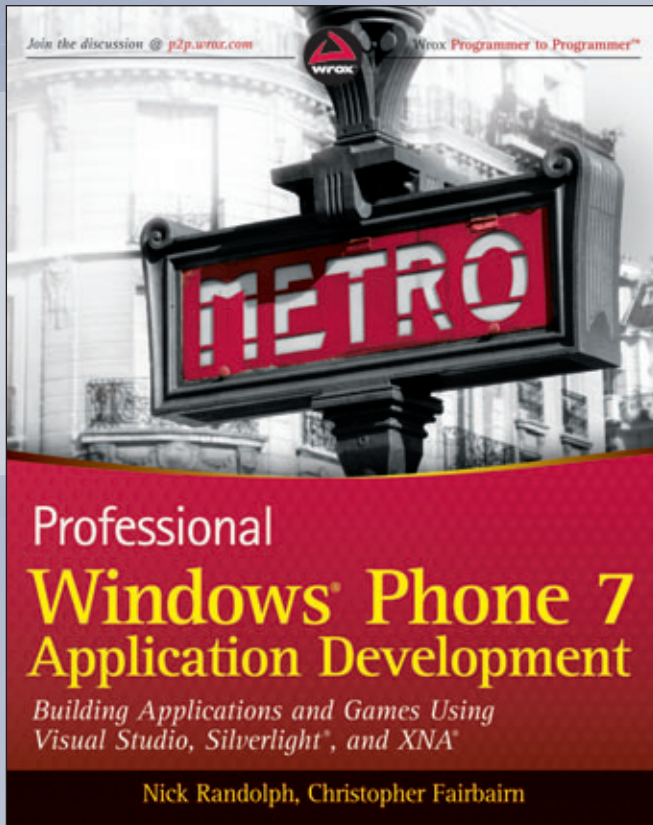
    // More members
}

```

However, on par with the built-in WCF announcements, by default it won't announce its availability. To enable announcements, you need to configure an announcement endpoint with the discovery behavior. In most cases, this is all you'll need to do. *DiscoveryRequestService* will fire its availability events on the "AvailabilityAnnouncements" URI under the service namespace.

Windows Phone 7

from the experts.



Everything you need to start developing apps for the Windows Phone 7 platform today.

Join us on facebook where you can connect with Wrox and learn more about the latest books on Windows Phone 7!

facebook.com/wroxpress



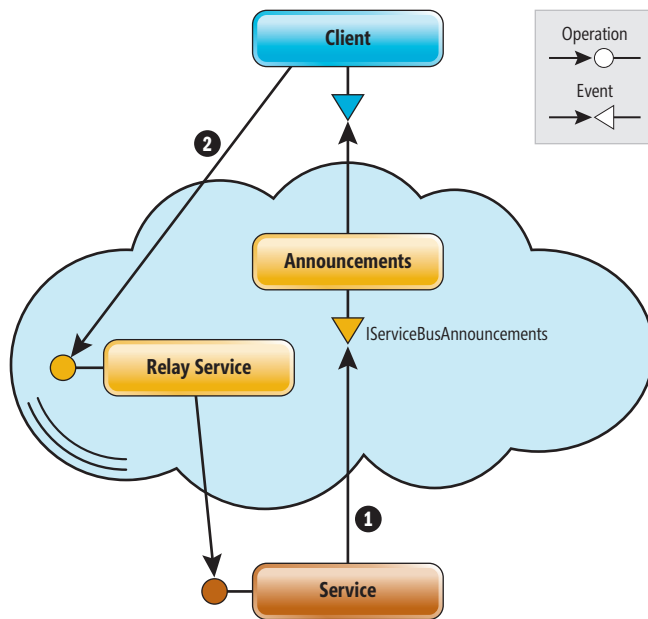


Figure 6 Availability Announcements over the Service Bus

You can change that default by setting the `AnnouncementsAddress` property before opening the host. The events will be fired by default using a plain one-way relay binding, but you can provide an alternative using the `AnnouncementsBinding` property before opening the host. `DiscoveryRequestService` will fire its availability events asynchronously to avoid blocking operations during opening and closing of the host. **Figure 7** shows the announcement-support elements of `DiscoveryRequestService`.

The `CreateAvailabilityAnnouncementsClient` helper method uses a channel factory to create a proxy to the `IServiceBusAnnouncements` announcements events endpoint. After opening and before closing `DiscoveryRequestService`, it fires the notifications. `DiscoveryRequestService` overrides both the `OnOpened` and `OnClosed` methods of `ServiceHost`. If the host is configured to announce, `OnOpened` and `OnClosed` call `CreateAvailabilityAnnouncementsClient` to create a proxy and pass it to the `PublishAvailabilityEvent` method to fire the event asynchronously. Because the act of firing the event is identical for both the hello and bye announcements, and the only difference is which method of `IServiceBusAnnouncements` to call, `PublishAvailabilityEvent` accepts a delegate for the target method. For each endpoint of `DiscoveryRequestService`,

Figure 7 Supporting Announcements with `DiscoveryRequestService`

```
public class DiscoverableServiceHost : ServiceHost,...
{
    Uri m_AnnouncementsAddress;

    bool IsAnnouncing
    {
        get
        {
            ServiceDiscoveryBehavior behavior =
                Description.Behaviors.Find<ServiceDiscoveryBehavior>();
            if (behavior != null)
            {
                return behavior.AnnouncementEndpoints.Any();
            }
            return false;
        }
    }

    public Uri AnnouncementsAddress
    {
        get
        {
            if (m_AnnouncementsAddress == null)
            {
                m_AnnouncementsAddress = ServiceBusEnvironment.
                    CreateServiceUri("sb", Namespace, AnnouncementsPath);
            }
            return m_AnnouncementsAddress;
        }
        set
        {
            m_AnnouncementsAddress = value;
        }
    }

    IServiceBusAnnouncements CreateAvailabilityAnnouncementsClient()
    {
        ChannelFactory<IServiceBusAnnouncements> factory =
            new ChannelFactory<IServiceBusAnnouncements>(
                AnnouncementsBinding, new EndpointAddress(AnnouncementsAddress));

        return factory.CreateChannel();
    }

    protected override void OnOpened()
    {
        base.OnOpened();

        if (IsAnnouncing)
        {
            IServiceBusAnnouncements proxy =
                CreateAvailabilityAnnouncementsClient();
            PublishAvailabilityEvent(proxy.OnHello);
        }
    }

    protected override void OnClosed()
    {
        if (IsAnnouncing)
        {
            IServiceBusAnnouncements proxy =
                CreateAvailabilityAnnouncementsClient();
            PublishAvailabilityEvent(proxy.OnBye);
        }
        ...
    }

    void PublishAvailabilityEvent(Action<Uri, string, string, Uri[]> notification)
    {
        foreach (ServiceEndpoint endpoint in Description.Endpoints)
        {
            if (endpoint is DiscoveryEndpoint || endpoint is
                ServiceMetadataEndpoint)
            {
                continue;
            }
            Uri[] scopes = LookupScopes(endpoint);

            WaitCallback fire = delegate
            {
                try
                {
                    notification(endpoint.Address.Uri, endpoint.Contract.Name,
                        endpoint.Contract.Namespace, scopes);
                    (notification.Target as ICommunicationObject).Close();
                }
                catch
                {
                }
            };
            ThreadPool.QueueUserWorkItem(fire);
        }
    }
}
```


patterns & practices

Symposium



Redmond
October 18-22, 2010

NET ADO.NET Ajax ASP.NET BizTalk Client Development Cloud
Dependency Injection Enterprise Library Enterprise Service Bus MVC p&p
Prism Rich Client RIA SOA Server Development Services
Development SharePoint SilverLight Software Factory Solution
Development Fundamentals SQL Server Unity VisualBasic Visual
studio WPF Web Application Web Services Windows Application
Windows XML

Highlights

Keynote Sessions by Senior Microsoft Executives and Technical Industry Leaders including Jason Zander, Robert C. Martin, Yousef Khalidi, and Charlie Kindel

3 Hands-on Developer Workshops covering Enterprise Library, Prism and Windows Azure

24 Thought-provoking sessions on Windows Phone 7, SharePoint, ASP.NET, Dependency Injection, Agile practices, and much more including patterns for how to apply these technologies in proven ways

Evening Networking Reception with entertainment, food and drinks

Symposium Party on Thursday Night at Lucky Strike Billiards in Bellevue

"Ask The Expert" Lunches and several Open Space sessions

Register using
'msdnmag'
and save \$150



For more info go to <http://cut.ms/Ygu>

patterns & practices

Figure 8 Receiving Announcements

```
class MyClient
{
    AddressesContainer<IMyContract> m_Addresses;

    public MyClient()
    {
        string serviceNamespace = "...";
        string secret = "...";

        m_Addresses = new ServiceBusAnnouncementSink<IMyContract>(
            serviceNamespace, secret);

        m_Addresses.Open();

        ...
    }
    public void OnCallService()
    {
        EndpointAddress address = m_Addresses[0];

        IMyContract proxy = ChannelFactory<IMyContract>.CreateChannel(
            new NetTcpRelayBinding(), address);
        proxy.MyMethod();
        (proxy as ICommunicationObject).Close();
    }
    ...
}
```

PublishAvailabilityEvent looks up the scopes associated with that endpoint and queues up the announcement to the Microsoft .NET Framework thread pool using a WaitCallback anonymous method. The anonymous method invokes the provided delegate and then closes the underlying target proxy.

The events will be fired by default using a plain one-way relay binding.

Receiving Announcements

I could have mimicked the WCF-provided AnnouncementService, as described in my January article, but there's a long list of things I've improved upon with my AnnouncementSink<T>, and I didn't see a case where you would prefer to use AnnouncementService in favor of AnnouncementSink<T>. I also wanted to leverage and reuse the behavior of AnnouncementSink<T> and its base class.

Figure 9 Implementing ServiceBusAnnouncementSink<T> (Partial)

```
[ServiceBehavior(UseSynchronizationContext = false,
    InstanceContextMode = InstanceContextMode.Single)]
public class ServiceBusAnnouncementSink<T> : AnnouncementSink<T>,
    IServiceBusAnnouncements
{
    Uri m_AnnouncementsAddress;

    readonly ServiceHost Host;
    readonly string ServiceNamespace;
    readonly string Owner;
    readonly string Secret;

    public ServiceBusAnnouncementSink(string serviceNamespace,
        string owner, string secret)
    {
        Host = new ServiceHost(this);
        ServiceNamespace = serviceNamespace;
        Owner = owner;
        Secret = secret;
    }

    public Uri AnnouncementsAddress
    {
        get
        {
            if(m_AnnouncementsAddress == null)
            {
                m_AnnouncementsAddress =
                    ServiceBusEnvironment.CreateServiceUri(
                        "sb", ServiceNamespace,
                        DiscoverableServiceHost.AnnouncementsPath);
            }
            return m_AnnouncementsAddress;
        }
        set
        {
            m_AnnouncementsAddress = value;
        }
    }

    public override void Open()
    {
        base.Open();

        Host.AddServiceEndpoint(typeof(IServiceBusAnnouncements),
            AnnouncementsBinding, AnnouncementsAddress.AbsoluteUri);
        Host.SetServiceBusCredentials(Owner, Secret);
        Host.Open();
    }

    public override void Close()
    {
        Host.Close();

        base.Close();
    }

    void IServiceBusAnnouncements.OnHello(Uri address, string contractName,
        string contractNamespace, Uri[] scopes)
    {
        AnnouncementEventArgs args =
            DiscoveryHelper.CreateAnnouncementArgs(
                address, contractName, contractNamespace, scopes);
        OnHello(this, args); //In the base class AnnouncementSink<T>
    }

    void IServiceBusAnnouncements.OnBye(Uri address, string contractName,
        string contractNamespace, Uri[] scopes)
    {
        AnnouncementEventArgs args = DiscoveryHelper.CreateAnnouncementArgs(
            address, contractName, contractNamespace, scopes);
        OnBye(this, args); //In the base class AnnouncementSink<T>
    }

    public static class DiscoveryHelper
    {
        static AnnouncementEventArgs CreateAnnouncementArgs(Uri address,
            string contractName, string contractNamespace, Uri[] scopes)
        {
            Type type = typeof(AnnouncementEventArgs);
            ConstructorInfo constructor =
                type.GetConstructors(BindingFlags.Instance | BindingFlags.NonPublic)[0];

            ContractDescription contract =
                new ContractDescription(contractName, contractNamespace);

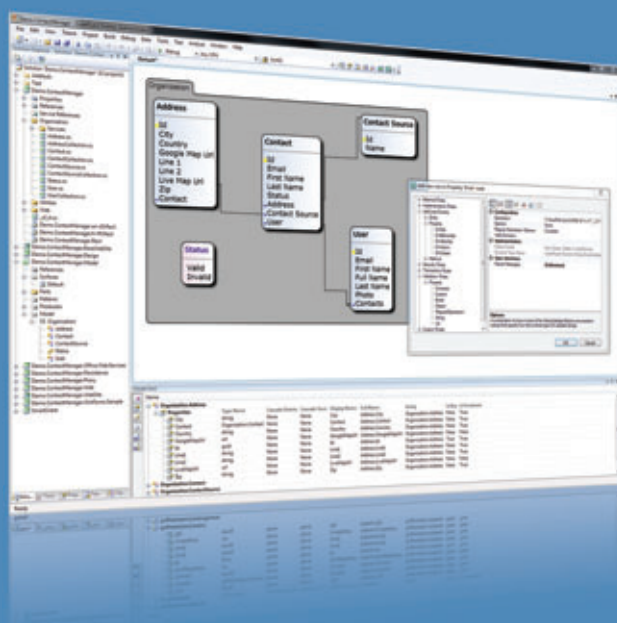
            ServiceEndpoint endpoint =
                new ServiceEndpoint(contract, null, new EndpointAddress(address));
            EndpointDiscoveryMetadata metadata =
                EndpointDiscoveryMetadata.FromServiceEndpoint(endpoint);

            return constructor.Invoke(
                new object[] { null, metadata }) as AnnouncementEventArgs;
        }
        // More members
    }
}
```



Need a **HAND?**

Use **CODEFLUENT ENTITIES!**



APPLICATION BLOCKS

- Localization
- Data Binding
- Rules and Validation
- Concurrency
- Security
- Caching
- Blob handling

SUPPORTED ARCHITECTURES

- SOA, SmartClient
- Rich Client, RIA, Silverlight,
- Web, Webparts
- Client/Server, N-Tier
- Office
- SharePoint
- SaaS, Cloud

FEATURED TECHNOLOGIES

- .NET (2 to 4), C#, Linq
- ASP .NET (WebForms, MVC)
- Silverlight (2 to 4)
- WPF, WinForms
- WCF, ASMX
- SQL Server (2000 to 2008)
- Oracle Database (9 to 11)
- Office (97 to 2010)
- SharePoint (2007 to 2010)
- Visual Studio (2005 to 2010)

CUSTOMER APPROVED MODEL-DRIVEN SOFTWARE FACTORY

We understand the challenges that come with today's and tomorrow's technology integration and evolution.

CodeFluent Entities is a fully integrated Model-Driven Software Factory which provides architects and developers a structured method and the corresponding tools to develop .NET applications, based on any type of architecture, from an ever changing business model and rules, at an unprecedented productivity level.

CodeFluent Entities is based on a pluggable producer logic, which, from a declarative model you designed, continuously generates ready-to-use, state-of-the-art, scalable, high-performance, and easily debuggable source code, components, and everything you need.

Download your **FREE** trial today at:

www.CodeFluentEntities.com/Msdn

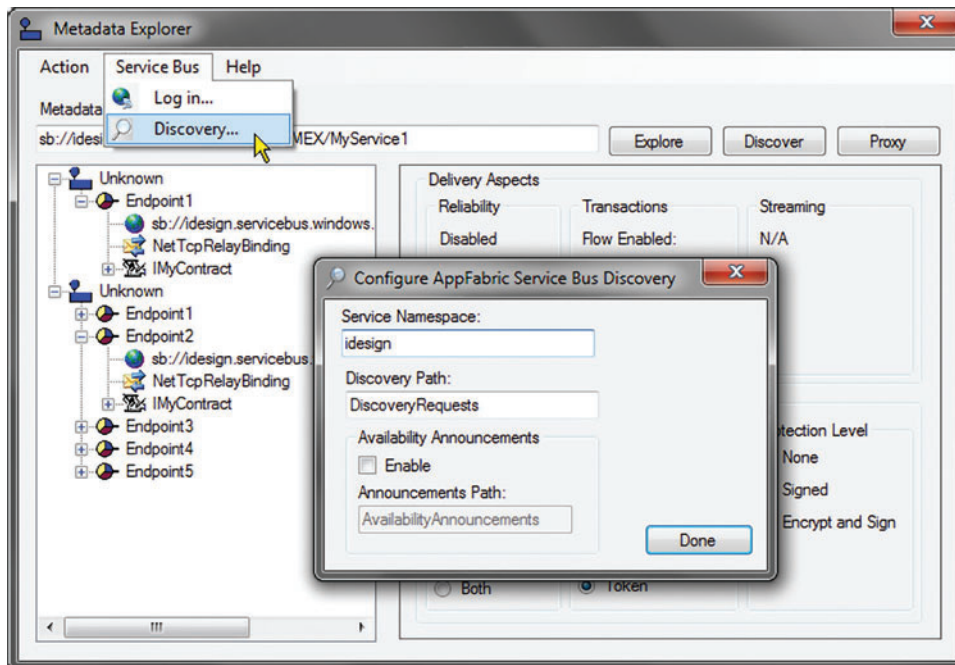


Figure 10 Configuring Discovery over the Service Bus

Therefore, for the client, I wrote `ServiceBusAnnouncementSink<T>`, defined as:

```
[ServiceBehavior(UseSynchronizationContext = false,
    InstanceContextMode = InstanceContextMode.Single)]
public class ServiceBusAnnouncementSink<T> : AnnouncementSink<T>,
    IServiceBusAnnouncements, where T : class
{
    public ServiceBusAnnouncementSink(string serviceNamespace, string secret);

    public ServiceBusAnnouncementSink(string serviceNamespace, string owner,
        string secret);
    public Uri AnnouncementsAddress {get;set;}

    public NetEventRelayBinding AnnouncementsBinding {get;set;}
}
```

The constructors of `ServiceBusAnnouncementSink<T>` require the service namespace.

`ServiceBusAnnouncementSink<T>` supports `IServiceBusAnnouncements` as a self-hosted singleton. `ServiceBusAnnouncementSink<T>` also publishes itself to the service bus registry. `ServiceBusAnnouncementSink<T>` subscribes by default to the availability announcements on the “AvailabilityAnnouncements” URI under the service namespace. You can change that (before opening it) by setting the `AnnouncementsAddress` property. `ServiceBusAnnouncementSink<T>` uses (by default) a plain `NetEventRelayBinding` to receive the notifications, but you can change that by setting the `AnnouncementsBinding` before opening `ServiceBusAnnouncementSink<T>`. The clients of `ServiceBusAnnouncementSink<T>` can subscribe to the delegates of `AnnouncementSink<T>` to receive the announcements, or they can just access the address in the base address container. For an example, see Figure 8.

Figure 9 shows the partial implementation of `ServiceBusAnnouncementSink<T>` without some of the error handling.

The constructor of `ServiceBusAnnouncementSink<T>` hosts itself as a singleton and saves the service namespace. When you open

`ServiceBusAnnouncementSink<T>`, it adds to its own host an endpoint supporting `IServiceBusAnnouncements`. The implementation of the event handling methods of `IServiceBusAnnouncements` creates an `AnnouncementEventArgs` instance, populating it with the announced service address, contract and scopes, and then calls the base class implementation of the respective announcement methods, as if it was called using regular WCF discovery. This both populates the base class of the `AddressesContainer<T>` and fires the appropriate events of `AnnouncementSink<T>`. Note that to create an instance of `AnnouncementEventArgs`, you must use reflection due to the lack of a public constructor.

The Metadata Explorer

Using my support for discovery for the service bus, I extended the discovery feature of the Metadata Explorer tool (presented in previous articles) to support the service bus. If you click the Discover button (see Figure 10), for every service namespace you have already provided credentials for, the Metadata Explorer will try to discover metadata exchange endpoints of discoverable services and display the discovered endpoints in the tree.

The Metadata Explorer will default to using the URI “DiscoveryRequests” under the service namespace. You can change that path by selecting Service Bus from the menu, then Discovery, to bring up the Configure AppFabric Service Bus Discovery dialog (see Figure 10).

For each service namespace of interest, the dialog lets you configure the desired relative path of the discovery events endpoint in the Discovery Path text box.

The Metadata Explorer also supports announcements of service bus metadata exchange endpoints. To enable receiving the availability notification, bring up the discovery configuration dialog box and check the Enable checkbox under Availability Announcements. The Metadata Explorer will default to using the “AvailabilityAnnouncements” URI under the specified service namespace, but you can configure for each service namespace any other desired path for the announcements endpoint.

The support in the Metadata Explorer for announcements makes it a simple, practical and useful service bus monitoring tool. ■

JUVAL LOWY is a software architect with IDesign providing .NET and architecture training and consulting. This article contains excerpts from his recent book, “Programming WCF Services 3rd Edition” (O’Reilly, 2010). He’s also the Microsoft regional director for the Silicon Valley. Contact Lowy at idesign.net.

THANKS to the following technical expert for reviewing this article:
Wade Wegner

GET THE FASTEST CONTROLS...



At Infragistics, we make sure our **NetAdvantage for .NET** controls make every part of your User Interface the very best it can be. That's why we've tested and re-tested to make sure our **Data Grids are the very fastest** grids on the market and our **Data Charts outperform** any you've ever experienced. Use our controls and not only will you get the fastest load times, but your apps will always look good too. Fast and good-looking...that's a killer app. Try them for yourself at infragistics.com/wow.

Infragistics
KILLER APPS. NO EXCUSES.

Infragistics Sales 800 231 8588
Infragistics Europe Sales +44 (0) 800 298 9055
Infragistics India +91-80-6785-1111
twitter.com/infragistics

Runtime Data Sharing Through an Enterprise Distributed Cache

Iqbal Khan

Many organizations use a combination of Microsoft .NET Framework and Java applications, especially midsize to large organizations that can't commit to only one technology for various reasons. Often, they employ Web applications, service-oriented architecture (SOA) Web services and other server applications that process lots of transactions.

Many of these applications need to share data with one another at run time. Often, they're all working on common business data that's stored in a database. They typically deal with continuous streams of data (for example, financial trading applications), and they need to process it and share results with other applications, again all at run time.

Although the database should be the master data store for permanent storage, it's not well-suited for runtime data sharing. One

reason for this is that performance isn't always great when reading data from the database. Furthermore, the database may not scale nicely in terms of handling transactions, so it may quickly become a bottleneck and slow down all the applications relying on it.

Moreover, you can't effectively share data in real time. Real-time data sharing requires that as soon as one application updates some data, all other applications interested in that data should be informed. Similarly, some applications may be waiting for certain types of data to be created and made available, and when this happens, they should be notified immediately.

These problems are common whether the applications needing to share data are all based on the .NET Framework or whether some are .NET and others Java. In fact, if the applications are a mix of .NET and Java, the problems are compounded because there's no automatic way for these applications to share data at the app-to-app level in a native fashion.

The Solution: Enterprise Distributed Cache

Luckily, the enterprise distributed cache can resolve these problems. This in-memory store spans multiple servers, pooling the memory of the servers so that memory storage capacity is scalable. Transaction capacity also becomes scalable, so the more servers you add, the greater the transaction load you can handle.

Enterprise distributed caches also provide event notification mechanisms, allowing applications to alert one another when they have updated data. Hence, you can have an asynchronous event notification mechanism where one application produces some data and others may consume it, creating a producer/consumer

This article discusses:

- How .NET and Java apps communicate via a cache
- Item-based event notifications
- App-generated custom event notifications
- Continuous query-based event notifications
- Read-through and write-through handlers
- Database synchronization
- High availability via a self-healing dynamic cluster
- Scalability via cache partitioning and replication

Technologies discussed:

Microsoft .NET Framework, Java

Figure 1 A .NET App Using an Enterprise Distributed Cache

```
using System;
...
using Alachisoft.NCache.Web.Caching;

namespace Client
{
    class Program
    {
        static string _sCacheName = "myAppCache";
        static Cache _sCache = NCache.InitializeCache(_sCacheName);

        static void Main(string[] args)
        {
            string employeeId = "1000";
            string key = "Employee:EmployeeId:" + employeeId;

            // First check the cache for this employee
            Employee emp = _sCache.Get(key);

            // If cache doesn't have it then make database call
            if (emp == null)
            {
                emp = LoadEmployeeFromDb(employeeId);

                // Now add it to the cache for next time
                _sCache.Insert(key, emp);
            }
        }
    }
}
```

model or publish/subscribe model. Multiple applications subscribe to certain types of data and are notified when it's published.

There's also a read-through/write-through mechanism, which means the enterprise distributed cache itself can read considerable data from the data source and the applications. Regardless of whether those applications are Java or .NET, their code becomes much simpler because they read the data from the enterprise distributed cache. They don't need to have all that database access

code built into them. See Figure 1 for a simple example of a .NET Framework app using an enterprise distributed cache.

Moreover, an enterprise distributed cache can synchronize itself with any data changes in the database made by other third-party applications. It has a connection with the database, allowing the database to notify it whenever a certain type of data changes in the database. Figure 2 illustrates how .NET and Java applications can share data with one another at run time through an enterprise distributed cache.

.NET and Java Apps Sharing Data

With an enterprise distributed cache, multiple applications—both .NET and Java—can access the same cache and share data through it. If it were only .NET applications (or only Java applications) sharing data through a distributed cache, they could store the objects in a native binary form and serialize/deserialize them. But when both types try to share data with each other, they need a portable data format in which to store the data in the distributed cache.

That's because when a .NET application stores an object in the distributed cache, it actually transforms the object into an XML document and stores that XML. On the other side, when a Java application reads that data from the distributed cache, it transforms the XML into a Java object. In effect, the XML is used as a portable data storage mechanism as a .NET object is transformed into XML and then from XML into Java and vice versa.

A number of open source libraries can help you transform your .NET or Java objects into XML and then back into the object form. You can develop your own, of course, but I recommend you pick an open source library. I personally like Web Objects in XML (WOX), developed by Carlos Jaimez and Simon Lucas

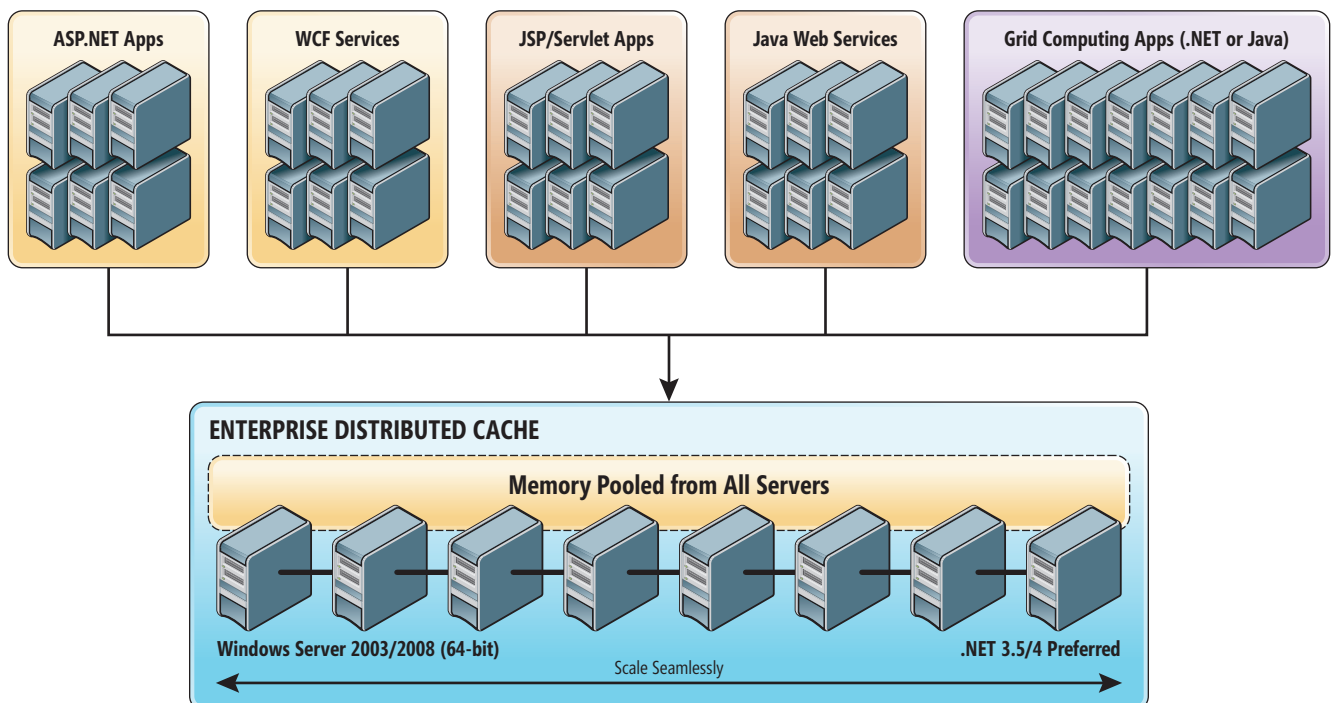


Figure 2 .NET and Java Apps Sharing Data Through a Distributed Cache

Figure 3 Student and Course Classes in Java and C#

```
// Java classes
public class Student
{
    private String name;
    private int registrationNumber;
    private Course[] courses;
}

public class Course
{
    private int code;
    private String name;
    private int term;
}

// *****
// .NET classes in C#
public class Student
{
    private String name;
    private Int32 registrationNumber;
    private Course[] courses;
}

public class Course
{
    private Int32 code;
    private String name;
    private Int32 term;
}
```

(woxserializer.sourceforge.net). I'll use examples of Java-to-.NET transformation from their Web site in this article (with their permission). **Figure 3** shows Student and Course classes defined both in Java and C#.

If we use .NET and Java apps to store these Student and Course objects in an enterprise distributed cache, we can then use the WOX library to transform them into XML. Then, when an application wants to read these objects from the enterprise distributed cache, it reads the WOX library again to transform the XML back into the Java or .NET object form. **Figure 4** shows both Student and Course classes transformed into XML.

Within your application, you should call WOX from your caching layer or the data access layer.

Figure 4 Java and .NET Classes Transformed into XML

```
<object type="Student" id="0">
  <field name="name" type="string" value="Carlos Jaimez"/>
  <field name="registrationNumber" type="int" value="76453"/>
  <field name="courses">
    <object type="array" elementType="Course" length="3" id="1">
      <object type="Course" id="2">
        <field name="code" type="int" value="6756"/>
        <field name="name" type="string" value="XML and Related Technologies"/>
        <field name="term" type="int" value="2"/>
      </object>
      <object type="Course" id="3">
        <field name="code" type="int" value="9865"/>
        <field name="name" type="string" value="Object Oriented Programming"/>
        <field name="term" type="int" value="2"/>
      </object>
      <object type="Course" id="4">
        <field name="code" type="int" value="1134"/>
        <field name="name" type="string" value="E-Commerce Programming"/>
        <field name="term" type="int" value="3"/>
      </object>
    </object>
  </field>
</object>
```

Item-Based Event Notifications

Event notifications are a powerful mechanism that allows multiple applications (both .NET and Java) to coordinate data sharing asynchronously. This mechanism helps avoid the expensive polling of the database that applications would have to do if they didn't have such a facility. It's shared among .NET and Java applications, so they can notify one another seamlessly.

When a .NET application stores an object in the distributed cache, it actually transforms the object into an XML document and stores that XML.

One common type of event notification is item-based notification. In this type, applications register interest in various cached item keys (that may or may not exist in the cache yet), and they're notified separately whenever that item is added, updated or removed from the distributed cache by anybody for any reason. For example, even if an item is removed due to expiration or eviction, the item-remove event notification is fired.

Both .NET and Java applications can register interest for the same cached items and be notified about them. The notification usually includes the affected cached item as well, which, as we saw in the previous section, is transformed into either .NET or Java, depending on the type of application.

App-Generated Custom Event Notifications

An enterprise distributed cache is also a powerful event propagation platform for both .NET and Java applications. Any applications connected to an enterprise distributed cache can fire custom events into the cache, and then all other applications that have registered interest in those custom events will be notified by the cache, regardless of where those applications are located. This by itself provides a powerful language- and platform-independent event propagation mechanism in an enterprise distributed cache.

This feature allows applications to collaborate in data sharing asynchronously. For example, if one application puts some data in the distributed cache, it can then fire a custom event so other applications that are supposed to consume or process this data further are notified immediately.

Continuous Query-Based Event Notifications

Item-based event notification is powerful but requires the application to know the key of the cached item. If you combine item-based event notification with other grouping features commonly provided in an enterprise distributed cache (such as tags, groups/subgroups and more), you can pretty much handle most

Celebrating 100,000 Users

World's Largest File Format Community

- . Create Files
- . Print Files
- . Importing
- . Modify Files
- . Merge Files
- . Exporting
- . Save Files
- . Convert Files
- . Reporting

100% Standalone - No Automation

Serving more than 50% of the Fortune 100 Companies!



Aspose provides extensive file format processing capabilities for the most popular formats including:

**DOCX PDF PPT ODF Report SWF InfoPath
XLSX BarCode MPP(Project) MSG(Outlook) ++**

The TOTAL Solutions for .NET, Java, SQL Server Rendering Extensions, SharePoint, and JasperReports Exporters.

Get your **FREE** evaluation copy at www.aspose.com

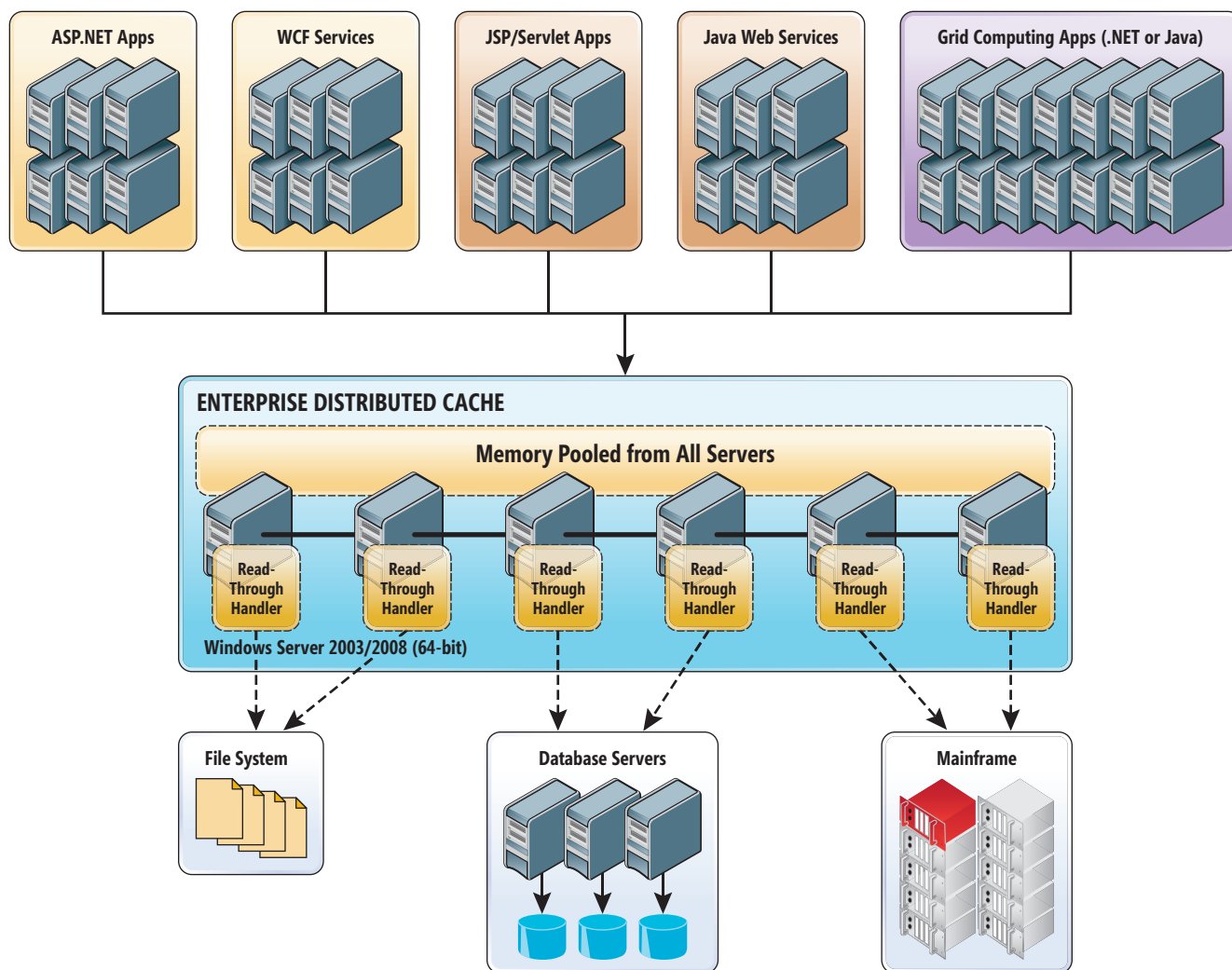


Figure 5 How Read-Through/Write-Through Is Used

of the cases where applications need to be notified based on what happens to various cached items.

However, there are two limitations with item-based events. First, as noted, the application has to know all the keys of cached items about which it wants to be notified. Second, it will be notified no matter what change is made to these items. The application can't put more detailed criteria in place, so it's notified only when specific changes are made to the data.

To handle such cases, an enterprise distributed cache provides a continuous query—a SQL-like query that captures an application's business rules about data in which it's interested. A continuous query isn't a search query but rather a "criteria" the enterprise distributed cache keeps; anytime something is added or updated in the distributed cache, the operation is compared to the continuous query criteria. If the criteria match, an event is fired and the application issuing the continuous query criteria is notified.

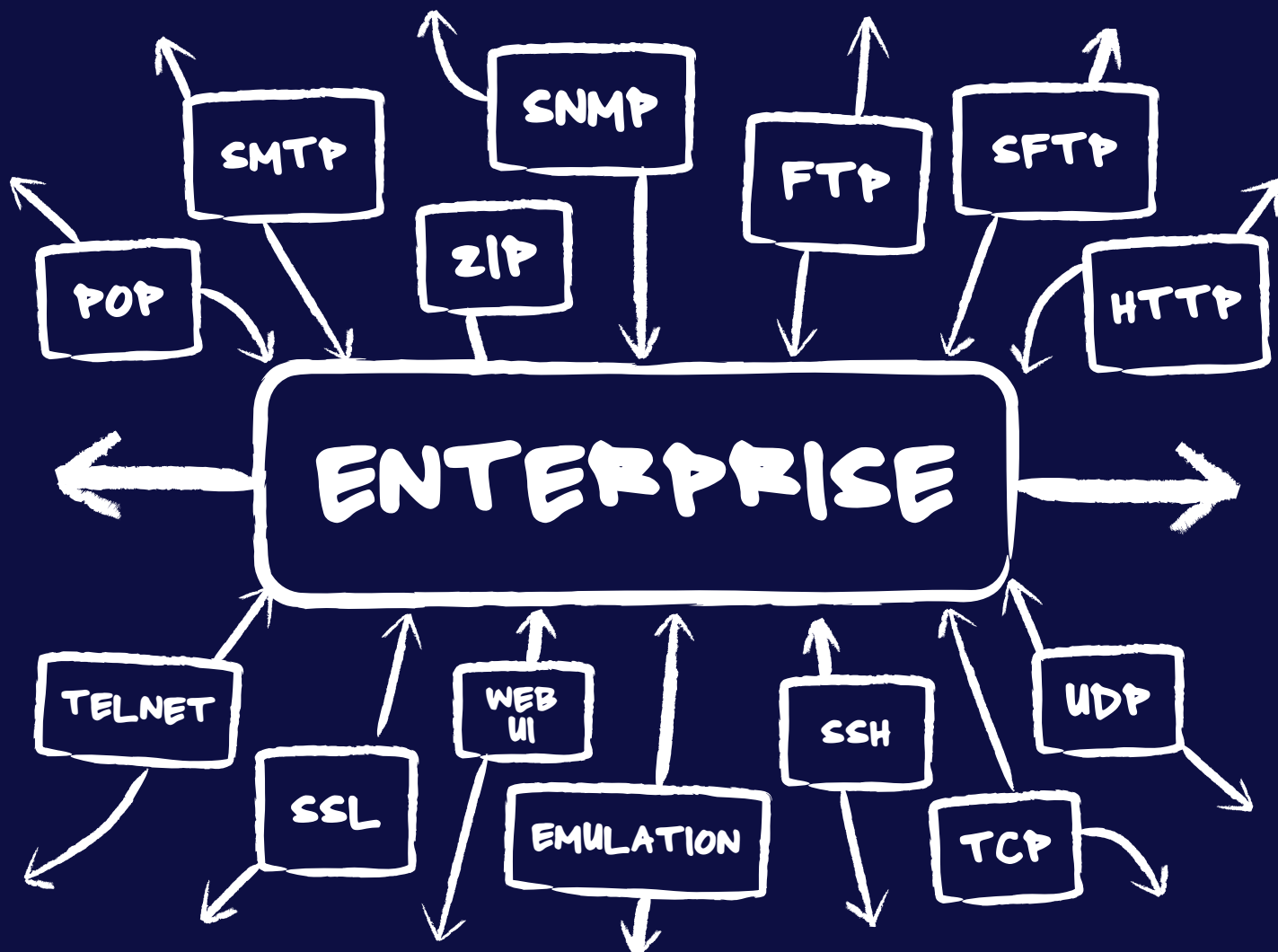
Continuous query allows applications to watch for more complex and sophisticated changes and be notified only when those changes happen.

Read-Through and Write-Through Handlers

Many times, applications try to read data that doesn't exist in the enterprise distributed cache and must be read from a database. In these situations, applications could go directly to the database and read that data, but that would mean that all applications end up having the same data access code duplicated (especially in both .NET and Java). Or, they can ask the enterprise distributed cache to read the data from the database for them when they need it.

The read-through/write-through feature allows an enterprise distributed cache to read data directly from the data source. The applications can simplify their code so they don't have to go to the database. They can just ask the enterprise distributed cache to give them the data, and if the cache doesn't have the data, it will go and read it from the data source. **Figure 5** shows how read-through and write-through fit into an enterprise distributed cache.

I want to mention one point of caution here. Although there's great benefit in having the distributed cache read the data from the database for you, many types of data are best read directly from the database by the application. If you're reading collections of data that involve complex joins, it's best to read it yourself and then put it in the distributed cache.



Internet Connectivity for the Enterprise

Since 1994, Dart has been a leading provider of high quality, high performance Internet connectivity components supporting a wide range of protocols and platforms. Dart's three product lines offer a comprehensive set of tools for the professional software developer.



PowerSNMP for ActiveX and .NET

Create custom Manager, Agent and Trap applications with a set of native ActiveX, .NET and Compact Framework components. **SNMPv1, SNMPv2, SNMPv3** (authentication/encryption) and **ASN.1** standards supported.

PowerWEB for ASP.NET

AJAX enhanced user interface controls for responsive ASP.NET applications. Develop unique solutions by including streaming file upload and interactive image pan/zoom functionality within a page.

PowerTCP for ActiveX and .NET

Add high performance Internet connectivity to your ActiveX, .NET and Compact Framework projects. Reduce integration costs with detailed documentation, hundreds of samples and an expert in-house support staff.

SSH
UDP
TCP
SSL

FTP
SFTP
HTTP
POP

SMTP
IMAP
S/MIME
Ping

DNS
Rlogin
Rsh
Rexec

Telnet
VT Emulation
ZIP Compression
more...

Ask us about Mono Platform support. Contact sales@dart.com.

Download a fully functional product trial today!

 **DART.com**

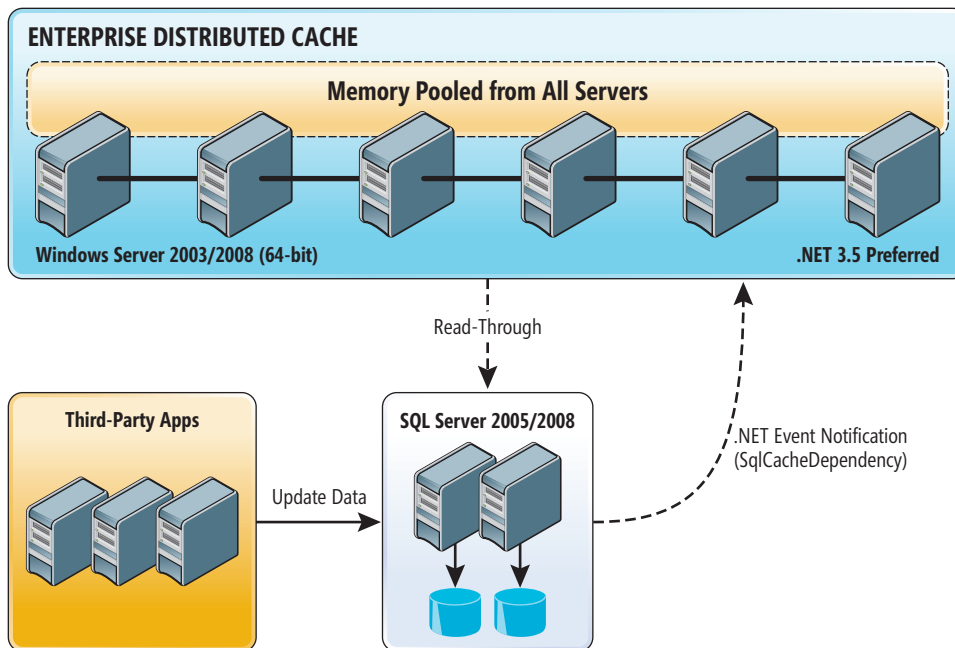


Figure 6 Database Synchronization in Distributed Cache

Database Synchronization

Because a lot of data is being put in the enterprise distributed cache, it only makes sense to make sure this data is kept synchronized with the master data source, usually a relational database. An enterprise distributed cache provides such a feature.

This database synchronization feature allows applications to specify a relationship (a dependency) between cached items and rows in database tables. Whenever data in the database changes, the database server fires a .NET event if it's a SQL Server 2005/2008 database and notifies the enterprise distributed cache of such a change. For other databases that don't support .NET events, an enterprise distributed cache also provides configurable polling, where it can poll the database (say, once every 15 seconds) and synchronize if data has changed there.

The distributed cache then either removes that data from the cache or reads a fresh copy of it if you've configured the read-through feature. **Figure 6** shows how an enterprise distributed cache synchronizes with SQL Server.

High Availability: Self-Healing Dynamic Cluster

An enterprise distributed cache is used as a runtime data sharing platform among multiple applications (.NET to .NET, .NET to Java, and Java to Java). In many cases, these applications are mission-critical for your business.

This means an enterprise distributed cache must be highly available, because so many

mission-critical applications depend on it. The enterprise distributed cache can't go down or stop working, and it should require virtually no downtime for maintenance or other normal operations.

An enterprise distributed cache achieves high availability by having a self-healing, dynamic cluster of cache servers. Self-healing here means the cluster is aware of all its members and adjusts dynamically if a member leaves or joins. It also ensures that data is replicated for reliability, and if a cluster member leaves, its backup data is made available to the applications automatically. All of this must be done quickly and without causing any interruptions in the applications using the enterprise distributed cache.

Scalability: Cache Partitioning and Replication

Many applications using an enterprise distributed cache are high-transaction apps. Therefore, the load on the cache cluster can grow rapidly; however, if the response time of the enterprise distributed cache drops, it loses its value. In fact, this is an area where an enterprise distributed cache is superior to relational databases; it can handle a lot more transactions per second because it can keep

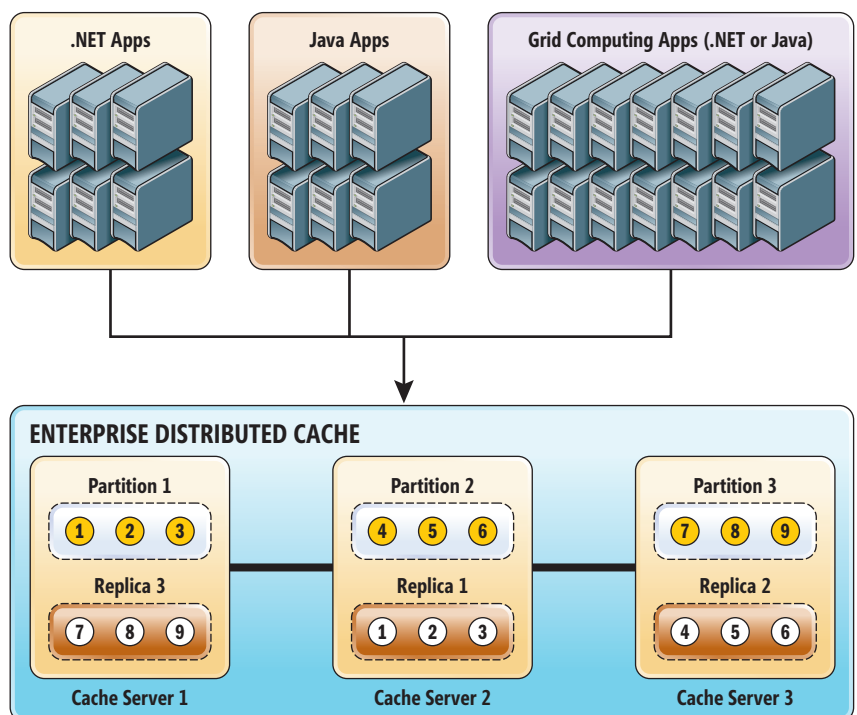


Figure 7 Partitioned-Replication Topology for Reliable Scalability

adding more servers to the dynamic cluster. But scalability can't be achieved unless data in the distributed cache is stored intelligently. This is accomplished through data partitioning, with each partition replicated for reliability.

Thanks to the enterprise distributed cache, you're allowed to exploit the benefits of a partitioned topology for scalability. **Figure 7** shows a partitioned-replication topology.

An enterprise distributed cache automatically partitions all data you're storing in the cache. Every partition is stored on a different server, and a backup for this partition is created and stored on yet another server. This ensures that if any server goes down, no data is lost.

An enterprise distributed cache is extremely fast and scalable by design.

So, to summarize, partitioning allows you to keep adding more cache servers to the dynamic cluster to grow storage capacity, and it also increases the transaction-per-second capacity as you add more servers. And replication ensures reliability of data because no data loss occurs if a server goes down.

Capable and Collaborative

Wrapping up, an enterprise distributed cache is an ideal way for high-transaction .NET and Java applications to share data with other apps. It ensures that data sharing is done in real time because of its powerful event propagation mechanisms, including item-based event notification, application-generated custom event notifications and continuous query-based event notifications.

An enterprise distributed cache is extremely fast and scalable by design. It's fast because it's in-memory. It's scalable because it can grow into multiple servers. It partitions the actual storage and stores each partition on a different server, and it stores a backup of that partition onto yet another server, like a RAID disk.

Today's applications need to be much more capable than in the past. They need to work in a more collaborative fashion to share data and interact with one another. They need to be exceedingly fast while meeting the needs of extremely high loads to avoid compromising performance and scalability. Moreover, they must perform across different platforms so

.NET applications can transparently and effectively work with Java apps. An enterprise distributed cache helps meet all these goals. ■

IQBAL KHAN is the president and technology evangelist of Alachisoft (alachisoft.com), which provides NCache, a .NET-distributed cache for boosting performance and scalability in enterprise applications. Khan received a master's degree in computer science from Indiana University in 1990. You can reach him at iqbal@alachisoft.com.

THANKS to the following technical expert for reviewing this article:
Stefan Schackow



The banner features the Microsoft Dynamics AX logo at the top left, followed by the text "Microsoft Dynamics AX" and "TECHNICAL CONFERENCE 2011" in large, bold, white letters on a blue background. Below the title is a photograph of two men in a meeting, one pointing at a screen. Under the photo is the tagline "Learn | Inspire | Innovate". The dates "January 17 - 20, 2011 • Redmond, WA" are prominently displayed. Below this, a paragraph describes the event, and another paragraph lists the products included. Logos for Office, SharePoint Server 2010, Visual Studio 2010, Windows Server 2008 R2, SQL Server 2008 R2, and Microsoft .NET are shown in a grid. The bottom section has a dark blue background with the text "Experience How. Get Involved." and a URL. The Microsoft logo is in the bottom right corner.

Microsoft Dynamics AX

TECHNICAL CONFERENCE 2011

Learn | Inspire | Innovate

January 17 - 20, 2011 • Redmond, WA

The Microsoft Dynamics AX team has been hard at work developing our most innovative release to date. We are inviting our partners and customers to Redmond to experience the product pre-release.

Microsoft Dynamics AX provides industry solutions for customers by bringing together a breadth of Microsoft products and technologies, including:

- Office
- SharePoint Server 2010
- Visual Studio 2010
- Windows Server 2008 R2
- SQL Server 2008 R2
- Microsoft .NET

Experience How. Get Involved.

Find out more at: <http://www.microsoft.com/dynamics/daxconf2011/MSDN>

Want to get involved? Email: daxconf@microsoft.com.

Microsoft

Building a Real-Time Transit Application Using the Bing Map App SDK

Luan Nguyen

During the **Tech•Ed 2010 conference** in June, Microsoft announced the availability of the free Bing Map App SDK, which enables developers to write map-centric applications on top of the Bing Maps explore site located at bing.com/maps/explore/.

This presents ample opportunities for organizations, businesses or hobbyists to create their own mapping experiences within Bing Maps. Businesses can write apps to advertise their products or to complement their online services. For example, bestparking.com developed a “Parking finder” app, which helps you find all parking facilities in your city using their database.

This article discusses:

- Downloading the Bing Map App SDK
- The One Bus Away API
- Defining a map entity
- Writing a plug-in
- Using a layer to show content and pushpins
- Testing, debugging and submitting a map app

Technologies discussed:

The Bing Map App SDK

Code download available at:

code.msdn.microsoft.com/mag201010BingMap

To see what a map app looks like, go to the Bing Maps explore site and click on the MAP APPS button located near the bottom left corner of the page. The Map apps gallery will open, showing the available apps. There are already many apps featured in the gallery, and it continues to grow.

In this article, I'll describe what it takes to write a map app using the Bing Map App SDK. I'll walk you through the development of a sample app that shows real-time bus arrival information around King County in Washington state. This app is inspired by the popular One Bus Away app.

The Final Product

Before I delve into code, let's take a look at what the final map app will look like. When activated, if the map is zoomed into the King County area, my map app will show in the left panel the list of all bus stops within the map viewport (see **Figure 1**).

Each bus stop has information about all buses serving that stop. Clicking on the “Arrival times” hyperlink for each stop will bring up the second panel, which displays upcoming arrival times (see **Figure 2**).

The app also places a visually appealing pushpin (marked with the letter B) at every bus stop on the map. You can interact with the pushpins to activate a pop-up, from which you can invoke common commands for a particular bus stop, such as getting driving or walking directions to or from it (see **Figure 3**).

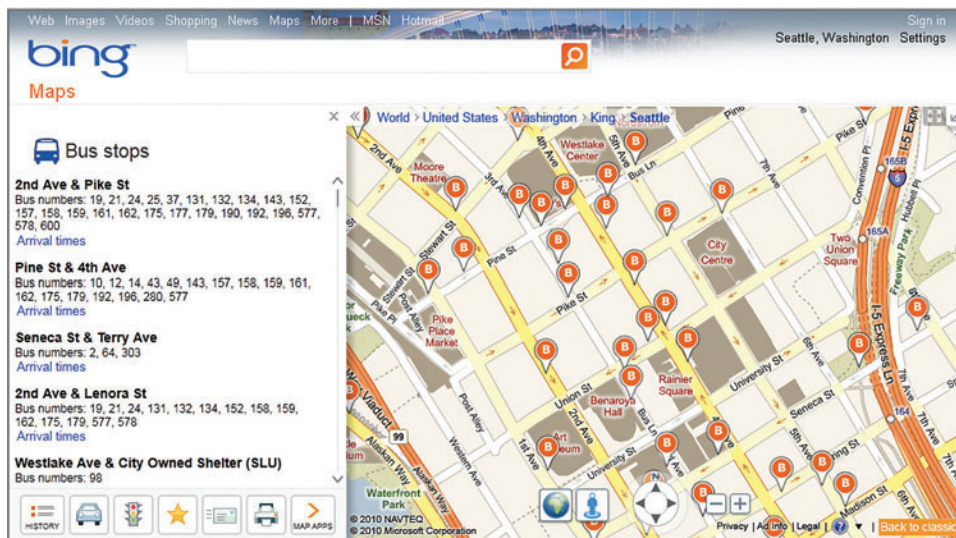


Figure 1 The OBA App Shows Bus Information on the Left Panel and Pushpins on the Map

Download the Bing Map App SDK

Before starting to write your first map app, you need to install the Bing Map App SDK at connect.microsoft.com/bingmapapps. The SDK provides the reference assemblies, one sample project and offline documentation. In addition, because the SDK is based on Silverlight 4, you also need to install the Silverlight 4 Tools for Visual Studio 2010, available at silverlight.net/getstarted/.

Map App Basics

The Bing Maps explore site was built with extensibility as one of the top priorities. The Bing Maps team wanted to enable developers around the world to easily add functionality that they would find useful and that would be potentially useful to others. In that sense, a *map app* is a concrete feature that makes use of the building block services provided by the core of Bing Maps. If you look at the default Bing Maps site, the driving directions, business search and location search features are all built from the same building blocks.

With that goal in mind, the Bing Maps team decided to build an extensible framework to support the concept of map apps. When written on top of the framework, map apps take advantage of a dependency injection-style approach to provide access to functionality, similar to that found in the Managed Extensibility Framework (MEF).

At the core of the framework is the concept of plug-ins. A plug-in is a unit of extensibility that allows developers to add features in a maintainable, decoupled way. It can be thought of as the equivalent of a composable part in MEF terminology. Through attributes, a plug-

in declares the imports it needs as well as exports it wants to share. At run time, the framework will resolve dependencies among plug-ins and match the imports with the exports of the same contract.

To write a map app, you create a Silverlight 4 class library that contains exactly one subclass of the base *Plugin* class. In order to do anything useful, your plug-in will likely import a number of built-in contracts that allow you to access various core services. The SDK documentation has a section listing all of the built-in contracts exported by the framework.

To reference the *Plugin* class and all built-in contract types, your

project will need to reference the provided assemblies from the SDK. Figure 4 briefly describes these four assemblies.

Access the One Bus Away API

To obtain real-time bus arrival information, I'll be making use of the publicly available One Bus Away REST API (simply referred to as Oba API for the rest of this article). You'll find details about the Oba API at code.google.com/p/onebusaway/wiki/OneBusAwayRestApi. (Note that the API is currently free for noncommercial use, but you're required to register for an application key before you can access it. In the downloadable code for this article, I've removed the key assigned to me, so if you want to compile and try the app yourself, you need to replace it with your own key.)

When a Silverlight application tries to access a Web service on a different domain, the Silverlight runtime mandates that the service must explicitly opt in to allow cross-domain access. A service indicates its consent by placing either a *clientaccesspolicy.xml* or *crossdomain.xml* file at the root of the domain where the service is hosted. More details regarding the schemas of these two files are documented at msdn.microsoft.com/library/cc197955%28VS.95%29. Luckily, the Oba API provides the *crossdomain.xml* file, which allows my map app to call it in Silverlight code.

In the downloadable solution, you'll see two library projects, *ObaApp* and *ObaLib*. *ObaApp* is the main project, which contains the map app plug-in, and it references

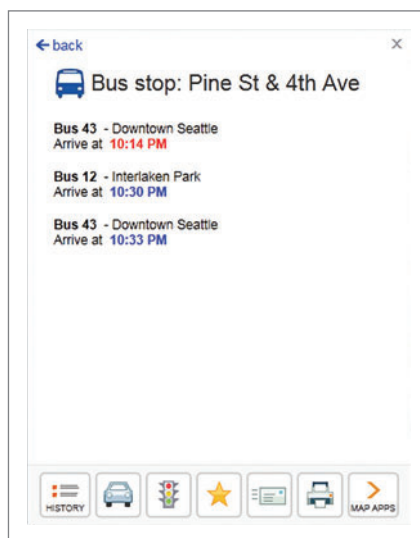


Figure 2 Bus Arrival Times for a Particular Bus Stop

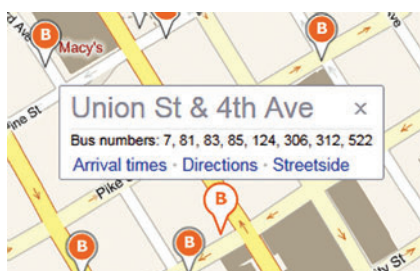


Figure 3 Clicking on a Pushpin Shows a Pop-Up UI with Extra Information

Figure 4 Reference Assemblies for a Map App Project

Assembly Name	Description
Microsoft.Maps.Plugins.dll	Contains the base Plugin class and related Import/Export attribute classes.
Microsoft.Maps.Core.dll	Contains all the contract types that Bing Maps provides.
Microsoft.Maps.MapControl.Types.dll	Contains the types required for working with the map control.
Microsoft.Maps.Extended.dll	Contains the types to interact with the StreetSide map mode.

ObaLib. ObaLib is another class library project that encapsulates all helper classes to communicate with the Oba API. I made it a separate library so I can easily reuse it in different projects if I need to. I won't go into the details of every class in this library, but you're encouraged to examine the source code to learn more about the classes in there.

The most important class to know is the ObaSearchHelper class, which provides convenient methods and events to make queries to the Oba API:

```
public sealed class ObaSearchHelper
{
    public void SearchArrivalTimesForStop(string stopId);
    public event EventHandler<BusTripsEventArgs>
        SearchArrivalTimesForStopCompleted;

    public void SearchStopsByLocation(double latitude, double longitude,
        double radius, int maxResultCount);
    public event EventHandler<BusStopsEventArgs>
        SearchStopsByLocationCompleted;

    // more method/event pairs
    // ...
}
```

It's easy to spot the pattern used in this class. For every REST endpoint of the Oba API, there's one method to trigger the search and one corresponding event to signal the completion of that

search. Subscribers of the events can obtain the results from event argument objects.

With this handy class ready, let's dissect the main classes in the ObaApp project.

You can interact with the pushpins to activate a pop-up, from which you can invoke common commands for a particular bus stop, such as getting driving or walking directions to or from it.

Figure 5 The BusStopEntity Class

```
public class BusStopEntity : Entity
{
    private readonly DependencyProperty NamePropertyKey =
        Entity.RetrieveProperty("Name", typeof(string));

    private BusStop _busStop;

    // Short name of this bus stop
    public string Name
    {
        get { return _busStop.Name; }
    }

    // Unique id of this bus stop
    public string StopId
    {
        get { return _busStop.Id; }
    }

    public BusStopEntity(BusStop busStop,
        PushpinFactoryContract pushpinFactory)
    {
        _busStop = busStop;

        // Set the location of this Entity
        Location location =
            new Location(_busStop.Latitude, _busStop.Longitude);
        this.Primitive = pushpinFactory.CreateStandardPushpin(location, "B");

        // Set the name of this Entity
        this.SetValue(NamePropertyKey, _busStop.Name);
    }
}
```

Define the Map Entity

The first thing you need to do when writing a map app is to define your map entity. Here's the definition of entity from the SDK documentation: "An Entity is a geo-associated item on the map. A map entity can be a point, a polyline, a polygon or an image overlay." The simple rule of thumb is that if you want to place *something* on the map, you need an instance of Entity to represent it. Usually, you want to create a subclass of Entity to add extra properties and custom logic for your entities. In my app, because I want to show the locations of bus stops, I wrote a BusStopEntity class to represent a bus stop. **Figure 5** shows the BusStopEntity class.

My BusStopEntity class exposes two properties, Name and StopId, which I'll later use to data-bind to UI controls. These values come from the underlying BusStop instance. Defined in the ObaLib project, the BusStop class encapsulates data of a bus stop, which it gets from the Oba API.

In the constructor, I also set the Primitive and the Name properties. The Primitive property represents the type (for example, point, polygon or polyline) and location of my entity. I set the location to the longitude and latitude values of the BusStop instance. In addition, setting the Primitive property to the returned value of PushpinFactoryContract.CreateStandardPushpin gives my entity the look and feel of the standard Bing Maps pushpins. The PushpinFactoryContract type provides handy methods for creating common pushpin UI elements. You don't have to use it if you want to create your own custom pushpin shapes. Here I just simply put a letter B (for bus) inside the pushpin, but you can use an image or any UIElement.

INSPIRE! EMPOWER! IMPRESS!



You've got the data, but time, budget and staff constraints can make it hard to present that valuable information in a way that will impress. With Infragistics' **NetAdvantage for Silverlight Data Visualization** and **NetAdvantage for WPF Data Visualization**, you can create Web-based data visualizations and dashboard-driven applications on Microsoft Silverlight and WPF that will not only impress decision makers, it actually empowers them. Go to infragistics.com/sldv today and get inspired to create killer apps.

Infragistics
KILLER APPS. NO EXCUSES.

Infragistics Sales 800 231 8588
Infragistics Europe Sales +44 (0) 800 298 9055
Infragistics India +91-80-6785-1111
twitter.com/infragistics

Figure 6 Imports Declared in ObaPlugin Class

```
public class ObaPlugin : Plugin
{
    #region Imports

    [ImportSingle("Microsoft/LayerManagerContract", ImportLoadPolicy.Synchronous)]
    public LayerManagerContract LayerManager { get; set; }

    [ImportSingle("Microsoft/PopupContract", ImportLoadPolicy.Synchronous)]
    public PopupContract PopupContract { get; set; }

    [ImportSingle("Microsoft/ConfigurationContract",
        ImportLoadPolicy.Synchronous)]
    public ConfigurationContract ConfigurationContract { get; set; }

    [ImportSingle("Microsoft/MapContract", ImportLoadPolicy.Synchronous)]
    public MapContract Map { get; set; }

    [ImportSingle("Microsoft/PushpinFactoryContract",
        ImportLoadPolicy.Synchronous)]
    public PushpinFactoryContract PushpinFactory { get; set; }

    [ImportSingle("Microsoft/ContributorHelperContract",
        ImportLoadPolicy.Synchronous)]
    public ContributorHelperContract ContributorHelper { get; set; }

    #endregion
}
```

Although I define the Name property in my subclass, it won't be recognized by other types in the SDK—those types simply have no knowledge of my property. Therefore, I also set the same value to the Name dependency property, which I retrieve via the call `Entity.RetrieveProperty("Name", typeof(string))`. By doing this, I enable other features to retrieve the name of a bus stop entity.

Write the Main Plug-in

As explained earlier, you need a Plugin-derived class to represent your map app. **Figure 6** shows mine, which is aptly named `ObaPlugin`. My plug-in imports a total of six contracts. Notice that all contracts provided by Bing Maps follow the naming convention of `Microsoft/*`. The SDK documentation provides detailed descriptions of all contracts that can be imported, shown in **Figure 7**.

Figure 7 Contracts Imported by the ObaPlugin

Contract Type	Description
LayerManagerContract	Allows a plug-in to add or remove map layers.
PopupContract	Allows a plug-in to show the pop-up UI when user hovers or clicks on the entities/pushpins.
ConfigurationContract	Allows a plug-in to dynamically load configuration values from the configuration file at run time.
MapContract	Allows a plug-in to control the map.
PushpinFactoryContract	Allows a plug-in to render standard pushpins for its entities.
ContributorHelperContract	Allows a plug-in to create asynchronous or demand-load contributors. (I'll use this helper to add "Driving directions" and "StreetSide" contributor links to my pop-up later.)

After declaring the imports, I override the virtual `Initialize` method:

```
public override void Initialize()
{
    // Obtain the application key from configuration file
    IDictionary<string, object> configs =
        ConfigurationContract.GetDictionary(this.Token);
    string appKey = (string)configs["ApplicationKey"];
    _searchHelper = new ObaSearchHelper(appKey);
    _searchHelper.SearchStopsByLocationCompleted +=
        OnSearchBusStopsCompleted;
    _searchHelper.SearchArrivalTimesForStopCompleted +=
        OnSearchArrivalTimesComplete;

    // Update the bus stop every time map view changes
    Map.TargetViewChangedThrottled += (s, e) => RefreshBusStops();
}
```

The Bing Maps explore site was built with extensibility being one of the top priorities.

This method is called exactly once after the plug-in instance is constructed and all the declared imports are satisfied. This is the perfect place to do any initialization work that depends on the imported contracts. If your plug-in has any exports, you also have to make sure all the exported properties are fully instantiated by the time `Initialize` returns.

I first obtain the application key from the configuration file via the imported `ConfigurationContract` instance and pass it to the `ObaSearchHelper` constructor. By putting the application key in the configuration file, I can easily change it at any time without recompiling my project.

The second thing I do is hook up the event `TargetViewChangedThrottled` from the `MapContract` instance. This event is raised every time the map's view is about to change, either programmatically or through user interaction. As the name suggests, the event is throttled internally so that it doesn't fire too many times within a short duration. Therefore, it's a perfect event to consider if you want to keep your entities in sync with the map view. In my case, I call the `RefreshBusStops` method to refresh the list of bus stops. Here's the definition of `RefreshBusStops`:

```
internal void RefreshBusStops()
{
    if (Map.ZoomLevel >= 14)
    {
        // Search within the radius of 1km, maximum 150 results
        _searchHelper.SearchStopsByLocation(
            Map.Center.Latitude, Map.Center.Longitude, radius: 1000, 150);
    }
    else
    {
        // Clear all bus stops
        ClearBusStops();
    }
}
```

Here, I check that the current map zoom level is at least 14, then I issue a call to the `SearchStopsByLocation` method, which will return the list of all bus stops within a specified radius around the map center. Otherwise, if the zoom level is less than 14, meaning the map is not zoomed in close enough to the city level, I clear out all bus stops.

VSP²

VISUAL STUDIO PARTNER PROFILE

A Visual Studio 2010 Q&A with Pamela Szabo of Stone Bond Technologies



Pamela Szabo
Executive Vice President & CTO
Stone Bond Technologies

www.agileintegrationsoftware.blogspot.com

Q What is Enterprise Enabler®?

A Enterprise Enabler is our established agile integration software platform that automates complex bi-directional data aggregation, transformation and migration of entities among multiple line of business applications.

Q What makes Enterprise Enabler® different?

A I think it is the holistic view in Enterprise Enabler of all the different aspects of integration. It is designed to cover the full range of needs and possibilities in a single environment. Our grass roots transformation engine that we built from the ground up, is not an off the shelf XSLT engine, which we have as well, but ours is far more flexible. It gives us the capability with our AppComm [Not Adapter] technology, to reach out to applications live and "talk" among the different applications that we're connecting. We do both: reading or writing and can align the data, so in the end we've got virtual relationships without having to stage data anywhere.

Q You call Enterprise Enabler "Agile Integration Software", what is that?

A That is really the end result of this holistic approach, characterized by 10 key points:

1. Off-the-shelf
2. Can create virtual relationships
3. Has embedded code editors and compilers
4. Has Design Time Validation
5. Has a transformation engine that is not limited to XML formats
6. A Single environment for development, testing, deployment and monitoring
7. Data Workflow logic
8. Secure, Auditable environment
9. Embeddable in software applications
10. Integration infrastructure change management

Q What is different for programmers?

A We don't want them to have to keep doing the same thing over and over again, so we have automated most every repeatable task, allowing the programmers to be innovative and do other things. Our Users love that they don't have to keep doing the same tedious tasks. Instead of wasting time, they can enhance the tools or extend it or drop in their own code, it becomes one with them in their environment. So it becomes their tool to do whatever they want with. Everything from building composite applications, to ecommerce and so forth. They even create their own reusable components they can drop into compilers right from the User Interface they are already working in.

Q What systems and applications can Enterprise Enabler extend connectivity for?

A Enterprise Enabler can be embedded in applications like ADO.Net, WSS, SharePoint 2010 and connect with systems like SAP, Salesforce, SQL, Oracle DB2, EDI. You name it, the product can codelessly connect!

Q What business value does Enterprise Enabler bring to an organization?

A There are so many benefits with Enterprise Enabler. For example, it eliminates the need for staging databases by providing real-time access to data aligned and merged from multiple sources. It also improves business decision-making by enhancing the capability of dashboards, business intelligence, analytics and SharePoint. The bottom line is a huge savings in resources and time to generate complex integration.

For more information please visit:
www.stonebond.com

STONE BOND
TECHNOLOGIES
Enterprise Enabler®
Agile Integration Software

Figure 8 Handler for the SearchBusStopsCompleted Event

```
private int _searchCount;
private Dictionary<BusStop, int> _busStopCache =
    new Dictionary<BusStop, int>();
private ObaLayer _layer;

private void OnSearchBusStopsCompleted(object sender, BusStopsEventArgs e)
{
    _searchCount++;

    // Contains new bus stops not present in the current view
    Collection<BusStopEntity> addedEntities =
        new Collection<BusStopEntity>();
    foreach (BusStop stop in e.BusStops)
    {
        // If this bus stop is not in the cache, it is a new one
        if (!_busStopCache.ContainsKey(stop))
        {
            addedEntities.Add(new BusStopEntity(stop, PushpinFactory));
        }

        // Marks this bus stop as being in the current search
        _busStopCache[stop] = _searchCount;
    }

    // Contains the old bus stops
    // that should be removed from the current view
    Collection<BusStopEntity> removedEntities =
        new Collection<BusStopEntity>();
    foreach (BusStopEntity bse in _layer.Entities)
    {
        // This bus stop belongs to the previous search,
        // add it to removed list
        if (_busStopCache[bse.BusStop] < _searchCount)
        {
            removedEntities.Add(bse);
            _busStopCache.Remove(bse.BusStop);
        }
    }

    // Tells the layer to add new in-view entities
    // and remove out-of-view ones
    _layer.RefreshEntities(addedEntities, removedEntities);
}
```

When the SearchStopsByLocation method completes (asynchronously), it raises the SearchStopsByLocationCompleted event, shown in **Figure 8**, which I subscribed to earlier in the Initialize method.

Note the object of type ObaLayer, which I'll explain in the next section. For now, suffice it to say that this object is responsible for managing the left panel's content and entities on the map.

Notice that I use a Dictionary<BusStop, int> object to help me keep track of the current list of displayed bus stops. With the help of this dictionary, every time I get a brand-new set of bus stops from the service call, I can quickly determine the new stops that aren't already shown, as well as the old stops that are now out of view due to the map view change.

You may wonder why I don't just clear out all current bus stops and show the new set of bus stops altogether. Well, the reason is performance. If I did so, I'd force all the pushpin controls to be re-created even if many of them were at the same location in both the old and new map views. Even worse, users would see a short flash when the pushpins were removed and quickly added back. My approach eliminates both of these shortcomings.

Show Content and Pushpins with a Layer

The Plugin class represents your map app, but if you want to show UI representations of it, you need to create *layers*. A layer is

an abstract concept that allows you to place entities on the map as pushpins (or polylines, or polygons) and to show custom UI content on the left panel. Optionally, it can also place an arbitrary UI overlay on top of the map, but I don't need that for my app. Most apps have only one layer.

If you've used Photoshop, you'll find the concept very similar to Photoshop layers. The History button (located at the bottom left corner of the page) shows the list of all currently loaded layers. You can choose to show or hide any layer, or you can set a layer to become active. When a layer becomes active, its Panel UI element is shown in the left panel and all of its entities are brought forward in the z-index stack.

In code, a layer is a subclass of the abstract Layer class. As shown in **Figure 9**, ObaPlugin creates and manages an instance of the ObaLayer class, which derives from Layer.

A plug-in is a unit of extensibility
that allows developers to add
functionalities in a maintainable,
decoupled way.

In the constructor of ObaLayer, I set the title of my layer, which will show at the top of the left panel. I also set the Panel property to an instance of the BusStopsPanel user control, which will occupy the entire left panel region if my layer becomes active. The user control's DataContext property is set to an instance of ObservableCollection<Entity>.

So how does the layer get displayed? That's done by ObaPlugin as shown in **Figure 10**.

The override Activate method is called every time my map app is activated through the Map apps gallery. To show my layer, I refer to the LayerManagerContract type I imported earlier. The LayerManagerContract class defines methods to work with layers. If my

Figure 9 ObaLayer Class

```
internal class ObaLayer : Layer
{
    private ObaPlugin _parent;
    private BusStopsPanel _resultPanel;
    private ObservableCollection<BusStopEntity> _busStopEntities;

    public ObaLayer(ObaPlugin parent) : base(parent.Token)
    {
        _parent = parent;

        // Contains the set of active bus stop entities
        // for data binding purpose
        _busStopEntities = new ObservableCollection<BusStopEntity>();
        _resultPanel = new BusStopsPanel(parent, this);
        _resultPanel.DataContext = _busStopEntities;

        this.Title = "Bus stops";
        this.Panel = _resultPanel;
    }
    ...
}
```

ALM Summit

Sponsored by Microsoft

Application Lifecycle Management
for the Microsoft platform

Register at www.alm-summit.com

follow us on



November 16-18, 2010 - Microsoft Redmond

Keynote speakers:



Ken Schwaber
Scrum.org



Brian Harry
Technical Fellow



Dave West
Forrester Research



Sam Guckenheimer
Microsoft



Tony Scott
CIO, Microsoft

Speakers:



Amit
Chopra



Austina
De Bonte



Cameron
Skinner



Chris
Kinsman



David
Green



Eric
Willeke



Grant
Holliday



Jamie
Cool



JB
Brown



Jim
Newkirk



John
Szurek



Jon
Bach



Karel
Deman



Mario
Cardinal



Mary
Czerwinski



Peter
Provost



Richard
Hundhausen



Stephanie
Cuthbertson



Stuart
McGill



Vinod
Malhotra

Sponsors:



Figure 10 ShowResultLayer Method Shows the ObaLayer Instance to the User

```
public override void Activate(IDictionary<string, string>
activationParameters)
{
    ShowResultLayer();
    RefreshBusStops();
}

private void ShowResultLayer()
{
    if (_layer == null)
    {
        _layer = new ObaLayer(this);
    }

    if (LayerManager.ContainsLayer(_layer))
    {
        LayerManager.BringToFront(_layer);
    }
    else
    {
        LayerManager.AddLayer(_layer);
    }
}
```

layer is already added, I set it to active by calling the `BringToFront` method. Attempting to add the same layer twice results in an exception.

In **Figure 8**, I called the `ObaLayer.RefreshEntities` method to update the bus stops. **Figure 11** shows its definition.

To add or remove an entity on the map, I use the `Layer.Entities` collection property. And for every new `BusStopEntity`, I call the `PopupContract.Register` method to register for the pop-up UI on my bus stop pushpin. The `Register` method accepts the entity and a callback method that gets invoked whenever the pop-up control changes state on the entity (see **Figure 12**).

Inside the pop-up callback, the method argument of `PopupStateChangeContext` type gives me access to the current state of the pop-up and the entity currently under pop-up. Based on those, I set the pop-up title with the bus stop name and the pop-up content with the `BusStopEntity.BusRoutesAsString` property, which returns a comma-separated list of bus routes serving this particular bus stop.

If the pop-up is in Normal state, I also add three contributor links to the pop-up via the `Contributors` collection property—the “Arrival times” contributor shows the arrival times for this bus stop, the `Directions` contributor invokes the driving directions feature and the `Streetside` contributor switches to street-level map mode.

A contributor is represented by a hyperlink at the bottom of the pop-up control. It allows map apps to invoke a certain core feature of Bing Maps. To generate contributors, you call one of the two methods of the `ContributorHelperContract` type: `CreateAsyncContributor` or `CreateDemandLoadContributor`. Both methods return a proxy contributor synchronously and defer the loading of the real contributor instance. The only difference is that the `CreateAsyncContributor` loads the real contributor as soon as it returns, whereas `CreateDemandLoadContributor` only does so when the proxy contributor link is invoked the first time.

BusStopsPanel UserControl

`BusStopsPanel` is a `UserControl`, which is responsible for showing the list of in-view bus stops in the left panel (as shown in **Figure 1**). It contains an `ItemsControl` instance with the `ItemTemplate`

property set to a custom `DataTemplate` (see **Figure 13**). Notice that I turn on UI virtualization mode for my `ItemsControl` by setting the `ItemsPanel` property to use a `VirtualizingStackPanel`. Generally, it's a good practice to apply UI virtualization to `ItemsControl` if your app may load hundreds of items into it.

Inside the bus stop `DataTemplate` I added a `HyperlinkButton` control, which, when clicked, will trigger the search for the arrival times of the corresponding bus stop:

```
private void OnBusStopClick(object sender, RoutedEventArgs e)
{
    FrameworkElement element = (FrameworkElement)sender;
    BusStopEntity entity = (BusStopEntity)element.DataContext;
    _plugin.SearchArrivalTimes(entity);
}
```

Notice that its `Style` property is set to an object from the `StaticResource` collection, with the key as “App.P2.Hyperlink.” Bing Maps provides a default set of UI resources, such as `Styles` and `ControlTemplates` of common controls, as well as standard `Colors` and `Brushes` used by Bing Maps itself. Map app authors are encouraged to apply these resources to their UI elements so they have the same look and feel as native UI elements. Refer to the documentation for all the resources provided by Bing Maps.

`SearchArrivalTimes` is an internal method of the `ObaPlugin` class. It calls the `ObaSearchHelper.SearchArrivalTimesForStop` method to retrieve arrival times for the specified bus stop:

```
internal void SearchArrivalTimes(BusStopEntity _entity)
{
    _searchHelper.SearchArrivalTimesForStop(_entity.StopId, _entity);
}
```

When the search completes, the plug-in will instruct `ObaLayer` to show the arrival times in the left panel. `ObaLayer` does so by dynamically changing its `Title` and `Panel` properties.

Testing and Debugging the Map App

When you're done coding, you'll want to test your app. The Bing Maps site has a developer mode that's enabled by appending a “developer=1” query parameter to the URL, like this: `http://www.bing.com/maps/explore/?developer=1`. When in developer mode, you test your map app with the “Map app test tool,” which can be activated via the

Figure 11 The `ObaLayer.RefreshEntities` Method Definition

```
public void RefreshEntities(
    ICollection<BusStopEntity> addedEntities,
    ICollection<BusStopEntity> removedEntities)
{
    foreach (BusStopEntity entity in removedEntities)
    {
        // Remove this pushpin from the map
        this.Entities.Remove(entity);
        // Remove this bus stop entry from the panel
        _busStopEntities.Remove(entity);
    }

    foreach (BusStopEntity entity in addedEntities)
    {
        // Add this pushpin to the map
        this.Entities.Add(entity);
        // Add this bus stop entry to the panel
        _busStopEntities.Add(entity);

        // Register this entity to have popup behavior
        _parent.PopupContract.Register(entity, OnPopupStateChangeHandler);
    }
}
```

same Map apps gallery. The Map app test tool allows you to select the plug-in assemblies from your local hard drive. It then loads your plug-in into the site just as it does with all native plug-ins. To debug

your code, attach VS.NET to your browser while your app is loaded. Make sure you set the debug code type to Silverlight.

Figure 12 Changing the State of a Pop-Up Control

```
private void OnPopupStateChangeHandler(PopupStateChangeContext context)
{
    if (context.State == PopupState.Closed)
    {
        return;
    }

    BusStopEntity entity = (BusStopEntity)context.Entity;
    context.Title = entity.Name;
    context.Content = "Bus numbers: " + entity.BusRoutesAsString;

    // Only shows contributors link in the normal state of popup
    if (context.State == Pop-upState.Normal)
    {
        // Add Arrival times contributor
        context.Contributors.Add(new BusStopContributor(_parent, entity));

        Dictionary<string, object> parameter = new Dictionary<string, object>
        {
            {"Entity", entity}
        };
        var contributorHelper = _parent.ContributorHelper;

        // Add Directions contributor
        context.Contributors.Add(contributorHelper.
            CreateDemandLoadContributor(
                "Microsoft/DirectionsContributorFactoryContract",
                parameter, "Directions"));

        // Add Streetside contributor
        context.Contributors.Add(contributorHelper.CreateAsyncContributor(
            "Microsoft/StreetsideContributorFactoryContract", parameter));
    }
}
```

Figure 13 The BusStopsPanel UserControl

```
<UserControl.Resources>
<DataTemplate x:Key="BusStopTemplate">
<Border Margin="2,0,2,12">
<StackPanel>
<TextBlock Text="{Binding Name}" FontSize="14" FontWeight="Bold"
TextWrapping="Wrap" />
<TextBlock Text="{Binding BusRoutesAsString, StringFormat='Bus numbers:
{0}'}" FontSize="12" TextWrapping="Wrap" />
<HyperlinkButton Content="Arrival times" Click="OnBusStopClick"
Style="{StaticResource App.P2.Hyperlink}"
HorizontalAlignment="Left" />
</StackPanel>
</Border>
</DataTemplate>

<ControlTemplate x:Key="ScrollableItemsControl"
TargetType="ItemsControl">
<ScrollViewer Style="{StaticResource App.ScrollViewer}"
VerticalScrollBarVisibility="Auto">
<ItemsPresenter />
</ScrollViewer>
</ControlTemplate>
</UserControl.Resources>

<ItemsControl
ItemsSource="{Binding}"
VirtualizingStackPanel.VirtualizationMode="Recycling"
ItemTemplate="{StaticResource BusStopTemplate}"
Template="{StaticResource ScrollableItemsControl}">
<ItemsControl.ItemsPanel>
<ItemsPanelTemplate>
<VirtualizingStackPanel />
</ItemsPanelTemplate>
</ItemsControl.ItemsPanel>
</ItemsControl>
```

Submitting Your Map App

Finally, when you're satisfied with your code, you can submit your app to be published officially onto the Bing Maps site. You can submit a new app and view the status of previous submissions at the Bing Maps Account Center (bingmapsportal.com). The SDK documentation has detailed instructions on submitting your app, and it lists the requirements your apps must meet in order to be approved.

Call to Action

So there you have it: a full-fledged, real-time transit application that didn't require a lot of code. The SDK supplies the building blocks that make writing map-centric applications an enjoyable and rewarding experience. I encourage you to download the SDK today, learn it, and start writing your own map apps. ■

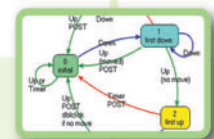
LUAN NGUYEN worked as a developer on the Bing Maps team (formerly known as Microsoft Virtual Earth) for almost four years. He was a member of the Shell feature team, which was responsible for the framework shell of the Bing Maps site and the Bing Map App SDK. He has recently switched to the ASP.NET team.

THANKS to the following technical experts for reviewing this article:

Alan Paulin, Chris Pendleton, Dan Polivy and Greg Schechter

Build interactive diagrams easier than you ever imagined

Create custom interactive diagrams, network editors, workflows, flowcharts and design tools. For web servers or local applications. It's easy-to-use and very flexible. We're the first developer of diagramming components and still the best. Find out for yourself: download our FREE fully functional evaluation kit, with full support at www.nwoods.com.



Now supporting
WPF & Silverlight

GoDiagram

Interactive diagram components

www.nwoods.com

Connected Devices Using the .NET Micro Framework

Colin Miller

Today, we see a proliferation of applications that include connected devices. In fact, the “Internet of Things” (devices connected over the Internet) is already estimated to be larger than the World Wide Web in terms of endpoints and is projected to grow to an order of magnitude larger in the next few years.

In the near future, we’ll be interacting with more intelligent devices instead of recognizable computers. Just look around your house. The things that can be usefully connected include your appliances (for energy management, software updates and maintenance); your

car (for coordination on recharging your new electric vehicle with the grid, automatic testing and maintenance and software updates); your irrigation system (for scheduling based on weather reports and water management); your pets (to determine their location and configure invisible fencing); your thermostat (for remote control); and much more.

These devices are connected to one another, to smart controllers, to the router and to the cloud. What does this mean for Microsoft .NET Framework developers? Right now, .NET developers can produce applications for all parts of the system to which small devices connect. And with the .NET Micro Framework, .NET developers can develop the full system right down to the small devices.

The .NET Micro Framework is an implementation of the .NET Framework specifically targeting the smallest devices possible with extensions for embedded programming needs. It runs directly on the hardware without an underlying OS in order to minimize its footprint. General information is available at microsoft.com/netmf and the project’s open source community site: netmf.com.

Several months ago, I started a series of articles on the .NET Micro Framework blog (blogs.msdn.com/b/netmfteam) that describes the process of building one of these small devices—a bicycle computer—from the ground up using only the .NET Micro Framework, Visual Studio and a minimal amount of electronics. I wanted to demonstrate how .NET Framework developers can build rich applications on small devices. (The first article in this series can be found at: tinyurl.com/2dpy6rx.) **Figure 1** shows the actual computer. I picked a bicycle computer because everyone is a domain expert when it

This article discusses:

- The NETMF bicycle computer open source project
- Using the Web services model to connect devices
- The Devices Profile for Web Services
- Configuring a Wi-Fi radio
- Setting up a service connection
- Using WDSL to define the service
- Generating code with MFSvcUtil.exe
- Uploading bicycle ride data

Technologies discussed:

.NET Micro Framework

Code download available at:

code.msdn.microsoft.com/mag201010Micro

comes to bicycles. The application includes a gesture-based UI, supports a number of sensors and addresses issues like managing the battery power supply.

The blog discusses each feature's implementation, and the code for the project is available on CodePlex at netmfbikecomputer.codeplex.com/. However, I saved the best for last—in this article, I'm going to connect this device to a Microsoft Windows Azure-hosted Web service over Wi-Fi. Here's the scenario: you've just finished your ride and brought your bike into the garage. Your computer contains data it collected during your ride: distance, speed, cadence, incline, time and so on. You flip over to the data view and press the Upload button. The data for your ride is uploaded to the cloud, where it can be aggregated with all your other rides and shared with your friends.

In this article, I'll focus on making the connection and uploading the data rather than discussing the form of the cloud service to which you might connect. Check out examples like bikejournal.com and cyclistats.com to see what the possibilities are for tracking your bicycling progress and competing with your friends. I'll discuss how easy it is to make that connection.

First, a Little Background

The Web services model is great for connecting devices because it supports a full range of device and service interactions that may not be fully known at the time we create an application. On the device side, we use a subset of the full Web service infrastructure called Devices Profile for Web Services (DPWS)—see en.wikipedia.org/wiki/Devices_Profile_for_Web_Services to learn more about it. DPWS is described as Universal Plug and Play (UPNP) for networked devices. In the .NET Micro Framework, DPWS supports the WS-Addressing, WS-Discovery, WS-MetadataExchange and WS-Eventing interfaces, and is built on the underlying technologies of SOAP, XML, HTTP, MTOM and Base64 encoding.

With DPWS, you can connect to devices that are clients (devices that consume services from others), or servers (devices that provide services for others) or both. You can negotiate what you provide and what you can consume through the metadata, and you can publish and subscribe to notifications of changes in other entities. **Figure 2** shows the DPWS stack.

The implementation in the .NET Micro Framework supports DPWS version 1.0—which is compatible with Windows 7—and DPWS version 1.1—which is compatible with Windows Communication Foundation (WCF) 4. You can specify the binding you use for your connection: for example, SOAP over HTTP (`ws2007HttpBinding`) or a custom binding that you want to support.



Figure 1 The NETMF Bicycle Computer

Our application for the bicycle computer is really pretty simple—we'll just be uploading data to the cloud. DPWS can actually do so much more. For example, let's say my utility company installs a smart service meter that limits the resources I can use at any specific time. Along with that, I install a local energy management service that gives me control over how I use those limited resources. I can set priorities and heuristics that allow the system to make decisions about how to limit consumption—for example, hot water for showers has a high priority.

Now I go out and buy a new dishwasher. I bring it home and plug it in. In the background, the dishwasher finds the local network and "discovers" the management service. It informs the service of its power consumption in various states and rules for

how it can be used. Later, when the dishwasher is running, I jump in the shower and the hot water heater comes on. But once I start the dishwasher, I don't want it to just turn off and have the food harden on the plates. To cut total consumption, the management service tells the dishwasher to just rinse the dishes every 15 minutes to keep them from drying out, and to resume where it left off in the washing cycle when I'm done with the shower. As you can see, this entire scenario can support an arbitrary set of endpoints with the functionality defined in DPWS.

Enough Background—Let's Make It Work

Configuring the Wi-Fi radio is easy enough. There are two ways to do this—one uses the MFDeploy.exe utility, and the other is a programmatic interface supported by the GHI SDK (from GHI Electronics). Let's start with MFDeploy.exe, which is a tool that comes with the .NET Micro Framework and can be found in the tools section of the SDK installation. In the Target | Configuration | Network Configuration dialog box (see **Figure 3**), I enable DHCP,

select the security mechanism and enter the pass phrase and other configuration aspects of my home network.

DHCP will take care of filling in the Gateway and DNS field of the network settings. This information is available to the managed application through the `NetworkInterface` type and its child `Wireless80211`. The HTTP stack will implicitly use this information when sending and receiving bytes, but it might need an additional piece of information to enable using a proxy—something your network might require. To help the HTTP stack use the proxy correctly, it's advisable to add the following to a program as a further hint as to where to connect:

```
WebRequest.DefaultWebProxy =  
    new WebProxy("<router IP Address>");
```

If there's no explicit proxy role in your network, you can usually default to using the Gateway

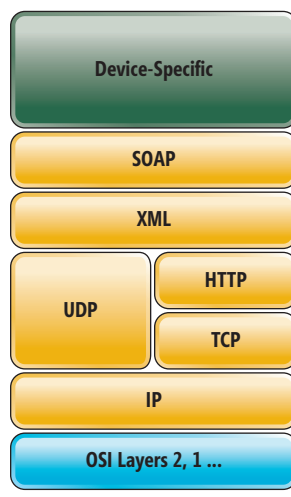


Figure 2 The DPWS Stack

address. With the GHI programmatic interface, you use some derivative of the code shown in **Figure 4**.

The **Figure 4** example assumes that you're using WPA or WPA2 security. WEP is also supported. The first thing you do is identify the SPI port and control lines used by the radio. This configuration represents the connections for the hardware, a FEZ Cobra board from GHI. All that's required is to set up the `WiFiSettings`, save the configuration and call `EnableDHCP`. Notice that some of these calls are blocking, and this can take some time, so you'll need to ensure that the user knows what's happening. Also, there's no way in this programmatic interface to enumerate the available networks so you can select from them. I hard-coded the network information in the **Figure 4** example.

For a commercial implementation of the bicycle computer, I need to write an integrated Wi-Fi configuration UI that exposes the information entered into the dialog shown in **Figure 3**, and an onscreen keyboard to enter it. I may have time to write this up in another blog article before this article is published. I'm working on a Time Service article that shows how to use the Wi-Fi connection to get the date and time when I start up the computer so that I don't have to keep the device running to maintain that information or have the user (me) enter it on startup.

Setting up the Service Connection

All that remains is the DPWS implementation. As a reminder, I'm going to focus on the device side. The Windows Azure service I'm using provides a simple "Hello World" template. I start with this and

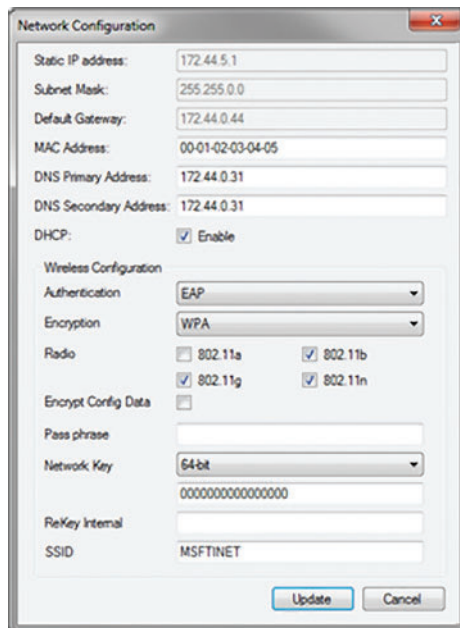


Figure 3 The MFDeploy Network Configuration Dialog

extend the contract by adding the fields that I need to store and the operations by writing the `Upload` and `Get` operations. This creates a service that can accept my data, store it and give me back the last data stored. Obviously, a full service requires much more work, but that's another article. Let's look briefly at the contract created for this service. The `ServiceContract` contains the two operations and the `DataContract` contains the fields (see **Figure 5**).

The actual implementation of the contract is minimal to support the bicycle computer example (see **Figure 6**).

The WSDL Defines the Service

From the contract and schema that I created, the Windows Azure service auto-generates a Web Service Definition Language (WSDL) file. It contains the Service Modeling Language (SML) definition of the service. The W3C specification for the WSDL document can be found at w3.org/

`TR/wSDL`. It defines the operations and messages that the service supports. The WSDL is stored as an XML description on a Web site, where it's accessible to anyone connecting to the service. Ours can be found at netmfbikecomputerservice.cloudapp.net/BikeComputerService.svc?wsdl. **Figure 7** shows a small snapshot of the WSDL file so you can see what it looks like, but remember that this file is auto-generated and is only consumed by other programs. You don't need to ever write this complex and touchy XML.

You can see that the WSDL includes the definitions for messages for data input and output and operations for getting and uploading data. Now that we have our simple interface to the service defined and published, how do I program to that? That's pretty easy as well.

Figure 4 Wi-Fi Configuration with the GHI Programmatic Interface

```
// -- Set up the network connection -- //
WiFi.Enable(SPI.SPI_module.SPI2, (Cpu.Pin)2, (Cpu.Pin)26);

NetworkInterface[] networks = NetworkInterface.GetAllNetworkInterfaces();
Wireless80211 WiFiSettings = null;

for (int index = 0; index < networks.Length; ++index)
{
    if (networks[index] is Wireless80211)
    {
        WiFiSettings = (Wireless80211)networks[index];
        Debug.Print("Found network: " + WiFiSettings.Ssid.ToString());
    }
}

WiFiSettings.Ssid = "yourSSID";
WiFiSettings.PassPhrase = "yourPassphrase";
WiFiSettings.Encryption = Wireless80211.EncryptionType.WPA;
Wireless80211.SaveConfiguration(
    new Wireless80211[] { WiFiSettings }, false);

_networkAvailabilityBlocking = new ManualResetEvent(false);

if (!WiFi.IsLinkConnected)
{
    _networkAvailabilityBlocking.Reset();
    while (!_networkAvailabilityBlocking.WaitOne(5000, false))
    {
        if (!WiFi.IsLinkConnected)
        {
            Debug.Print("Waiting for Network");
        }
        else
        {
            break;
        }
    }
    Debug.Print("Enable DHCP");
    try
    {
        if (!WiFiSettings.IsDhcpEnabled)
        {
            WiFiSettings.EnableDhcp(); // This function is blocking
        }
        else
        {
            WiFiSettings.RenewDhcpLease(); // This function is blocking
        }
    }
    catch
    {
        Debug.Print("DHCP Failed");
    }
}
```



BEST SELLER

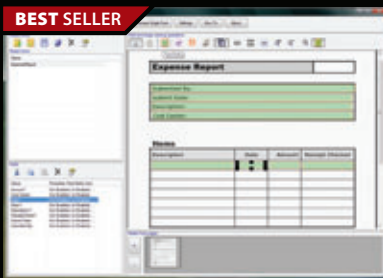


FusionCharts | from \$195.02

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 16,000 customers and 330,000 users in 110 countries

BEST SELLER



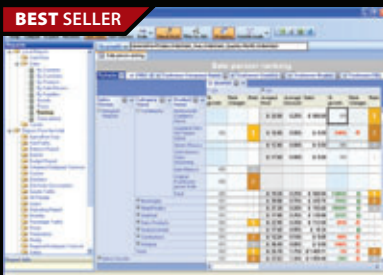
LEADTOOLS Recognition SDK | from \$3,595.50



Add robust 32/64 bit document imaging & recognition functionality into your applications.

- Features accurate, high-speed multi-threaded OCR and forms recognition
- Supports text, ICR, OMR, image, and 1D/2D barcode fields
- Auto-registration and clean-up to improve recognition results
- Includes .NET, C/C++, WPF, WF, WCF and Silverlight interfaces
- Latin character set support is included. Arabic and Asian support is available.

BEST SELLER



ContourCube | from \$900.00



OLAP component for interactive reporting and data analysis.

- Embed Business Intelligence functionality into database applications
- Zero report coding - design reports with drag and drop
- Self-service interactive reporting - get hundreds of reports by managing rows/columns
- Royalty free - only development licenses are needed
- Provides extremely fast processing of large data volumes

BEST SELLER



TX Text Control .NET and .NET Server | from \$499.59



Word processing components for Visual Studio .NET.

- Add professional word processing to your applications
- Royalty-free Windows Forms 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

Generating Code with MFSvcUtil.exe

On the desktop, there's a utility called the ServiceModel Metadata-Utility Tool (SvcUtil.exe) that can generate service model code from metadata documents (like our WSDL) and vice versa. The .NET Micro Framework has a similar utility called MFSvcUtil.exe. This is a command-line tool that's best run in the project directory. So, we run it by pointing it at the published WSDL specification:

```
<SDK_TOOLS_PATH>\MFSvcUtil.exe http://netmfbikecomputerservice.cloudapp.net/BikeComputerService.svc?wsdl
```

The tool generates three files (see **Figure 8**).

Remember that devices can be clients, servers or both.

The BikeComputerService.cs file contains the definition of the data in the messages, the classes defining the operations supported by the service (because our device is a client) and a number of helper functions for serializing and de-serializing the data (see **Figure 9**).

The BikeComputerClientProxy.cs file contains the proxy interfaces for the Web service:

```
namespace BikeComputer.org
{
    public class IBikeComputerServiceClientProxy : DpwsClient
    {
        private IRequestChannel m_requestChannel = null;

        public IBikeComputerServiceClientProxy(Binding binding,
            ProtocolVersion version) : base(binding, version)...

        public virtual GetLastComputerDataResponse
            GetLastComputerData(GetLastComputerData req) ...

        public virtual UploadBikeComputerDataResponse
            UploadBikeComputerData(UploadBikeComputerData req) ...
    }
}
```

The third file that MFSvcUtil.exe creates is the BikeComputerServiceHostedService.cs file; this contains the interface logic that runs on the service side of the interface. In our case, the WSDL was generated from the service and data contracts that were created, so this file is a “throw away” for us. It's generated to cover scenarios where you get a published WSDL and you want to replicate a service from it or where you want to run a service on another device. Remember that devices can be clients, servers or both. The option to have devices provide services for other devices enables some interesting applications. This is what BikeComputerServiceHostedService contains:

```
namespace BikeComputer.org
{
    public class IBikeComputerServiceClientProxy : DpwsHostedService
    {
        private IBikeComputerService m_service;

        public IBikeComputerService(IBikeComputerService service,
            ProtocolVersion version) : base(version) ...

        public IBikeComputerService(IBikeComputerService service) :
            this(service, new ProtocolVersion10())...

        public virtual WsMessage GetLastComputerData(WsMessage request) ...

        public virtual WsMessage UploadBikeComputerData(WsMessage request) ...
    }
}
```

Figure 5 The Service Contract

```
[ServiceContract]
public interface IBikeComputerService
{
    [OperationContract]
    BikeComputerData GetLastComputerData();

    [OperationContract]
    void UploadBikeComputerData(BikeComputerData rideData);
}

// Use a data contract as illustrated in the sample below
// to add composite types to service operations.
[DataContract]
public class BikeComputerData
{
    DateTime _Date;
    TimeSpan _StartTime;
    TimeSpan _TotalTime;
    TimeSpan _RidingTime;
    float _Distance;
    float _AverageSpeed;
    float _AverageCadence;
    float _AverageIncline;
    float _AverageTemperature;
    bool _TempIsCelcius;

    [DataMember]
    public DateTime Date...

    [DataMember]
    public TimeSpan StartTime...

    [DataMember]
    public TimeSpan TotalTime...

    [DataMember]
    public TimeSpan RidingTime...

    [DataMember]
    public float Distance...

    [DataMember]
    public float AverageSpeed...

    [DataMember]
    public float AverageCadence...

    [DataMember]
    public float AverageIncline...

    [DataMember]
    public float AverageTemperature...

    [DataMember]
    public bool TemperatureIsInCelcius...
}
```

Figure 6 The Contract Implementation

```
public class BikeComputerService : IBikeComputerService
{
    static BikeComputerData _lastData = null;

    public BikeComputerData GetLastComputerData()
    {
        if (_lastData != null)
        {
            return _lastData;
        }
        return new BikeComputerData();
    }

    public void UploadBikeComputerData(BikeComputerData rideData)
    {
        _lastData = rideData;
    }
}
```

Your Recipe, Our Kitchen

Got the recipe for a great application?
We've got the tools to help make it even better.



With Microsoft Platform Ready you can develop a great application, find new customers, and generate new revenue streams.

Help yourself to the tools you need to speed your time to market, including one-on-one technical support, software development kits, and tailor-made marketing materials.

Get started with MPR today
microsoftplatformready.com

With Microsoft Platform Ready, you can get your applications compatible with the latest Microsoft technologies:

- Windows® Azure™ Platform
- Windows 7
- Windows Server® 2008 R2
- Microsoft SharePoint® Server 2010
- Microsoft SQL Server® 2008 R2
- Microsoft Office 2010
- Microsoft Exchange Server 2010
- Microsoft Dynamics® CRM 2011
- Windows Phone 7



Microsoft Partner Network™

```

<?xml version="1.0" encoding="utf-8" ?>
wsdl:definitions name="BikeComputerService" targetNamespace="http://BikeComputer.org/" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xs="http://schemas.xmlsoap.org/ws/2004/09/mex" xmlns:wsa10="http://www.w3.org/2005/08/addressing"
xmlns:tns="http://BikeComputer.org/" xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/" xmlns:wsu="http://docs.oasis-
open.org/ws/2004/01/oasis-200401-wss-wsssecurity-utility-1.0.xsd" xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:wspag="http://schemas.xmlsoap.org/ws/2004/09/addressing/policy" xmlns:msc="http://schemas.microsoft.com/ws/2005/12/wsdl/contract"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing" xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
wsp:Policy Id="WSHttpBinding_IBikeComputerService_policy"
wsp:ExactlyOne>
wsp:All>
wsaw:UsingAddressing />
</wsp:All>
</wsp:ExactlyOne>
</wsp:Policy>
wsdl:types>
xsd:schema targetNamespace="http://BikeComputer.org/Imports">
xsd:import schemaLocation="http://netmfbikecomputerservice.cloudapp.net/BikeComputerService.svc?xsd=xsd0" namespace="http://BikeComputer.org/" />
xsd:import schemaLocation="http://netmfbikecomputerservice.cloudapp.net/BikeComputerService.svc?xsd=xsd1"
namespace="http://schemas.microsoft.com/2003/10/Serialization/" />
xsd:import schemaLocation="http://netmfbikecomputerservice.cloudapp.net/BikeComputerService.svc?xsd=xsd2"
namespace="http://schemas.datacontract.org/2004/07/BikeComputerServiceWebRole" />
</xsd:schema>
</wsdl:types>
wsdl:message name="IBikeComputerService_GetLastComputerData_InputMessage">
wsdl:part name="parameters" element="tns:GetLastComputerData" />
</wsdl:message>
wsdl:message name="IBikeComputerService_GetLastComputerData_OutputMessage">
wsdl:part name="parameters" element="tns:GetLastComputerDataResponse" />
</wsdl:message>
wsdl:message name="IBikeComputerService_UploadBikeComputerData_InputMessage">
wsdl:part name="parameters" element="tns:UploadBikeComputerData" />
</wsdl:message>
wsdl:message name="IBikeComputerService_UploadBikeComputerData_OutputMessage">
wsdl:part name="parameters" element="tns:UploadBikeComputerDataResponse" />
</wsdl:message>
wsdl:portType name="IBikeComputerService">
wsdl:operation name="GetLastComputerData">
wsdl:input wsaw:Action="http://BikeComputer.org/IBikeComputerService/GetLastComputerData"
message="tns:IBikeComputerService_GetLastComputerData_InputMessage" />
wsdl:output wsaw:Action="http://BikeComputer.org/IBikeComputerService/GetLastComputerDataResponse"
message="tns:IBikeComputerService_GetLastComputerData_OutputMessage" />
</wsdl:operation>
wsdl:operation name="UploadBikeComputerData">
wsdl:input wsaw:Action="http://BikeComputer.org/IBikeComputerService/UploadBikeComputerData"
message="tns:IBikeComputerService_UploadBikeComputerData_InputMessage" />
wsdl:output wsaw:Action="http://BikeComputer.org/IBikeComputerService/UploadBikeComputerDataResponse"
message="tns:IBikeComputerService_UploadBikeComputerData_OutputMessage" />
</wsdl:operation>
</wsdl:portType>
</wsdl:definitions>

```

Figure 7 The WSDL File

Uploading the Data

As you've seen, all the device application code so far has been auto-generated from the WSDL. What code do you actually have to write on the client to connect to the service and post your data and read it back just to be sure that it's been received? Just a few lines of code are required. Here's what I did in the bicycle computer project.

In the RideDataModel class, I added the following to the constructor to set up the DPWS connection:

```

public RideDataModel()
{
    _currentRideData = new CurrentRideData();
    _summaryRideData = new SummaryRideData();
    _today = DateTime.Now; //change this to the time service later.

    //--Setup the Web Service Connection
    WS2007HttpBinding binding = new WS2007HttpBinding(
        new HttpTransportBindingConfig(new Uri
            ("http://netmfbikecomputerservice.cloudapp.net/BikeComputerService.svc"))

```

```
));
```

```

m_proxy = new
    IBikeComputerServiceClientProxy(
        binding, new ProtocolVersion11());

_upload = new
    UploadBikeComputerData();

_upload.rideData = new
    schemas.datacontract.org.
    BikeComputerServiceWebRole.
    BikeComputerData();
}

```

Next, I created a method in that class for uploading the data to the Web service. This routine puts my summary data from the ride into the fields for the Web service schema (referenced in the WSDL and reflected in BikeComputerService.cs), which is designed to match that data. Then I invoke the proxy's UploadBikeComputerData method with the upload data and retrieve the date of the last ride on the Web service to validate that my data was received (see Figure 10).

This assumes that you take only one ride a day, so if you do take

more rides, you need to change that comparison logic. I expect to use the "Pause" feature on the bicycle computer and treat all the rides in a day as one set of data. I already have the ability to store the data in a file on an SD card on the bike as I described in a

Navigate to the Save Your Data screen with a simple gesture.

previous article on the blog. I'll add the ability to track which sets of ride data have been posted to the Web service and post any that are missing. This allows me to be out of range of my wireless connection for some time and still be able to update the Web service

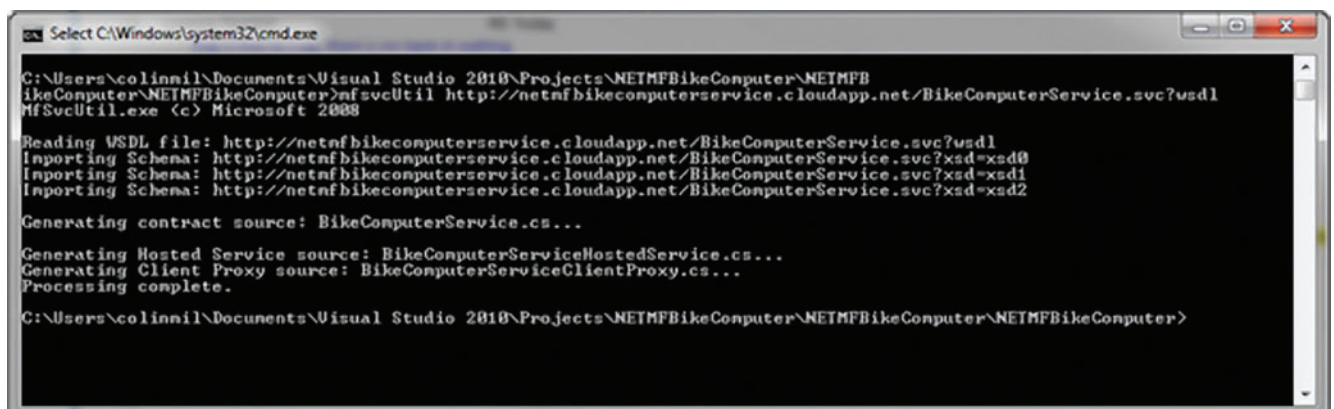
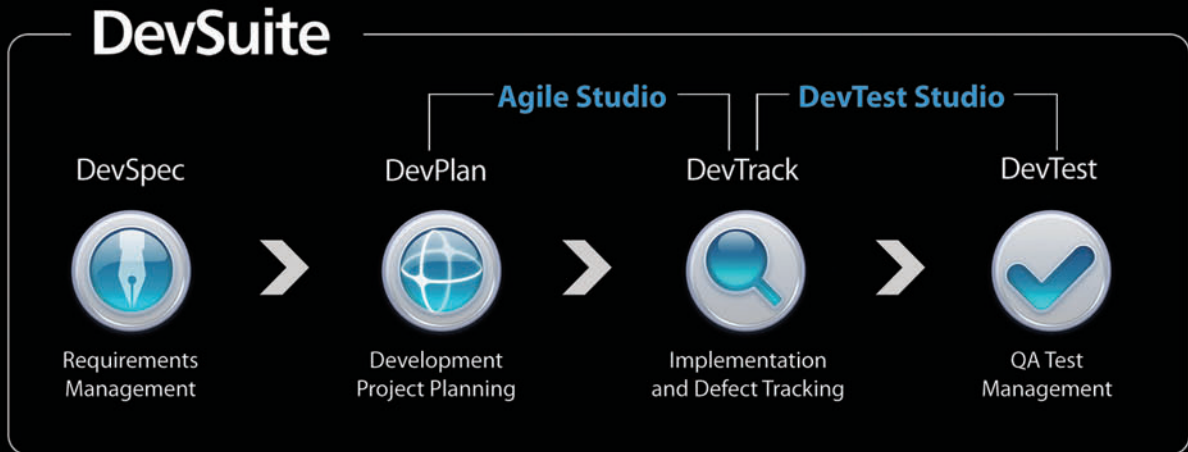


Figure 8 Executing the MfsvcUtil.exe Command

Gain Full Traceability

Requirements Driven Quality Management



A Modular Approach to ALM

- DevSuite can be purchased and deployed as a complete ALM solution or as individual modules
- Convenient bundles provide solutions for quality management (DevTest Studio) and Agile development (Agile Studio)
- Individual modules and bundles can be easily expanded when you need additional functionality

Agile Ready

- DevSuite is a hybrid platform that allows you to mix elements from multiple methods, including traditional development, to achieve the right balance for your team
- Out-of-the box methodology templates for Scrum, XP, Test Driven, Waterfall, and Iterative development

TechExcel



Download your FREE 30-day trial of DevSuite now at: www.techexcel.com/downloads

www.techexcel.com

1-800-439-7782

Figure 9 The BikeComputerService.cs File

```
namespace BikeComputer.org
{
    [DataContract(Namespace="http://tempuri.org/")]
    public class GetLastComputerData ...

    public class GetLastComputerDataDataContractSerializer :
    DataContractSerializer...

    [DataContract(Namespace="http://tempuri.org/")]
    public class GetLastComputerDataResponse ...

    public class GetLastComputerDataResponseDataContractSerializer :
    DataContractSerializer...

    [DataContract(Namespace="http://tempuri.org/")]
    public class UploadBikeComputerData ...

    public class UploadBikeComputerDataDataContractSerializer :
    DataContractSerializer...

    [ServiceContract(Namespace="http://tempuri.org/")]
    [PolicyAssertion(Namespace="http://schemas.xmlsoap.org/ws/2004/09/
    policy",
        Name="ExactlyOne",
        PolicyID="WSHttpBinding_IBikeComputerService_policy")]
    public interface IBikeComputerService ...
}
namespace schemas.datacontract.org.BikeComputerServiceWebRole...
```

later. Another enhancement is to enable connecting to a PC as an intermediary when my home network isn't available.

So, 17 lines of code that I wrote in the application (most of it mapping the summary data into the service schema fields) and I'm uploading data to a service and doing validity checks. Not bad.

Now We End Our Ride Connected

At the end of a ride, as I come into my garage, I can navigate to the Save Your Data screen with a simple gesture and post my data up to the cloud.

Figure 10 Uploading Data to the Web Service

```
public bool postDataToWS()
{
    //-- Load the ride summary data into the upload fields --//

    _upload.rideData.AverageCadence = _summaryRideData.averageCadence;
    _upload.rideData.AverageIncline = _summaryRideData.averageIncline;
    _upload.rideData.AverageSpeed = _summaryRideData.averageSpeed;
    _upload.rideData.AverageTemperature =
    _summaryRideData.averageTemperature;
    _upload.rideData.Date = _summaryRideData.rideDate;
    _upload.rideData.Distance = _summaryRideData.distance;
    _upload.rideData.RidingTime = _summaryRideData.ridingTime;
    _upload.rideData.StartTime = _summaryRideData.startTime;

    //-- Upload the data --//

    m_proxy.UploadBikeComputerData(_upload);

    //-- Validate the upload by retrieving the data and comparing --//

    GetLastComputerData req = new GetLastComputerData();

    GetLastComputerDataResponse back = m_proxy.GetLastComputerData(req);

    if (back.GetLastComputerDataResult.Date == _upload.rideData.Date)
    {
        return false;
    }

    return true;
}
```

There's still some work to do at that end to make the data more useful, but that's another story.

Embedded devices have been islands of specialized technology where ease of programming and flexibility were traded off for small footprint/low-cost and high-performance requirements. More and more frequently, we're seeing small devices connected to other devices and to networks to create compelling solutions. At the same time, the price of processors and memory continues to go down to the point where we no longer have to give up the productivity of desktop tools and languages to be able to make rich and price-competitive devices. The result is that .NET programmers find themselves with both the skills and the opportunities to work on small devices.

.NET Framework
programmers can now add
"Embedded Programmer"
to their business cards.

The Web services model provides a strong option for connecting these small devices, because discovering remote services, subscribing to remote events and exchanging information about services through metadata enables the flexible connection of a number of devices. The cost is that the connection—using SOAP and XML—is verbose, so it may not be appropriate in all cases.

The bicycle computer project demonstrates that .NET Framework programming skills enable you to write compelling UIs for small devices, write drivers for a variety of sensors and connect those devices with the cloud. As I showed you, the work involves defining the Web service, and the majority of the code that runs on the device is auto-generated by MFSvcUtil.exe. Just a few lines of additional code were required to upload the data. Other applications may require probing for the Web service (WS_Discovery), or subscribing to events on other endpoints (WS_Eventing), or dealing with devices and services of varying capabilities (WS_MetaDataExchange). All of these can be added to the basic data exchange model as needed.

As a friend of mine once commented, .NET Framework programmers can now add "Embedded Programmer" to their business cards. We'd love to hear about the devices that you build with .NET on the Discussions at netmf.com, where I can also answer any questions that you have on this article or on the blog site.

Happy riding! ■

COLIN MILLER started his affair with computing as a scientific programmer, building 8- and 16-bit experimental control systems. He strayed from small devices, working for 25 years (including 15 at Microsoft) on PC software including databases, desktop publishing, consumer products, Word, Internet Explorer, Passport (LiveID) and online services. As the product unit manager of the .NET Micro Framework, he has finally (and happily) been able to merge these disparate parts of his career.

THANKS to the following technical experts for reviewing this article:
Jane Lawrence and Patrick Butler Monterde

Esri® Developer Network

Integrate Mapping and GIS into Your Applications



Give your users an effective way to visualize and analyze their data so they can make more informed decisions and solve business problems.

By subscribing to the Esri® Developer Network (EDNSM), you have access to the complete Esri geographic information system (GIS) software suite for developing and testing applications on every platform. Whether you're a desktop, mobile, server, or Web developer, EDN provides the tools you need to quickly and cost-effectively integrate mapping and GIS into your applications.

Subscribe to EDN and leverage the power of GIS to get more from your data. Visit www.esri.com/edn.



Getting Started with Windows Phone Development Tools

Joshua Partlow

Like many of you, I've spent the past year or so inundated by a slew of post-apocalyptic, Terminator-esque ads telling me what Droid Does. I've hummed along with the T-Mobile My Touch commercials. I've read articles about how many apps Apple has sold for the iPhone.

Unlike many of you, I've also spent the past year repeatedly telling friends and family that, yes, I do in fact work on a mobile phone, but no, none of those are the phone that I work on.

So it's fair to say that I was rather excited when Steve Ballmer and Joe Belfiore announced Windows Phone 7 at Mobile World Congress (MWC) in Barcelona, Spain. It was 6 a.m. Redmond time on Feb. 15, 2010, but I was sitting in a Microsoft conference room with a few hundred coworkers, some still in pajamas, anxiously waiting to watch a live feed of Belfiore demoing our new vision for the phone (microsoft.com/presspass/press/2010/feb10/02-15mwc10pr.mspx). I'll probably watch many more products launched in the years to come, but I'm

sure this will remain one of the highlights. It was certainly the most exciting moment I've had at Microsoft to date.

In an attempt to get you equally excited about the potential for Windows Mobile app development, over the course of this article I'm going to introduce you to the Windows Phone 7 application platform. Specifically, I'll talk about the basic components of the application platform, give you an overview of the Windows Phone Developer Tools, and walk you through the creation of a Windows Phone application that accesses a Web service.

A Break from the Past

With Windows Phone 7, Microsoft acknowledged that there's been a change in the mobile landscape. There's a new expectation that, beyond just allowing you to read e-mail and documents, a phone should also be an integral part of your life. Allowing you to do things like listen to music, share pictures and videos, and keep in touch with friends. Apparently, business users enjoy Facebook, too, and teenagers enjoy browsing the Web. It's not that shocking, I know, but it really does represent an evolution of the role a phone is expected to play in our lives.

As potential customers, I hope you saw Belfiore's demo and walked away ready to buy a phone. More importantly, as developers, I hope you came away curious about the application story. After all, creating a new OS is only part of competing in the crowded mobile market. If our premise is that users want a phone that enables them to merge their work and personal lives in new and exciting ways, then it's going to be a combination of the OS and an ecosystem of awesome applications that makes that possible.

I won't spend time describing the phone, because I'm sure you can find breakdowns on your favorite blog, but rather get to what I imagine are two bigger questions for you as developers: What's the Microsoft application platform for Windows Phone 7, and how do you get started building apps?

This article discusses a prerelease version of Windows Phone Developer Tools. All information is subject to change.

This article discusses:

- Windows Phone 7 application platform architecture
- Using Windows Phone Developer Tools
- Creating a Windows Phone application project
- Formatting the application bar
- Implementing auto-generated event handlers for the application bar buttons

Technologies discussed:

Windows Phone 7, Silverlight, XNA Framework, Visual Studio 2010 Express for Windows Phone

Code download available at:

code.msdn.microsoft.com/mag201010Phone

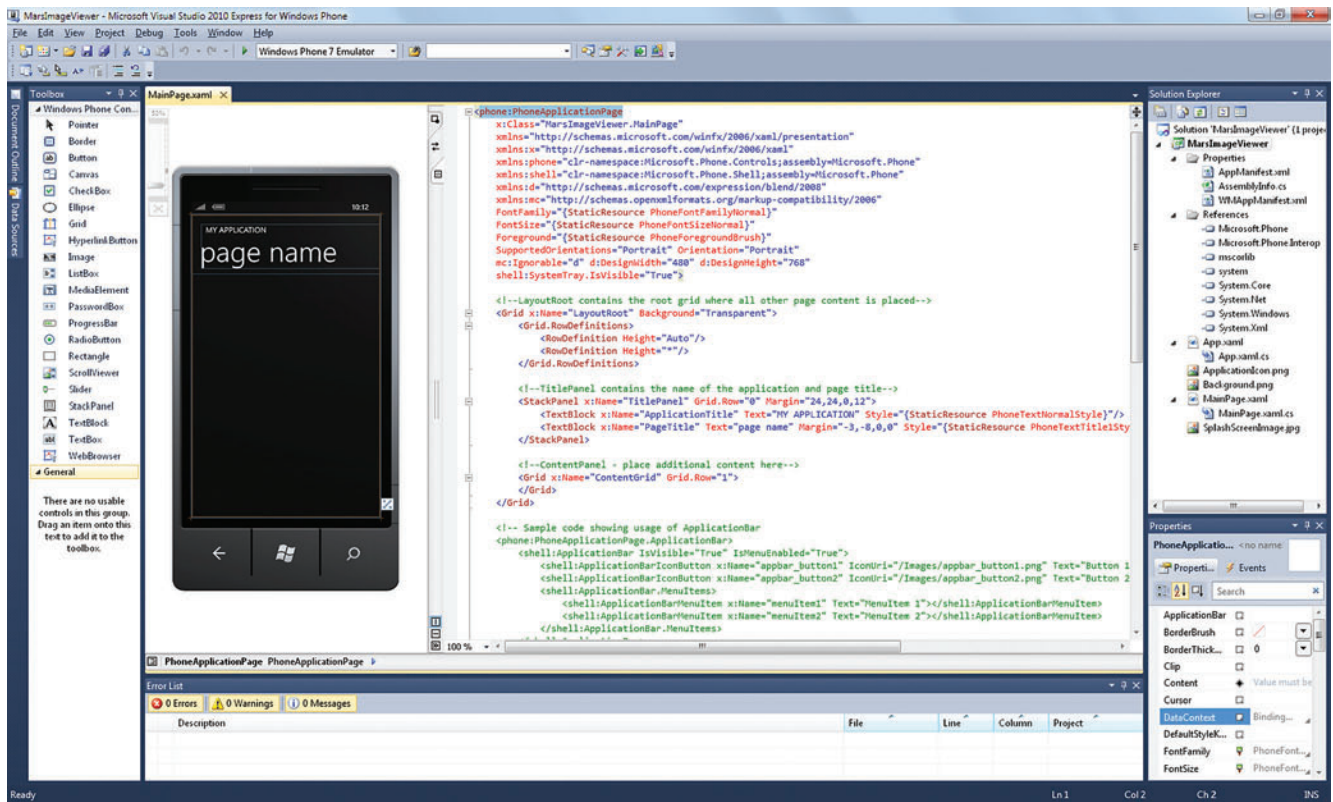


Figure 1 Your Initial Windows Phone Project in Visual Studio

A New World for Developers

Well, if MWC started the revelation of Windows Phone 7, then the Game Developers Conference (GDC) and MIX10 completed the story. At GDC and MIX, Microsoft announced that the Windows Phone 7 application platform is based on Silverlight and the XNA Framework. If you've developed for Windows Mobile in the past, then you know that this marks a fundamental divergence from previous versions of the OS.

So why change things up? Well, my opinion is that the strength of Microsoft lies in its broad suite of products, and that it's never more successful than when it ties them together. Prior to Windows Phone 7, the mobile development story might have leveraged some Microsoft assets, but not nearly to the extent you'll see now.

The XNA Framework was designed to allow developers to create complex and powerful 2D and 3D games that could span desktop, console and mobile spaces. It was designed around a broad set of hardware, from the graphics capabilities of the Xbox console to the sensor and touch capabilities of the Zune HD. What's more, it was also designed with networked gameplay in mind.

Silverlight is the Microsoft desktop platform for Web-based media and productivity applications. Here's a platform that, when combined with Expression Blend, is designed to enable the creation of compelling UIs that can work well standing alone or tied into Web services.

Both the XNA Framework and Silverlight are strong platforms in their own right. But when combined on the phone, they take things to a whole new level, allowing for the development of applications with beautiful UIs and stunning graphics that can easily leverage the capabilities of a mobile device.

As I experimented with Silverlight and XNA applications early in the Windows Phone 7 development cycle, I was quickly convinced that it was a good direction. I was new to both platforms, as well as to managed code, and the speed with which I could develop visually cool applications impressed me.

Another notable feature of the new application platform is the standardization of hardware and your programmatic access to it. Specifically, another announcement that came out of MIX was that Windows Phones would support a core set of hardware that developers could access in a consistent and reliable manner. Why is that important? Historically, it has been difficult to develop applications to run on multiple phones because you ended up having to create device-specific versions of your application. Some phones wouldn't run your application without significant rework, or wouldn't work at all because they didn't support features needed by your app.

With Windows Phone 7, there won't be a question of whether the phone your app is running on has support for location services—all of them will. There won't be a question of whether the phone has an accelerometer and whether you can access it—all of them will, and the access will be consistent. There won't be a question of whether your phone is touch-enabled—all of them will be, and again the access will be consistent. The list goes on, but I hope you get the idea.

With the Windows Phone 7 application platform, you'll be able to develop a single application and know that what you test on your phone will work on all of them. There will still be differentiation of hardware on Windows Phones, but there will also be a core foundation of hardware that your applications will be able to rely upon.

For more details on the Windows Phone hardware and the architecture of the application platform, you can refer to the application platform and hardware overviews on MSDN at [msdn.microsoft.com/library/ff402531\(v=VS.92\)](http://msdn.microsoft.com/library/ff402531(v=VS.92)) and [msdn.microsoft.com/library/ff637514\(v=VS.92\)](http://msdn.microsoft.com/library/ff637514(v=VS.92)).

Getting Started

OK, now that we've talked briefly about the application platform, let's talk about how you develop on it. Currently in beta, the Windows Phone Developer Tools are available for download from the Windows Phone Developer Portal (developer.windowsphone.com). If you don't have Visual Studio, the Developer Tools come with Visual Studio 2010 Express for Windows Phone. If you have Visual Studio 2010, the tools will integrate directly with it. Either way, you'll have everything you need to develop for Windows Phone 7.

The Developer Tools include a Windows Phone Emulator and Microsoft Expression Blend for Windows Phone. While not final, the beta is a good indication of what you'll have at launch. It's possible you might need to make some modifications to your code between the beta and final version of the tools, but those changes should be minor or at least well-documented.

I've installed the beta version of the tools, using Visual Studio 2010 Express for Windows Phone, and it's what I'll be using for my example. If you want a slightly simpler tutorial to start with, you can check out the Windows Phone Getting Started Guide on MSDN at [msdn.microsoft.com/library/ff402529\(v=VS.92\)](http://msdn.microsoft.com/library/ff402529(v=VS.92)).

So, what to create? Rather than do a general "hello world" or "make that flashlight" app I know you're dying to see, I thought I'd share a portion of a fun little project I've been working on.

At MIX I happened to attend a session on a Windows Azure project: Microsoft Codename "Dallas" (microsoft.com/windowsazure/dallas). Dallas is basically a marketplace for developers interested in getting data for their applications from Web services looking to sell it. Currently in its second community technology preview (CTP), you can utilize both the online portal and a number of free data sources provided as a trial to experiment with what the service has to offer. While the current list of providers isn't huge, there's still a bunch of stuff to play with. I personally found the image data provided by NASA on the Mars missions interesting and decided it would be cool to create a Windows Phone application that could browse through Mars rover pictures.

As a warning, the code included here and available for download is a working sample, but if you want to run it you'll need to register for the Dallas CTP to get an account key and user ID, which I've left blank in my code (microsoft.com/windowsazure/developers/dallas).

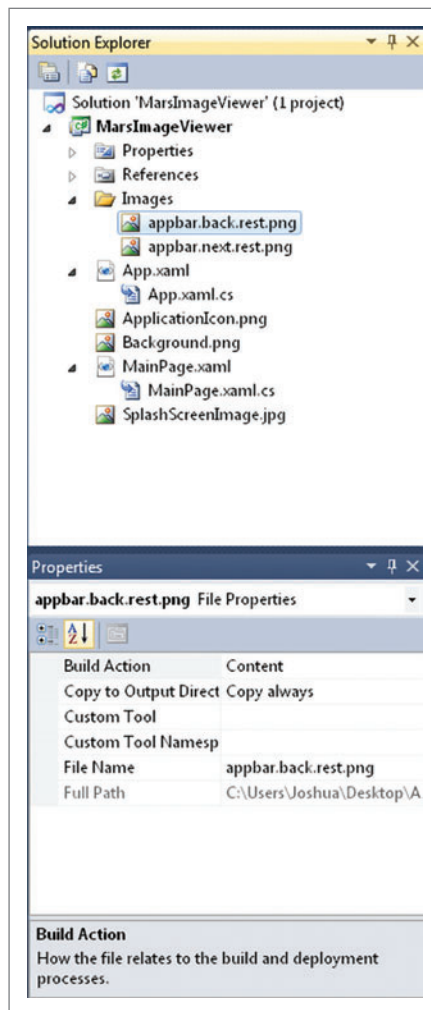


Figure 2 Adding Image Resources to the Project

Creating Your Project

The first step to creating the Mars rover image viewer is to create a Windows Phone application project. Launching the express version of Visual Studio 2010 for Windows Phone, you're presented with a standard Visual Studio Start Page. From there, you can select New Project, which will allow you to choose from a number of different project templates. If you've only installed the Windows Phone Developer Tools, your list will be limited to Silverlight for Windows Phone and XNA Game Studio 4.0.

I selected a Windows Phone app from the Silverlight templates and named my project MarsImageViewer. From there, Visual Studio does its magic and generates a project for you.

If you've worked with Windows Presentation Foundation (WPF) or Silverlight before, you shouldn't be all that surprised by what you get. You'll see that you're provided a design surface with a cool phone skin, a toolbox with some basic controls and a number of XAML files with their associated C# code-behind files (see **Figure 1**). If you care to learn about the exact differences between Silverlight and XNA on Windows and Silverlight and XNA on Windows Phone, you can reference the Windows Phone framework overview on MSDN at [msdn.microsoft.com/library/ff402528\(v=VS.92\)](http://msdn.microsoft.com/library/ff402528(v=VS.92)).

Understanding the XAML

As with Silverlight, the Windows Phone application template provides an App.xaml file and a MainPage.xaml file, which are the heart of your application. I won't spend time going over these in detail because they function in basically the same manner as their Silverlight counterparts, but I would like to point out two key differences before moving on to creating the application.

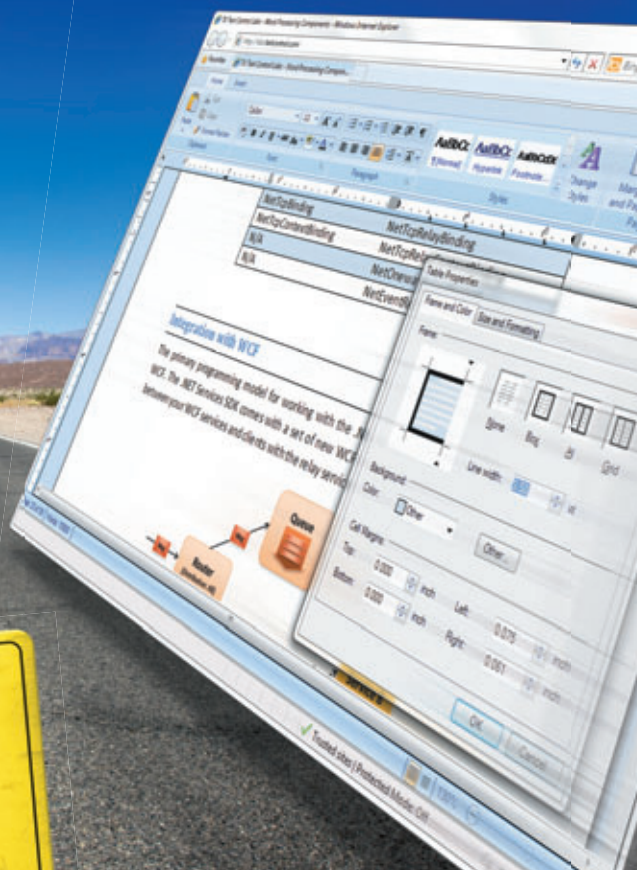
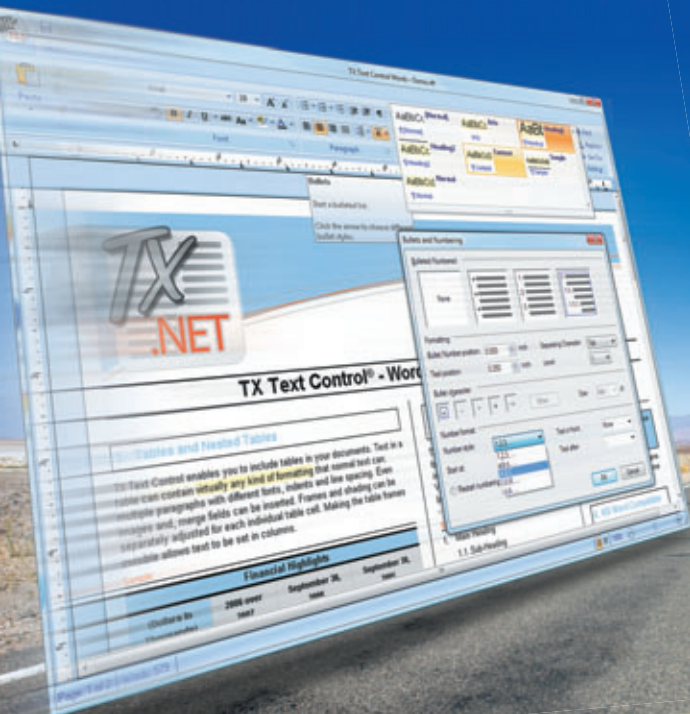
Figure 3 Configuring the App Bar

```
<phone:PhoneApplicationPage.ApplicationBar>
  <shell:ApplicationBar
    IsVisible="True" IsMenuEnabled="False">
    <shell:ApplicationBarIconButton
      x:Name="appbar_BackButton"
      IconUri="/Images/appbar.back.rest.png"
      Text="Back"
      Click="appbar_BackButton_Click">
    </shell:ApplicationBarIconButton>
    <shell:ApplicationBarIconButton
      x:Name="appbar_ForwardButton"
      IconUri="/Images/appbar.next.rest.png"
      Text="Next"
      Click="appbar_ForwardButton_Click">
    </shell:ApplicationBarIconButton>
  </shell:ApplicationBar>
</phone:PhoneApplicationPage.ApplicationBar>
```


WINDOWS FORMS / WPF / ASP.NET / ACTIVEX

WORD PROCESSING COMPONENTS

MILES BEYOND RICH TEXT



- ➔ TRUE WYSIWYG
- ➔ POWERFUL MAIL MERGE
- ➔ MS OFFICE NOT REQUIRED
- ➔ PDF, DOCX, DOC, RTF & HTML

MEET US AT
Microsoft
Visual Studio
CONNECTIONS
NOVEMBER 1-4, 2010, LAS VEGAS
BOOTH #514

TX
TEXT CONTROL[®]
word processing components

Word Processing Components
for Windows Forms & ASP.NET

WWW.TEXTCONTROL.COM

Microsoft
Visual Studio
PARTNER

TX Text Control Sales:

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

Figure 4 MainPage.xaml

```
<!--TitlePanel contains the name of the application and page title-->
<StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="24,24,0,12">
  <TextBlock x:Name="ApplicationTitle"
    Text="MarsImageViewer"
    Style="{StaticResource PhoneTextNormalStyle}"/>
  <TextBlock x:Name="PageTitle"
    Text="Images"
    Margin="-3,-8,0,0"
    Style="{StaticResource PhoneTextTitle1Style}"/>
</StackPanel>

<!--ContentPanel - place additional content here-->
<Grid x:Name="ContentGrid" Grid.Row="1">
  <Image Height="300"
    HorizontalAlignment="Left"
    Margin="36,104,0,0"
    Name="MarsImage"
    Stretch="Fill"
    VerticalAlignment="Top"
    Width="400" />
</Grid>
```

The first point of interest is in the default App.xaml file. You'll notice that while the majority of the auto-generated code is the same as you'd see in a desktop Silverlight project, there's an additional section that contains a PhoneApplicationService object:

```
<shell:PhoneApplicationService
  Launching="Application_Launching" Closing="Application_Closing"
  Activated="Application_Activated" Deactivated="Application_Deactivated"/>
```

With the Windows Phone Developer Tools beta, there's a new execution model that dictates the behavior of applications; this section of markup, combined with the code-behind in App.xaml.cs, is one place where that model surfaces. If you'd like to better understand the behavior of Windows Phone applications and this object, you can refer to the topic Execution Model for Windows Phone on MSDN at [msdn.microsoft.com/library/ff769557\(VS.92\)](http://msdn.microsoft.com/library/ff769557(VS.92)).

The second point of interest is in the MainPage.xaml file, and actually where I'll start creating the application. If you look closely, much of the file is similar to Silverlight, but there's a commented-out XAML section for an application bar. The application bar is a system control you can use to expose buttons and menu items. Not only does it save you the trouble of creating your own, its use lends consistency to the phone, as it appears and behaves exactly like the one used in the core phone applications. While I'll modify this template markup to create my application bar, you can also create one for a page using C# (see [msdn.microsoft.com/library/ff431786\(VS.92\)](http://msdn.microsoft.com/library/ff431786(VS.92)) for details).

Steal My App Bar

The first step to creating the application bar is to find the icons you want to use. You can create your own, or you can use some of the ones included with the developer tools. By default, the included icons can be found at C:\Program Files(x86)\Microsoft SDKs\Windows Phone\v7.0\Icons\ for 64-bit Windows and C:\Program Files\Microsoft SDKs\Windows Phone\v7.0\Icons\ for 32-bit Windows. They're definitely worth checking out because they match the look and feel of the phone.

Once you've picked the images you want, create an Images folder in your project and add the icons

to it. Then set the properties for each icon: Build Action should be "Content" and Copy to Output Directory should be "Copy always," as shown in Figure 2.

The next step is to uncomment the application bar markup and modify it for your application. In this case, that entails creating two buttons and their associated event handlers. Specifically, I created a button for retrieving the next photo and a button for retrieving the previous photo. I also added click events to the XAML and allowed Visual Studio to generate the event handlers. The application bar XAML should now look like the code in Figure 3.

Before moving to the code-behind, I also took the time to specify an application title and page name and to add an image control to the root grid of MainPage.xaml, as shown in Figure 4. This should leave the design surface looking like Figure 5, where you'll notice a blank app bar of four circles.

With the application interface defined, it's time to get some Mars rover images. The Dallas portal is designed to be user-friendly and allows you to experiment with queries. It also gives you the appropriate Web service URL, shows you how to add parameters and provides the appropriate header information for your queries.

What I discovered through the portal is that the NASA Web service allows queries for image information based on parameters or the retrieval of a specific JPEG image through an image ID. What that implies for this program is that two operations are required. The first is to query for image information, which includes the image ID. The second is to parse the returned XML for the image IDs, which can then be used to retrieve a specific image.

Calling Houston ... I mean, Dallas

So let's get started. In the default MainPage.xaml.cs file, I added using statements for three namespaces:

```
using System.Xml.Linq;
using System.IO;
using System.Windows.Media.Imaging;
```

Then I added a reference to the System.Xml.Linq DLL by right-clicking References in Solution Explorer, selecting Add Reference, selecting System.Xml.Linq and then clicking OK. System.Xml.Linq

provides access to classes that aid in loading XML from streams and then querying that XML through LINQ. Don't worry if you're not familiar with LINQ; this example uses a minimal amount of LINQ to XML and you can always refer to MSDN for more information.

I also created two private variables for the page. An IEnumerable of XElement objects called entries to store the result of a LINQ to XML query, and an integer index to keep track of the picture I'm on. I then modified the MainPage constructor to initialize the index to 0 and call a getImageIDs function:

```
private IEnumerable<XElement> entries;
private int index;
```

```
// Constructor
public MainPage() {
  InitializeComponent();
```

```
  index = 0;
  getImageIDs();
}
```

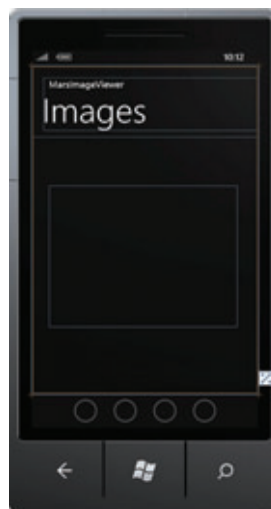


Figure 5 The Configured Design Surface

Does your Team do more than just track bugs?

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

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



Alexsys Team

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

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

Native Smart Card Login Support including Government and DOD



New in Team 2.11

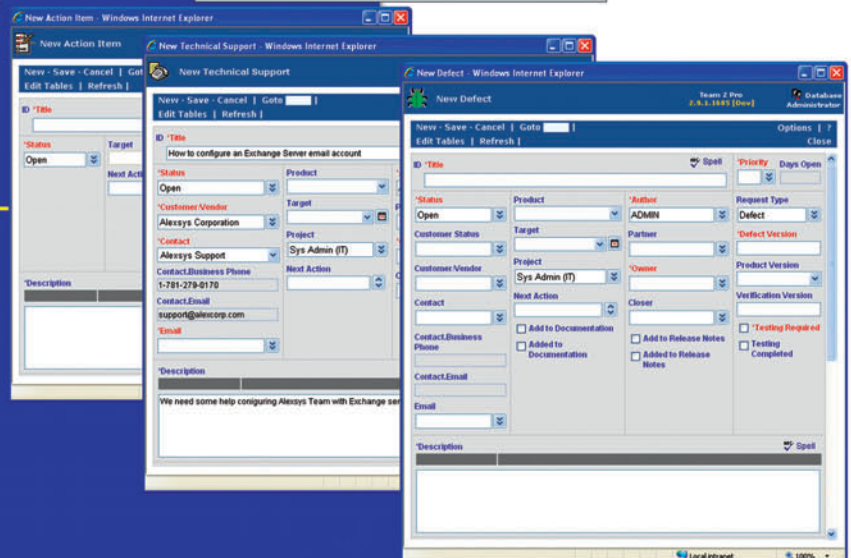
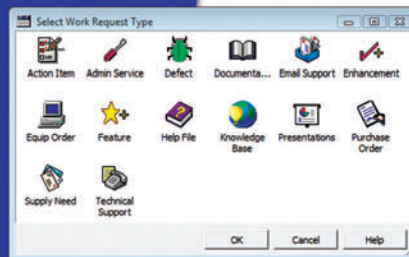
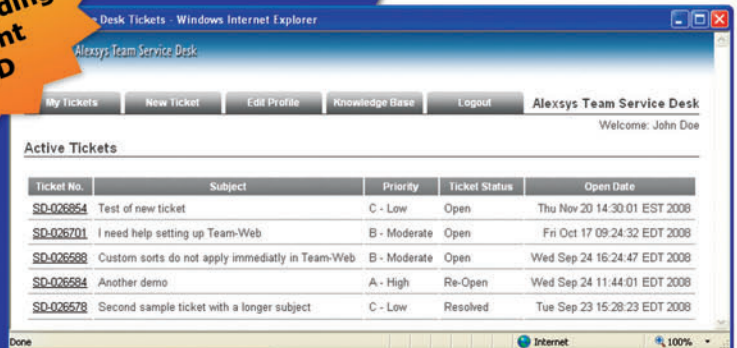
- Full Windows 7 Support
- Windows Single Sign-on
- System Audit Log
- Trend Analysis
- Alternate Display Fields for Data Normalization
- Lookup Table Filters
- XML Export
- Network Optimized for Enterprise Deployment

Service Desk Features

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

Team 2 Features

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



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

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

Figure 6 Retrieving the Image

```
private void getImage(string ID) {
    Uri serviceUri = new Uri(
        "https://api.sqlazureservices.com/NasaService.svc/MER/Images/" +
        ID + "?format=raw");
    WebClient imgDownloader = new WebClient();
    imgDownloader.Headers["$accountKey"] = "<Your account key>";
    imgDownloader.Headers["$uniqueUserID"] = "<Your user ID>";
    imgDownloader.OpenReadCompleted +=
        new OpenReadCompletedEventHandler(imgDownloader_OpenReadCompleted);
    imgDownloader.OpenReadAsync(serviceUri);
}

private void imgDownloader_OpenReadCompleted(
    object sender, OpenReadCompletedEventArgs e) {
    if (e.Error == null) {
        Stream imageStream = e.Result;
        BitmapImage imgsrc = new BitmapImage();
        imgsrc.SetSource(imageStream);
        MarsImage.Source = imgsrc;
    }
}
```

The `getImageIDs` function is designed to start the retrieval of image information from the Web service. The function uses the Web service URL and a `WebClient` to start an asynchronous request for the image information:

```
private void getImageIDs() {
    Uri serviceUri = new Uri("https://api.sqlazureservices.com/
    NasaService.svc/MER/Images?missionId=1&format=raw");
    WebClient recDownloader = new WebClient();
    recDownloader.Headers["$accountKey"] = "<Your account key>";
    recDownloader.Headers["$uniqueUserID"] = "<Your user ID>";
    recDownloader.OpenReadCompleted +=
        new OpenReadCompletedEventHandler(recDownloader_OpenReadCompleted);
    recDownloader.OpenReadAsync(serviceUri);
}
```

You'll notice that, for simplicity, I've hardcoded the `missionId` parameter to 1, which in this case represents the Opportunity mission. Ideally, this parameter and others would be dynamically defined by the user.

With any asynchronous request, you need a handler. This handler is called when the request for data completes. It then uses the returned stream along with some basic LINQ to XML to access all the "entry" tags in the returned XML; "entry" being the opening tag for each image record:

```
private void recDownloader_OpenReadCompleted(
    object sender, OpenReadCompletedEventArgs e) {
    if (e.Error == null) {
        Stream responseStream = e.Result;
        XNamespace ns = "http://www.w3.org/2005/Atom";
        XElement marsStuff = XElement.Load(responseStream);
        entries = marsStuff.Elements(ns + "entry");
        string imageID =
            (string)entries.ElementAt<XElement>(index).Element(
                ns + "title").Value;
        getImage(imageID);
    }
}
```

The resulting collection is stored to the `IEnumerable` of `XElement` objects (entries), which I declared earlier. With a final bit of LINQ to XML, the handler then retrieves the value of the title tag for the first `XElement` in entries. The value of the title tag in this XML schema happens to correspond to the image ID, which is then passed into a `getImage` function.

The `getImage` function is similar to the `getImageIDs` function. The only difference is the Web service URL that's used. This function asynchronously retrieves a stream to the image identified by the

ID parameter. Its handler then uses that stream to set the source of the picture control I defined in `MainPage.xaml` (see Figure 6).

Buttoning up Buttons

At this point, the rest of the application is pretty simple, with only the auto-generated event handlers for the application bar buttons remaining to be implemented. These are the buttons that will be used to advance forward and backward through the Mars rover pictures. As you can see, I basically just reused the `getImage` function and added some logic to handle changing the index of the current record in the entries collection. Here's the handler for the back button:

```
private void appbar_BackButton_Click(
    object sender, EventArgs e) {
    if (index > 0) {
        index--;
        XNamespace ns = "http://www.w3.org/2005/Atom";
        string imageID = (string)entries.ElementAt<
            XElement>(index).Element(ns + "title").Value;
        getImage(imageID);
    }
}
```

The forward button handler is pretty much the same, except for its indexing:

```
if ((index + 1) < entries.Count<XElement>()) {
    index++;
    ...
}
```

You can now run the program by using the included Windows emulator. Select Windows Phone 7 Emulator from the target device menu on the Standard toolbar. Press F5 and the program is built and deployed to the emulator (see Figure 7).

Ready for Launch

This sample was relatively simple, but I hope it's given you a rough idea of what the developer tools look like and how easy it is to pull together an application on Windows Phone. There are a lot of possibilities with Windows Phone 7, and you should take the time to explore them more.

What you've seen here is only a small introductory glimpse of what the application platform has to offer. As proof of that, consider the simple application I laid out. With one more button and roughly 12 lines of code, you could use the `MediaLibrary` class in

the `Microsoft.Xna.Framework.Media` namespace to save a given photo to the media library (see [msdn.microsoft.com/library/ff769549\(v=VS.92\)](http://msdn.microsoft.com/library/ff769549(v=VS.92))).

Yes, that's right—you can use XNA APIs from within your Windows Phone Silverlight-based applications. Unfortunately, discussions about the interweaving of Silverlight and XNA APIs in apps, along with much more, will have to wait for a later date. So watch out for some deeper, more-targeted articles and check out the documentation and samples on MSDN at [msdn.microsoft.com/library/ff402535\(v=VS.92\)](http://msdn.microsoft.com/library/ff402535(v=VS.92)). ■

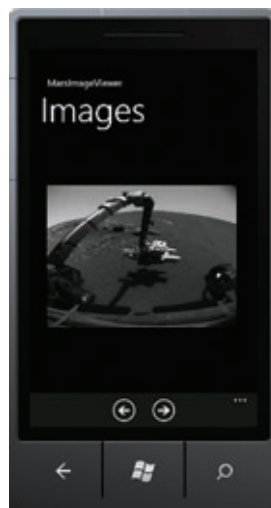


Figure 7 Running the App in the Emulator

JOSHUA PARTLOW is a programming writer on the Windows Phone 7 team. He works on documenting the phone bring-up process, device driver development and application development for OEMs creating Windows Phones.

THANKS to the following technical expert for reviewing this article: Windows Phone 7 Team

Speed • Reliability • Precision • Agility

DynamicPDF...Proven .NET Components for Real-Time PDFs



- Easy-to-use • Highly efficient
- Industry leading support • Huge feature set

Try it FREE today!

Experience our fully functional, never expiring evaluation and community editions.

DynamicPDF Suite v6.0 for .NET

Our easy-to-use tools integrate with ASP.NET and ADO.NET allowing for the quick, real-time generation of PDF documents and reports.

For easy maintenance and deployment, our most popular tools are now available as a bundled suite.



Layout reports in DynamicPDF Designer with its Visual Studio look and feel.

DynamicPDF Generator v6.0 for .NET

- Linearize/Fast Web View • JavaScript
- Encryption/Security • PDF/X-1a • PDF/A-1a/b
- Create Tagged PDFs • Interactive Form Fields
- Over 50 Ready-To-Use Page Elements Including 22 Bar Codes and Charting • Digital Signatures



DynamicPDF Merger v6.0 for .NET

- Merge • Stamp • Append • Split • Password/Security
- Form-Fill • Outline and Annotation Preservation
- Place, Rotate, Scale and Clip Pages • Decryption



DynamicPDF ReportWriter v6.0 for .NET

- GUI Report Designer • Event Driven • Recursive Sub-Reports
- Use PDF Templates • Automatic Pagination • Placeholders
- Record Splitting and Expansion • Column Support
- Full DynamicPDF Merger and Generator Integration



Also, check out our newest product, **DynamicPDF PrintManager for .NET**.

ceTesoftware
INFINITE POSSIBILITIES

ceTe Software has been delivering quality software applications and components to our customers for over 10 years. Our DynamicPDF product line has proven our commitment to delivering innovative software components and our ability to respond to the changing needs of software developers. We back our products with a first class support team trained to provide timely, accurate and thorough responses to any support needs.

www.cete.com
info@cete.com

800.631.5006
+1 410.772.8620

VISUAL STUDIO LIVE! FOCUS TOPICS:

Silverlight/WPF	Web	Visual Studio 2010 / .NET 4	SharePoint	Cloud Computing	Data Management
-----------------	-----	-----------------------------	------------	-----------------	-----------------

VISUAL STUDIO LIVE! PRE-CONFERENCE WORKSHOPS: SUNDAY, NOVEMBER 14, 2010 (SEPARATE ENTRY FEE REQUIRED)

Making Effective Use of Silverlight and WPF <i>Billy Hollis & Rockford Lhotka</i>	SQL Server 2008 and 2008 R2 for Developer <i>Andrew Brust & Leonard Lobel</i>	Better Process and Tools with TFS 2010 <i>Brian Randell</i>
--	--	--

VISUAL STUDIO LIVE! DAY 1: MONDAY, NOVEMBER 15, 2010

Getting started in Silverlight – A Jumpstart to Productivity <i>Tim Huckaby</i>	What's new in ASP.NET 4 WebForms - <i>Wallace McClure</i>	So Many Choices, So Little Time: Understanding Your .NET 4.0 Data Access Options <i>Leonard Lobel</i>	Building Business Applications with Visual Studio LightSwitch <i>Orville McDonald</i>
Transitioning from Windows Forms to WPF <i>Miguel Castro</i>	ASP.NET MVC Quick Primer <i>Gus Emery</i>	Best Kept Secrets in Visual Studio 2010 and .NET 4.0 <i>Deborah Kurata</i>	Overview of .NET 4
WPF & Silverlight: Data Visualization, NUI, and Next Generation of User Experience <i>Tim Huckaby</i>	What's this WebMatrix anyway? <i>Gus Emery</i>	What's New in the .NET 4.0 BCL <i>Jason Bock</i>	C# 4.0 Language Features <i>Alexandru Ghiondea</i>
Easing into Windows Phone 7 Development <i>Walt Ritscher</i>	ASP.NET "Razor" <i>Miguel Castro</i>	What's New in Visual Studio 2010 Debugging <i>Brian Peek</i>	Tips & Tricks: Visual Studio 2010 IDE & Extensions <i>Chris Granger</i>

DINE AROUND ORLANDO

VISUAL STUDIO LIVE! DAY 2: TUESDAY, NOVEMBER 16, 2010

CSLA 4, Silverlight and WPF <i>Rockford Lhotka</i>	AJAX with the UpdatePanel, WebForms, and the AJAX Control ToolkitX <i>Wallace McClure</i>	New IDE and Languages Features in VS 2010 using VB and C# <i>Ken Getz</i>	Visual Basic 2010 Overview
Silverlight, WCF RIA Services, and Your Business Objects <i>Deborah Kurata</i>	Leveraging Client Capabilities with jQuery in Visual Studio and ASP.NET <i>Robert Boedigheimer</i>	Windows Server AppFabric Caching <i>Jon Flanders</i>	Designing & Developing for the Rich Mobile Web <i>Joe Marini</i>
Digging Deeper into Windows Phone 7 <i>Walt Ritscher</i>	jQuery for ASP.NET Developers - Part 1 <i>Jeffrey McManus</i>	Building RESTful Services Using Windows Communication Foundation <i>Jon Flanders</i>	Pragmatic .NET Development with CodeFluent Entities <i>Omid Bayani</i>
Bind Anything to Anything in Silverlight and WPF <i>Ken Getz</i>	jQuery for ASP.NET Developers - Part 2 <i>Jeffrey McManus</i>	Building RESTful Applications with the Open Data Protocol <i>Todd Anglin</i>	Windows Phone 7 and Azure: Enhancing the User Experience with the Cloud <i>Danilo Diaz</i>
Gentle Introduction to 2D Animation in SL/WPF <i>Ken Getz</i>	ASP.NET Request Processing <i>Robert Boedigheimer</i>	Can you use MEF too much? Hint - NO! <i>Jon Flanders</i>	Sensors, Locations, Notifications Oh My; Advance Topic in Windows Phone 7 <i>Danilo Diaz</i>

VISUAL STUDIO LIVE! AFTER DARK WITH DEVOPALOOZA

VISUAL STUDIO LIVE! DAY 3: WEDNESDAY, NOVEMBER 17, 2010

Generating Dynamic UI in WPF 4 and Silverlight 4 <i>Billy Hollis</i>	Azure Platform Overview <i>Vishwas Lele</i>	Why Software Sucks <i>David Platt</i>	Introduction to SharePoint Development with Visual Studio 2010 <i>Paul Yuknewicz</i>
Design, Don't Decorate <i>Billy Hollis</i>	Architecting for Azure <i>Vishwas Lele</i>	Building a Testable Data Access Layer <i>Todd Anglin</i>	Creating Extensions for the Visual Studio 2010 SharePoint Tools <i>Jon Flanders</i>
Using Microsoft Prism – Loose Coupling for Application Development <i>David Platt</i>	Developing an Azure based Monte Carlo Simulator <i>Vishwas Lele</i>	What's New in VS 2010 for TFS? <i>Brian Randell</i>	PowerPivot and Excel Services <i>Andrew Brust</i>
Multi-touch Madness! <i>Brian Peek</i>	What is Microsoft Dallas? <i>Michael Stiefel</i>	Reserved for Late-Breaking Content	Application Lifecycle Management for Developers in SharePoint 2010 <i>Paul Yuknewicz</i>
Using WPF for Good and Not Evil <i>David Platt</i>	How do you decide between Relational Database or Table Storage in the Cloud? <i>Michael Stiefel</i>	VS10 ALM Features <i>Brian Randell</i>	SharePoint for ASP.NET Developers <i>Jon Flanders</i>

**VISIT US ONLINE AT VSLIVE.COM/ORLANDO
FOR DETAILS ON SESSIONS, SPEAKERS, AND MORE!**

Platinum Sponsor:



Supported by:



NOVEMBER 14-17, 2010

ORLANDO FLORIDA

HILTON IN THE WALT DISNEY WORLD RESORT

Visual Studio® **LIVE!**

EXPERT SOLUTIONS FOR .NET DEVELOPERS

FOUR DAYS THAT WILL ROCK YOUR CODE

ARE YOU READY to take your code to the next level? Expand your skillset and maximize the development capabilities of Visual Studio and .NET 4 during the four action-packed days of Visual Studio Live! Orlando!

Hard-hitting, real-world training on:

- **VISUAL STUDIO 2010/.NET 4**
- **SILVERLIGHT/WPF**
- **SHAREPOINT**
- **WEB**
- **CLOUD COMPUTING**
- **DATA MANAGEMENT**



REGISTER BY OCTOBER 20TH AND SAVE \$200!

© VSLIVE.COM/ORLANDO

USE PRIORITY CODE MSDOCT

Scalable Multithreaded Programming with Thread Pools

Ron Fosner

Programming is getting more challenging, particularly if you're working in a field that requires you to tune your application for the fastest possible throughput. One contributing factor is that the last few years have seen a change in the way PCs are evolving. Instead of relying on the ever-increasing speed of a single processor, the computational power of PCs is now being distributed across multiple cores. This is a good thing. Hefty increases in latent processing power are now available at relatively low cost, and often with much lower power consumption and cooling needs.

But there's a downside to the increasing prevalence of multiple-core systems. To use multiple processors, you need to delve into the world of parallel processing. That means it takes more work—and sometimes a significant learning curve—for programmers to take advantage of that latent processing power. It's possible to throw a few compiler hints into your project and get the compiler to write some multithreaded code for you. However, to take advantage of the full power of multicore PCs, you'll need to make some changes in the way that you program large jobs.

There are many different ways to distribute your work across multiple cores. One of the easiest and most robust is called task-based programming. Tasks allow you to take your application's work and spread it across some or all of the CPU cores that are available. With a bit of thoughtful programming, you can minimize or even eliminate any data-dependency or time-synchronization constraints. To achieve this state of multicore bliss, you'll have to reexamine some of your preconceptions of how to attack a programming problem and rethink them in terms of task-based programming.

To show you how this can work, I'll walk you through the steps—and the missteps—I took in converting a single-threaded application to one that can scale up to using all of the CPUs available in a computer. In this article, I'll present some of the concepts of multithreaded programming and show you some simple ways to introduce threaded execution into your code with OpenMP and thread pools. You'll also see how to use Visual Studio 2010 to measure the improvement in performance you gained from these techniques. In a future article, I'll build on that foundation to show you more sophisticated multithreaded executing with tasks.

This article discusses:

- Multithreading basics
- Using OpenMP
- Using thread pools
- Measuring core utilization

Technologies discussed:

Visual Studio 2010

Code download available at:

code.msdn.microsoft.com/mag201010Thread

From Threads to Tasks

The major challenge with making a program scale to the number of CPU cores is that you can't just decide to throw some jobs onto their own threads and let them run. In fact, this is what a lot of folks do, but it only scales well to the number of cores for which the app was designed. It doesn't scale well with fewer or more than the number of cores that were targeted, and it totally overlooks the built-in obsolescence that such an approach engenders.

A better way to make sure that your application scales well with a varying number of cores is to break larger jobs into smaller, thread-friendly sub-jobs called tasks. The most challenging part of

converting a monolithic single-threaded program or a program that has a few dedicated threads into a task-based job system is actually breaking your jobs into tasks.

There are a few guidelines to keep in mind when converting a large single-threaded job into multithreaded tasks:

- The large job can be arbitrarily divided into 1 to n tasks.
- The tasks should be able to run in any order.
- The tasks should be independent of each other.
- The tasks must have associated context data.

If all these were easy, you'd have no problem making your applications run on any number of cores. Unfortunately not all problems can be broken so neatly into tasks that meet these guidelines.

These guidelines are important because, if followed, they enable you to run each task independently on a thread, with no dependence between tasks. Ideally tasks should be able to be run in any order, so eliminating or reducing interdependencies is paramount to getting a task-based system up and running.

Most real-world programs will go through various stages of processing, with each stage required to complete prior to starting the next stage. These synchronization points are often unavoidable, but with a task-based system the goal is to make sure you take advantage of whatever CPU power is immediately available. With some judicious breaking of large jobs into smaller ones, it's often possible to start intermingling some completed task results with their next stage of processing while some of the initial tasks are still running.

Simple Multithreading with OpenMP

Before converting your entire application to use tasks, you should be aware of other ways to get some the benefits of multithreading without going through the rigorous exercise of making everything a task. There are numerous ways to incorporate multithreading into your application—many that require little actual effort—but that allow you to benefit from adding multithreading to your code.

OpenMP is one of the simplest ways to add parallel processing to your programs and has been supported by the Visual Studio C++ compiler since 2005. You enable OpenMP by adding pragmas to your code that indicate where and what kind of parallelism you'd like to add. For example, you can add parallelism to a simple Hello World program:

```
#include <omp.h> // You need this or it won't work
#include <stdio.h>
int main (int argc, char *argv[]) {
    #pragma omp parallel
    printf("Hello World from thread %d on processor %d\n",
        ::GetCurrentThreadId(),
        ::GetCurrentProcessorNumber());
    return 0;
}
```

The OpenMP pragma parallelizes the next block of code—in this case it's just the printf—and runs it simultaneously across all hardware threads. The number of threads will depend on how many hardware threads are available on the machine. The output is a printf statement running on each hardware thread.

To get any OpenMP program to parallelize (and not silently ignore your OpenMP pragmas), you need to enable OpenMP for your program. First, you have to include the /openmp compiler option (Properties | C/C++ | Language | Open MP Support). Second, you need to include the omp.h header file.

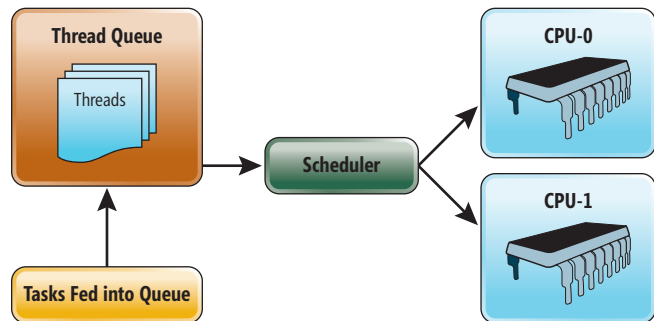


Figure 1 A Thread Pool

Where OpenMP really shines is when your application spends most of its time looping over functions or data and you want to add multiprocessor support. For example, if you have a for loop that takes a while to execute, you can use OpenMP to easily parallelize the loop. Here's an example that automatically breaks up array calculations and distributes them across the number of cores that are currently available:

```
#pragma omp parallel for
for (int i = 0; i < 50000; i++)
    array[i] = i * i;
```

OpenMP has other constructs that give you more control over the number of threads created, whether the distributed work needs to finish before the next block of code is executed, creating thread-local data, synchronization points, critical sections and so on.

As you can see, OpenMP is an easy way to gently introduce parallelism into an existing code base. However, while the simplicity of OpenMP is attractive, there are times you need more control over what your application is doing, such as when you want the program to dynamically adjust what it's doing or you need to make sure that a thread stays on a particular core. OpenMP is designed to easily

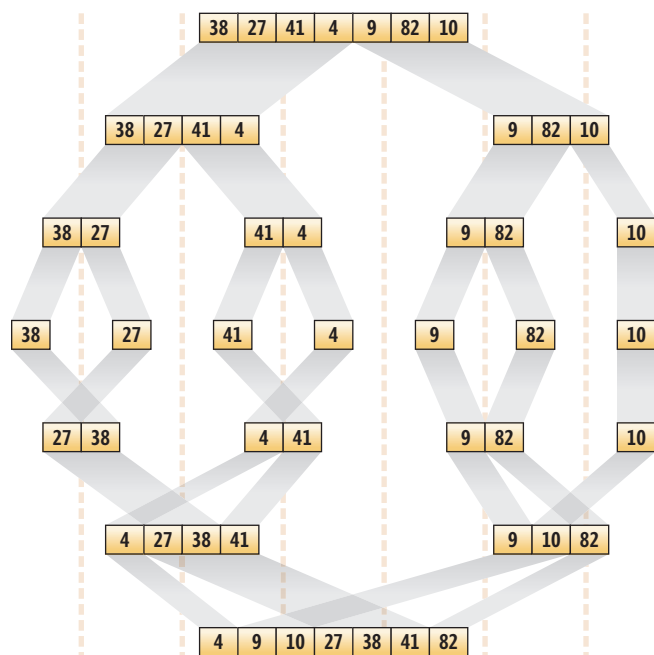


Figure 2 Sorting a String of Random Integers

Figure 3 Quick Sort

```
using System;
using System.Threading;
using System.Threading.Tasks;
using System.Diagnostics;

namespace ParallelSort {
    class Program {
        // For small arrays, use Insertion Sort
        private static void InsertionSort(
            int[] list, int left, int right) {

            for (int i = left; i < right; i++) {
                int temp = list[i];
                int j = i;

                while ((j > 0) && (list[j - 1] > temp)) {
                    list[j] = list[j - 1];
                    j = j - 1;
                }
                list[j] = temp;
            }
        }

        private static int Partition(
            int[] array, int i, int j) {

            int pivot = array[i];

            while (i < j) {
                while (array[j] >= pivot && i < j) {
                    j--;
                }

                if (i < j) {
                    array[i++] = array[j];
                }

                while (array[i] <= pivot && i < j) {
                    i++;
                }

                if (i < j) {
                    array[j--] = array[i];
                }
            }

            array[i] = pivot;
            return i;
        }

        static void QuickSort(
            int[] array, int left, int right) {

            // Single or 0 elements are already sorted
            if (left >= right)
                return;

            // For small arrays, use a faster serial routine
            if (right - left <= 32) {
                InsertionSort(array, left, right);
                return;
            }

            // Select a pivot, then quicksort each sub-array
            int pivot = Partition(array, left, right);

            QuickSort(array, left, pivot - 1);
            QuickSort(array, pivot + 1, right);
        }

        static void Main(string[] args) {

            const int ArraySize = 50000000;

            for (int iters = 0; iters < 1; iters++) {
                int[] array;
                Stopwatch stopwatch;

                array = new int[ArraySize];
                Random random1 = new Random(5);

                for (int i = 0; i < array.Length; ++i) {
                    array[i] = random1.Next();
                }

                stopwatch = Stopwatch.StartNew();
                QuickSort(array, 0, array.Length - 1);
                stopwatch.Stop();

                // Verify it is sorted
                for (int i = 1; i < array.Length; ++i)
                    if (array[i - 1] > array[i - 1])
                        throw new ApplicationException("Sort Failed");

                Console.WriteLine("Serialt: {0} ms",
                    stopwatch.ElapsedMilliseconds);
            }
        }
    }
}
```

integrate some aspects of multithreaded programming into your program, but it lacks some of the advanced features you'll find you require to get optimal use of out multiple cores. This is where tasks and thread pools come in.

Using the Thread Pool

Threads require a lot of bookkeeping by the OS, so it's not a good idea to wantonly create and destroy them. There are not-insignificant costs associated with creating and destroying a thread, so if you do this constantly, it's easy to lose any advantage you gain by multithreading.

Instead, it's best to use an existing set of threads, recycled as needed, for all of your threaded activity. This design is called a thread pool, and Windows provides one for you to use. Using this thread pool relieves you of having to deal with thread creation, destruction, and management, all of which is handled for you by the thread pool. OpenMP uses a thread pool to distribute work with threads, and both Windows Vista and Windows 7 provide optimized versions of the thread pool for you to use directly.

While it's entirely possible to create your own thread pool—which you might need to do if you've got some unusual scheduling requirements—you're much better off using one provided by the OS or the Microsoft .NET Framework.

At this point I need to clarify some terminology. When most people talk about a thread, they're referring to the flow of execution through a single CPU core—in other words, a software thread. On a CPU, the flow of execution (the actual execution of instructions) occurs on hardware threads. The number of hardware threads is limited by the hardware your application is running on. In the old days, this used to be a single-threaded CPU, but now it's common to find systems that have at least dual-core processors. A four-core CPU will have the ability to run four hardware threads, or eight if it's hyperthreaded. High-end desktop systems boast as much as 16 hardware threads, and some server configurations have more than 100!

While a hardware thread actually runs the instructions, a software thread refers to the entire context—register values, handles, security attributes and so on—required to actually execute the job on a hard-

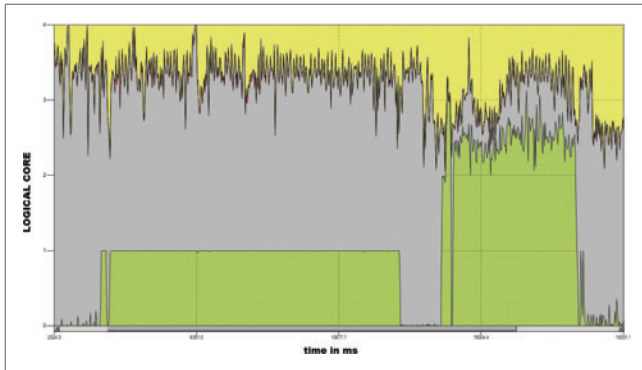


Figure 4 Core Utilization

ware thread. It's important to note that you can have many more software threads than hardware threads, and this is the underlying basis of a thread pool. It allows you to queue up tasks with software threads and then schedule them to execute on the actual hardware threads.

The advantage of using a thread pool instead of creating your own threads is that the OS will take care of scheduling tasks for you—your job will be to keep feeding tasks into the thread pool so that the OS will be able to keep all of the hardware threads busy. This is illustrated in **Figure 1**. Everything in the box makes up the thread pool and is out of the realm of the programmer. It's up to the application to feed tasks into the thread pool, where they are placed into the thread queue and eventually scheduled to run on a hardware thread.

Now you get to the hard part: What's the best way to structure jobs to keep the cores busy and the CPU utilization at its maximum? This depends on what your application needs to do.

I frequently work with video game companies and these are some of the most challenging types of applications to work with because there's a lot of work to be done—usually some in a particular serial order—and it's sensitive to delays. There's usually a certain frame rate at which the program updates, so if the frame rates start to fall behind, the game experience suffers. As a result, there's a lot of incentive to maximize use of the hardware.

On the other hand, if your application does one large thing at a time, it's a little more obvious what you need to do, but it's still challenging to try to distribute a single job across multiple cores.

Multithreaded Sorting

First let's take a look at a monolithic job that's frequently found in applications and see how you can go about transforming it into a more multithread-friendly form. I'm thinking, of course, about sorting. Sorting is a particularly good example because it's got

one major obstacle: How do you sort something and spread out the sorting across multiple cores so that the sorting on one core is independent from what's being sorted on another core?

A naïve approach frequently seen is to lock access to any data that can be accessed by multiple cores using something like a mutex, semaphore or critical section. This will work. However, if it's used as a panacea for not correctly scheduling access to shared data, at best you'll end up killing any gains you might have achieved by blocking other threads from executing. At worst, you might introduce some subtle race condition that will be excruciatingly difficult to track down.

Luckily, you can design the application to eliminate most of the shared access to the data across threads by choosing the appropriate sorting algorithm.

A better approach is to give each core a subsection of the array to sort. This divide-and-conquer method is an easy way to distribute work on the same data set across multiple cores. Algorithms such as merge sort and quick sort work from a divide-and-conquer strategy and are straightforward to implement in a way that takes advantage of a multicore system.

Let's look at how merge sort works on the string of random integers shown in **Figure 2**. The first step is to choose a midpoint in the array and divide it into two sublists. Keep dividing until you have lists that are zero or one element long.

In most implementations, there's actually a list size limit below which you have an efficient algorithm designed for small lists, but it also works if you just keep dividing until you can't divide any more. The important thing to note is that once you divide a list into two sublists, the lists are independent. This is illustrated by the dashed red lines in **Figure 2**. Once you divide the list into sublists, each sublist is independent and you can give each one to a CPU to manipulate as it likes without having to lock anything.

To make the sorting as efficient as possible, choose an algorithm that will take each sublist and sort it in place. This is important not only to prevent unnecessary copying of the data, but also to keep the data warm in the CPU's L2 cache. As you strive to write more and more efficient parallel code, you eventually have to be aware of how data gets swapped into and out of the L2 cache, which is generally about 256KB in most modern processors.

There are many sorting algorithms that lend themselves to parallelization. Quick sort, selection sort, merge sort, and radix sort are all algorithms that subdivide the data and operate on it independently. So let's take a look at a serial implementation of a sorting routine and convert it into a parallel one.

In theory, once you keep subdividing an array recursively you will eventually end up with a single element. At this point, there's

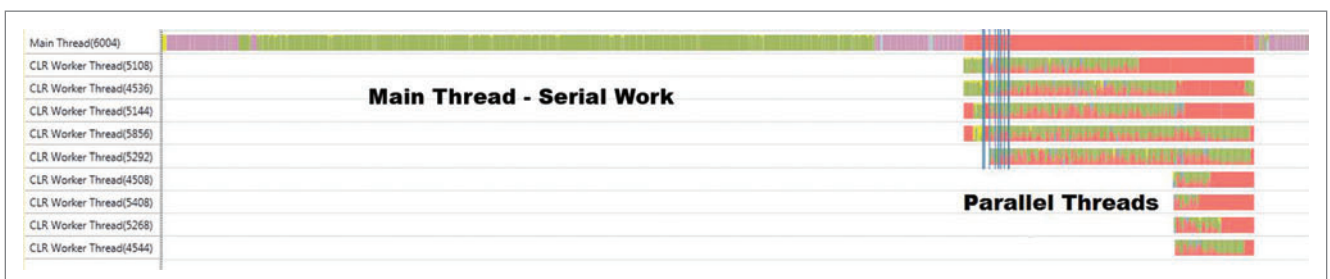


Figure 5 Thread Work

nothing to sort, so the algorithm moves onto the next step, which is merging sorted sublists together. The individual elements are merged into larger, sorted lists. These sorted sublists are then merged into even larger sorted lists until you have the original array in a sorted order. As mentioned previously, it's usually faster to switch to an algorithm that's optimized to sort small lists when the list size reaches a certain threshold.

There are many ways to write a sorting algorithm, and I've chosen to write a simple quick sort routine in C#, as shown in **Figure 3**. This program fills a large array with the same sequence of random numbers and then sorts them using a quick sort routine, reporting how long it took.

If you look at the `QuickSort` function, you'll see that it recursively divides the array in two until a threshold is reached, at which point it sorts the list without further subdivision. If you change this to a parallel version, all you have to do is change these two lines:

```
QuickSort( array, lo, pivot - 1);  
QuickSort( array, pivot + 1, hi);
```

The parallelized version is:

```
Parallel.Invoke(  
    delegate { QuickSort(array, left, pivot - 1); },  
    delegate { QuickSort(array, pivot + 1, right); }  
);
```

The `Parallel.Invoke` interface is part of the `System.Threading.Tasks` namespace found in the .NET Task Parallel Library. It allows you to specify a function to be run asynchronously. In this case, I tell it that I want to run each sorting function on a separate thread.

While it would be more efficient to spawn only one new thread and to use the current thread of execution to sort the other sublist, I wanted to maintain symmetry and illustrate how easy it is to convert a serial program into a parallel program.

Core Utilization

The next obvious question is: Has this parallelization improved performance at all?

Visual Studio 2010 includes several tools to help you understand where your program is spending its time and how it behaves as a multithreaded application. There's a great introduction to using these tools for measuring the performance of your multithreaded app with Visual Studio 2010 in the September 2009 *MSDN Magazine* article "Debugging Task-Based Parallel Applications in Visual Studio 2010" by Stephen Toub and Daniel Moth (msdn.microsoft.com/magazine/ee410778), plus there's a good video introduction by Daniel Moth on Channel 9 (channel9.msdn.com/posts/DanielMoth/Parallel-Tasks--new-Visual-Studio-2010-debugger-window/).

Parallel programming requires you to actually make measurements to verify that you've truly improved performance and are utilizing all the hardware. To learn more about how parallelization was being used in my example application, let's use these tools to measure the sorting routines in action. I launched the Visual Studio 2010 Performance Wizard to take concurrency measurements of my sorting application while it runs.

The first thing you want to look at is core utilization, which shows how the application made use of the CPU cycles available. My test program runs the serial sort, sleeps for a second, then runs the parallel version of the sort. On my 4-core machine I get the graph of core utilization shown in **Figure 4**. The green is my application, yellow

is the OS and other programs, and gray is idle. The flat line at the 1-core level shows that I totally saturate the processing on a single core when I'm running the serial version, and that I get about 2.25 out of four cores when running the parallel version. Not too surprisingly, the time to execute the parallel sort is about 45 percent of the time required for the serial sort. Not too shabby for changing two lines of code.

Now, let's switch from looking at the CPU utilization chart to the thread display shown in **Figure 5**, which shows how the application utilized threads that were available. Notice that there is a single thread for the majority of the execution time. It's not until you start spawning off tasks that more threads are created. In this display the salmon color indicates a thread that's been blocked by another thread.

In fact, the thread display shows that while I did get a significant increase in execution speed, I didn't do it very efficiently. It's perfectly OK to have a thread block, waiting for other threads as the main thread is waiting for the tasks to complete. However, what you really want to see is solid green on as many tasks as you have CPU cores. So even though the CPU utilization chart shows improved utilization of the CPU cores, when you take a closer look at how tasks were spread throughout the thread pool, you see room for further optimization.

In fact, you should always measure your code for performance after you've done some multithreading work—even work as simple as I've done here. For small jobs, you don't want to multithread because the overhead will overwhelm any threading performance. For larger jobs, you'd want to break the job up onto as many CPU cores as are available to you so as not to oversubscribe the thread pool.

What Next?

There are a number of ways to squeeze even better performance out of the code, but that's not the objective of this initial article. But you did see how to get 80 percent CPU utilization by making just a few changes to the code that make it thread-friendly. Rather than optimizing this code further, however, we're going to focus on getting maximum performance out of the CPUs on a system by architecting jobs a bit differently.

Sorting in the manner I've demonstrated here is particularly amenable to multithreading. You can calculate how far you're going to divide the job and then give each sub-job to a thread. However, while I did get a performance boost, I did leave some performance behind.

But in real applications you might run into a situation where you either have many jobs giving you groups of unique tasks, or possibly not knowing how long any particular task is going to run and having to schedule tasks around this uncertainty. It's a particularly challenging problem. In my next article, I'm going to take a look at an architecture that takes a holistic approach to threading, allowing you to distribute multiple, perhaps dissimilar jobs. I'll show you how to architect an application to make it multicore-aware from the start through the use of tasks and thread pools. ■

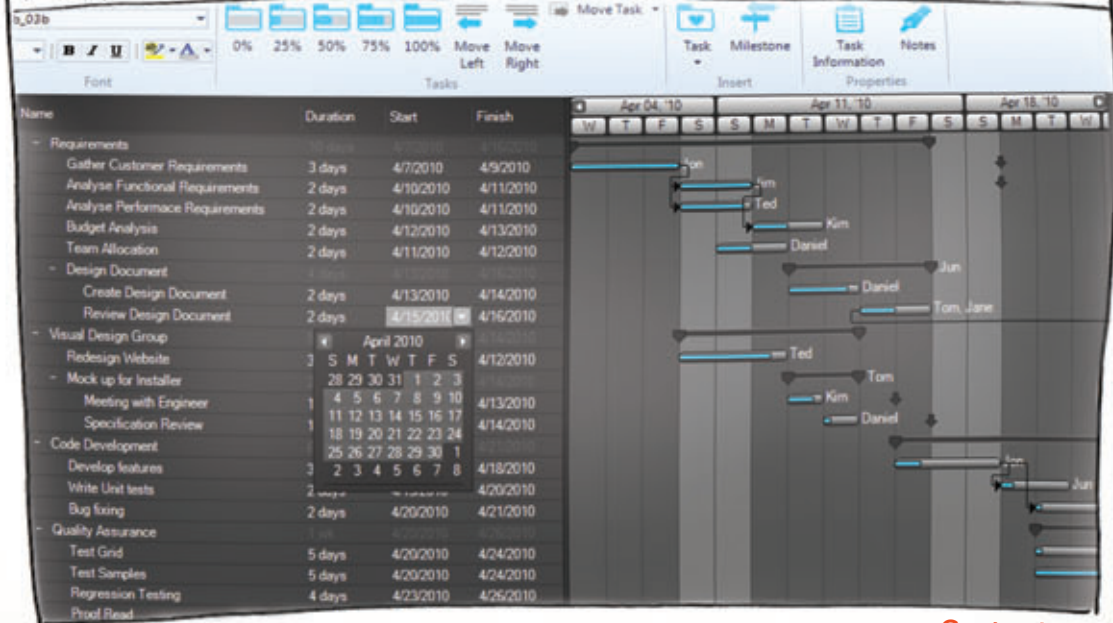
RON FOSNER has been optimizing high-performance applications and games on Windows for 20 years and is starting to get the hang of it. He's a graphics and optimization expert at Intel and is happiest when he sees all CPU cores running flat out. You can reach him at Ron@directx.com.

THANKS to the following technical expert for reviewing this article:
Stephen Toub

INNOVATION TO SPARK KILLER APPS

KILLER APPS. No Excuses.

3:53Q



Gantt Chart

You have the vision, but time, budget and staff constraints prevent you from seeing it through. With rich user interface controls like Gantt Charts that Infragistics **NetAdvantage® for .NET** adds to your Visual Studio 2010 toolbox, you can go to market faster with extreme functionality, complete usability and the “Wow-factor!” Go to infragistics.com/spark now to get innovative controls for creating Killer Apps.

Infragistics
KILLER APPS. No Excuses.

Infragistics Sales 800 231 8588
Infragistics Europe Sales +44 (0) 800 298 9055
Infragistics India +91-80-6785-1111
twitter.com/infragistics



Performance-Based Scaling in Windows Azure

Without a doubt, cloud computing is gaining lots of mindshare, and its practical use is building momentum across technology platforms and throughout the industry. Cloud computing isn't a new or revolutionary concept; indeed, it has been around for years in the form of shared hosting and other such services. Now, however, advances in technology and years of experience running servers and services have made cloud computing not only technically practical, but increasingly interesting to both consumers and providers.

Progress in cloud computing will reach beyond IT and touch every part of your company—from the people managing hardware and services, to the developers and architects, to the executives who will approve the budget and pay the bills. You'd better be prepared for it.

In this column I'll focus primarily on the developers and architects who need to understand and leverage cloud computing in their work. I'll supply some guidance on how to accomplish a given task, including notes on architecture considerations and their impact on cost and performance. Please tell me what you think of the topics I cover and, even more importantly, about topics that are of particular interest in cloud computing.

Seeding the Cloud

One of the first benefits people focus on in cloud computing is the idea that application owners don't have to worry about infrastructure setup, configuration or maintenance. Let's be honest: that's pretty compelling.

However, I think it's more important to focus on the ability to scale up or down to serve the needs of the application owner, thereby creating a more efficient cost model without sacrificing performance or wasting resources. In my experience, demand elasticity is something that comes up in any conversation about the cloud, regardless of the platform being discussed.

In this installment I'll demonstrate how to use performance counter data from running roles to automate the process of shrinking or growing the number of instances of a particular Web Role. To do this, I'll take a look at a broad cross-section of Windows Azure features and functionality, including Windows Azure Compute, Windows Azure Storage and the REST Management API.

The concept is quite simple: test collected performance data against a threshold and then scale the number of instances up or down accordingly. I won't go into detail about collecting diagnostic data—I'll leave that to you or to a future installment. Instead, I'll examine performance counter data that has been dumped to a table in Windows Azure Storage, as well as the code and setup

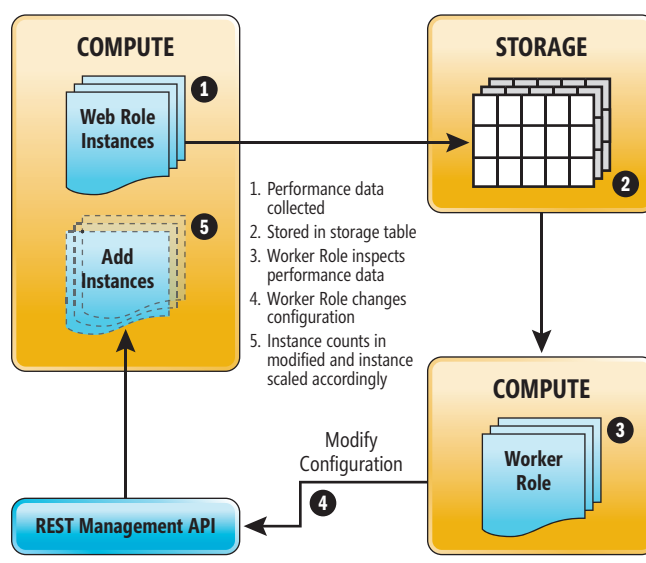


Figure 1 Performance-Based Scaling

required to execute the REST call to change the instance count in the configuration. Moreover, the downloadable code sample will contain a simple page that will make REST management calls to force the instance count to change based on user input. The scenario is something like the drawing in Figure 1.

Project Setup

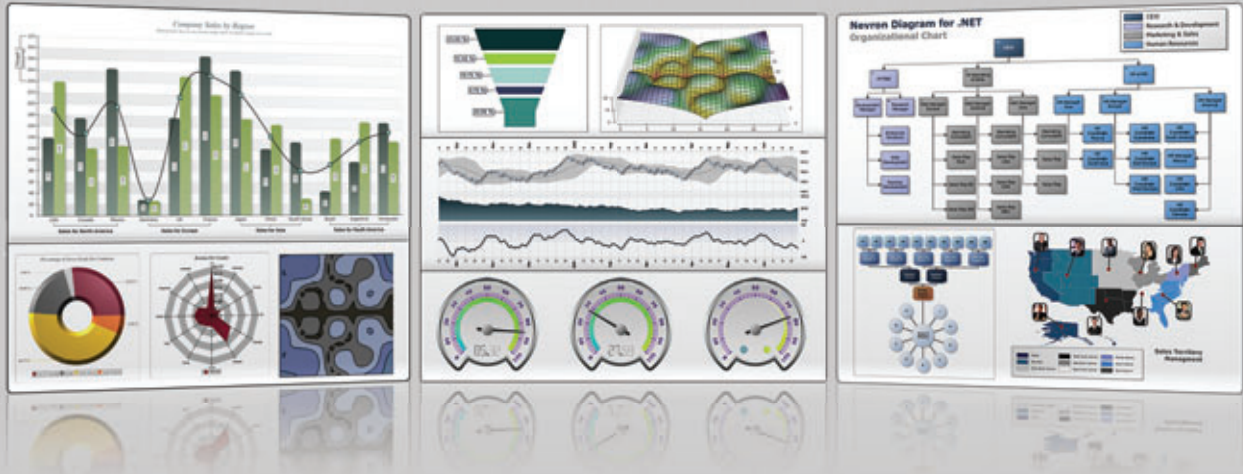
To get things started, I created a Windows Azure Cloud Service project that contains one Worker Role and one Web Role. I configured the Web Role to publish performance counter data, specifically % Processor Time, from the role and push it to storage every 20 seconds. The code to get that going lives inside of the WebRole::OnStart method and looks something like this:

```
var performanceConfiguration =  
    new PerformanceCounterConfiguration();  
performanceConfiguration.CounterSpecifier =  
    @"\Processor(_Total)\% Processor Time";  
performanceConfiguration.SampleRate =  
    System.TimeSpan.FromSeconds(1.0);  
  
// Add the new performance counter to the configuration  
config.PerformanceCounters.DataSources.Add(  
    performanceConfiguration);  
config.PerformanceCounters.ScheduledTransferPeriod =  
    System.TimeSpan.FromSeconds(20.0);
```

Code download available at code.msdn.microsoft.com/mag201010Cloudy.

Let us help you visualize your success




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



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



Developers

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



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

IT Professionals

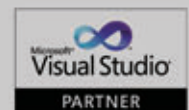
Nevron Reporting Services Vision instantly enhances your SQL Server Reporting Services 2005/2008 reports with the industry leading data visualization technology.

-  Chart for SSRS
-  Gauge for SSRS

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

-  Chart for SharePoint
-  Gauge for SharePoint

MAKE SURE THAT YOUR DATA IS MAKING THE VISUAL STATEMENT IT DESERVES BY DOWNLOADING YOUR FREE EVALUATION COPY FROM [WWW.NEVRON.COM](http://www.nevron.com) TODAY.



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

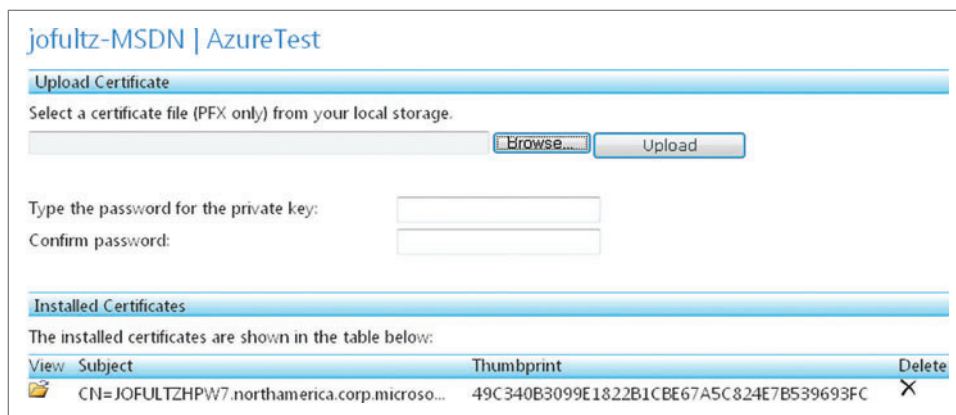


Figure 2 Importing Certificates

This code registers the performance counter, sets the collection interval for data and then pushes the data to storage. The values I used for intervals work well for this sample, but are not representative of values I'd use in a production system. In a production system, the collection interval would be much longer as I'd be concerned with 24/7 operations. Also, the interval to push to storage would be longer in order to reduce the number of transactions against Windows Azure Storage.

If the instances are running too hot, I can add instances.

Next I create a self-signed certificate that I can use to make the Azure REST Management API calls. Every request will have to be authenticated and the certificate is the means to accomplish this. I followed the instructions for creating a self-signed certificate in the TechNet Library article "Create a Self-Signed Server Certificate in IIS 7" (technet.microsoft.com/library/cc753127(WS.10)). I exported both a .cer file and a .pfx file. The .cer file will be used to sign the requests I send to the management API and the .pfx file will be imported into the compute role via the management interface (see Figure 2).

I'll come back later and grab the thumbprint to put it in the settings of both the Web Roles and Worker Roles that I'm creating so they can access the certificate store and retrieve the certificate.

Finally, to get this working in Windows Azure, I need a compute project where I can publish the two roles and a storage project to which I can transfer the performance data. With these elements in place, I can move on to the meat of the work.

Is It Running Hot or Cold?

Now that I've got the Web Role configured and code added to publish the performance counter data, the next step is to fetch that data and compare it to a threshold value. I'll create a TestPerfData method in which I retrieve the data from the table and test the values. I'll write a LINQ statement similar to the following:

```
double AvgCPU = (
    from d in selectedData
    where d.CounterName ==
        @"\"Processor(_Total)\% Processor Time"
    select d.CounterValue).Average();
```

By comparing the average utilization, I can determine the current application performance. If the instances are running too hot, I can add instances. If they're running cold and I'm wasting resources—meaning money—by having running instances I don't need, I can reduce the number of instances.

You'll find in-depth coverage of the code and setup needed to access the performance counter table data in a blog post I wrote at blogs.msdn.com/b/joseph_fultz/archive/2010/06/30/querying-azure-perf-counter-data-with-linq.aspx. I use a simple if-then-else block to assess the state and determine the desired action. I'll cover the details after I've created the functions needed to change the running service configuration.

Using the REST Management API

Before I can finish the TestPerfData method, I have a little more work to do. I need a few methods to help me discover the number of instances of a given role, create a new valid service configuration for that role with an adjusted instance count, and, finally, allow me to update the configuration.

To this end I've added a class file to my project and created the six static methods shown in Figure 3.

The calls that interact with the REST Management API must include a certificate. To accomplish this, the certificate is added to the hosted service and the thumbprint is added to the role configuration and used to fetch the certificate at run time. Once the service and role are configured properly, I use the following code to grab the certificate from the Certificate Store:

```
string Thumbprint =
    RoleEnvironment.GetConfigurationSettingValue(
        ThumbprintSettingName);
X509Store certificateStore =
    new X509Store(StoreName.My, StoreLocation.LocalMachine);
certificateStore.Open(OpenFlags.ReadOnly);
X509Certificate2Collection certs =
    certificateStore.Certificates.Find(
        X509FindType.FindByThumbprint, Thumbprint, false);
```

Figure 3 Configuration Methods

Method	Description
GetDeploymentInfo	Retrieves the deployment configuration, including the encoded service configuration.
GetServiceConfig	Retrieves and decodes the service configuration from the deployment info.
GetInstanceCount	Fetches the instance count for a specified role.
ChangeInstanceCount	Updates the service configuration and returns the complete XML.
ChangeConfigFile	Updates the service configuration with the service configuration provided to the function.
LookupCertificate	Passes in the environment setting containing the thumbprint and retrieves the certificate from the certificate store.

Figure 4 The Less-Than-85-Percent Block

```
else if (AvgCPU < 85.0) {
    Trace.TraceInformation("in the AvgCPU < 25 test.");
    string deploymentInfo =
        AzureRESTMgmtHelper.GetDeploymentInfo();
    string svcconfig =
        AzureRESTMgmtHelper.GetServiceConfig(deploymentInfo);
    int InstanceCount =
        System.Convert.ToInt32(
            AzureRESTMgmtHelper.GetInstanceCount(
                svcconfig, "WebRole1"));
    if (InstanceCount > 1) {
        InstanceCount--;
        string UpdatedSvcConfig =
            AzureRESTMgmtHelper.ChangeInstanceCount(
                svcconfig, "WebRole1", InstanceCount.ToString());
        AzureRESTMgmtHelper.ChangeConfigFile(UpdatedSvcConfig);
    }
}
```

This is the main code of the `LookUpCertificate` method and it's called in the methods where I want to interact with the REST API. I'll review the `GetDeploymentInfo` function as an example of how calls are constructed. For this example, I've hardcoded some of the variables needed to access the REST API:

```
string x_ms_version = "2009-10-01";
string SubscriptionID = "[your subscription ID]";
string ServiceName = "[your service name]";
string DeploymentSlot = "Production";
```

A simple conditional statement tests whether utilization is higher or lower than 85 percent.

I need to create a `HttpRequest` with the proper URI, set the request headers and add my certificate to it. Here I build the URI string and create a new `HttpRequest` object using it:

```
string RequestUri = "https://management.core.windows.net/" +
    SubscriptionID + "/services/hostedservices/" +
    ServiceName + "/deploymentslots/" + DeploymentSlot;
```

```
HttpRequest RestRequest =
    (HttpRequest)HttpRequest.Create(RequestUri);
```

For the call to be valid, it must include the version in the header. Thus, I create a name-value collection, add the version key and data, and add that to the request headers collection:

```
NameValueCollection RequestHeaders =
    new NameValueCollection();
RequestHeaders.Add("x-ms-version", x_ms_version);
if (RequestHeaders != null) {
    RestRequest.Headers.Add(RequestHeaders);
}
```

The last thing to do to prepare this particular request is to add the certificate to the request:

```
X509Certificate cert = LookupCertificate("RESTMgmtCert");
RestRequest.ClientCertificates.Add(cert);
```

Finally, I execute the request and read the response:

```
RestResponse = RestRequest.GetResponse();
using (StreamReader RestResponseStream = new StreamReader(RestResponse.
    GetResponseStream(), true)) {
    ResponseBody = RestResponseStream.ReadToEnd();
    RestResponseStream.Close();
}
```

That's the general pattern I used to construct requests made to the REST Management API. The `GetServiceConfig` function extracts the Service Configuration out of the deployment configuration, using LINQ to XML statements like the following:

```
XElement DeploymentInfo = XElement.Parse(DeploymentInfoXML);
string EncodedServiceConfig =
    (from element in DeploymentInfo.Elements()
     where element.Name.LocalName.Trim().ToLower() == "configuration"
     select (string) element.Value).Single();
```

In my code, the return of the `GetServiceConfig` is passed on to the `GetInstanceCount` or `ChangeInstanceCount` functions (or both) to extract the information or update it. The return from the `ChangeInstance` function is an updated Service Configuration, which is passed to `ChangeConfigFile`. In turn, `ChangeConfigFile` pushes the update to the service by constructing a request similar to the previous one used to fetch the deployment information, with these important differences:

1. `"/?comp=config"` is added to the end of the URI
2. The PUT verb is used instead of GET
3. The updated configuration is streamed as the request body

Putting It All Together

With the functions in place to look up and change the service configuration, and having done the other preparatory work such as setting up counters, configuring the connection string settings for Storage, and installing certificates, it's time to implement the CPU threshold test.

The Visual Studio template produces a Worker Role that wakes up every 10 seconds to execute code. To keep things simple, I'm leaving that but adding a single timer that will run every five minutes. In the timer, a simple conditional statement tests whether utilization is higher or lower than 85 percent, and I'll create two instances of the Web Role. By doing this I guarantee that the number of instances will definitely decrease from the initial two instances to a single instance.

Inside the Worker Role I have a `Run` method that declares and instantiates the timer. Inside of the timer-elapsed handler I add a call to the `TestPerfData` function I created earlier. For this sample,

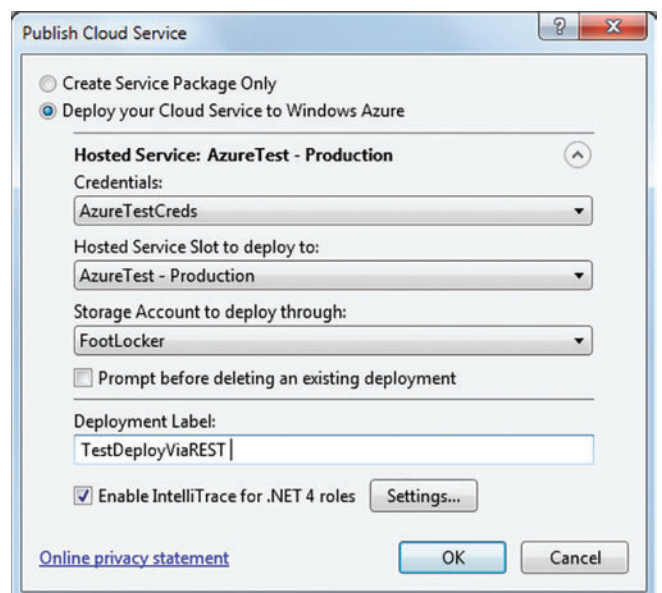


Figure 5 Publishing the Project

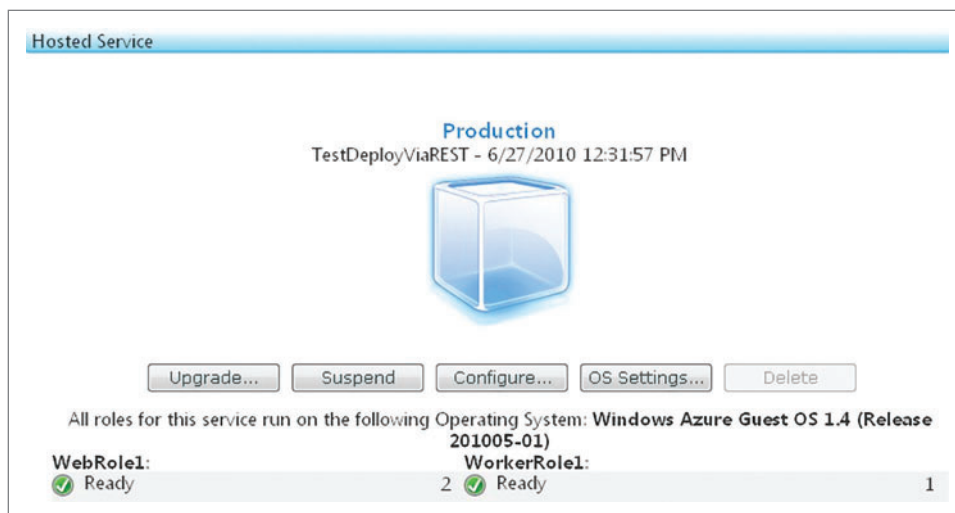


Figure 6 Two Web Roles and One Worker Role

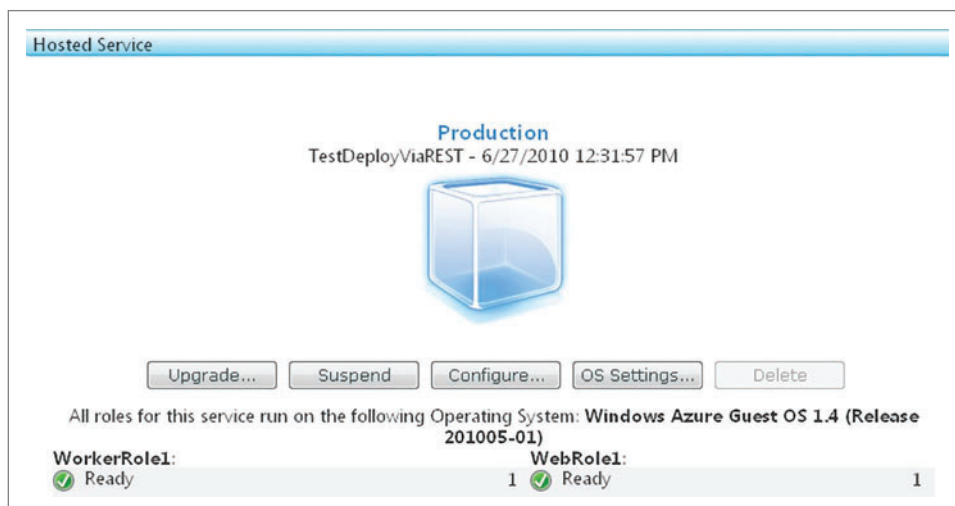


Figure 7 Now One Web Role and One Worker Role

I'm skipping the implementation of the greater-than condition because I know that the CPU utilization will not be that high. I set the less-than condition to be less than 85 percent as I'm sure the counter average will be lower than that. Setting these contrived conditions will allow me to see the change via the Web management console or via Server Explorer in Visual Studio.

In the less-than-85-percent block I check the instance count, modify the service configuration and update the running service configuration, as shown in **Figure 4**.

I make sure to check the instance count before adjusting down, because I don't want it to go to zero, as this is not a valid configuration and would fail.

Running the Sample

I'm now ready to execute the example and demonstrate elasticity in Windows Azure. Knowing that my code is *always* right the first time—ahem—I right-click on the Cloud Service Project and click Publish. The dialog gives you the option to configure your credentials, which I've already done (see **Figure 5**).

I click OK and just have to wait for the package to be copied up and deployed. When deployment is complete, I switch to the Web management console and see two Web Roles and one Worker Role running, as shown in **Figure 6**.

I wait for the timer event to fire, executing the code that will determine that the average CPU utilization is less than 85 percent, and decrement the WebRole1 instance count. Once this happens, the management page will refresh to reflect an update to the deployment.

Because I'm using small VMs, changing the count by only one, and the application is lightweight (one .aspx page), the update doesn't take long and I see the final, auto-shrunk deployment as shown in **Figure 7**.

Blue Skies

I want to share few final thoughts about the sample in the context of considering a real implementation. There are a few important points to think about.

First, the test is trivial and contrived. In a real implementation you'd need to evaluate more than simple CPU utilization and you'll need to take into account the quantum over which the collection occurred.

In addition, you need to evaluate the costs of using Windows Azure Storage. Depending on the solution, it might be advisable to scrub the records in the table for only ones that are of interest. You can decrease the upload interval to lower transaction costs, or you may want to move the data to SQL Azure to minimize that cost.

You also need to consider what happens during an update. A direct update will cause users to lose connectivity. It may be better to bring the new instances up in staging and then switch the virtual IP address. In either case, however, you'll have session and viewstate problems. A better solution is to go stateless and disable the test during scale adjustments.

That's it for my implementation of elasticity in Windows Azure. Download the code sample and start playing with it today. ■

JOSEPH FULTZ is an architect at the Microsoft Technology Center in Dallas where he works with both Enterprise Customers and ISVs designing and prototyping software solutions to meet business and market demands. He's spoken at events such as Tech•Ed and similar internal training events.

THANKS to the following technical expert for reviewing this article: Suraj Puri



Multiparadigmatic .NET, Part 2

In my previous article (msdn.microsoft.com/magazine/ff955611), the first of this series, I mentioned that the two languages central to the Microsoft .NET Framework—C# and Visual Basic—are multiparadigm languages, just like C++, their syntactic (in the case of C#) or conceptual (in the case of Visual Basic) predecessor. Using a multiparadigmatic language can be confusing and difficult, particularly when the purposes of the different paradigms aren't clear.

Commonality and Variability

But before we can start taking apart the different paradigms in these languages, a bigger question comes to mind: What, precisely, are we trying to do when we design a software system? Forget the “end result” goals—modularity, extensibility, simplicity and all that jazz—for a moment, and focus more on the “how” of the language. How, exactly, are we trying to create all those “end result” goals?

James O. Coplien—from his “Multi-Paradigm Design for C++” (Addison-Wesley Professional, 1998)—has an answer for us:

When we think abstractly, we emphasize what is common while suppressing detail. A good software abstraction requires that we understand the problem well enough in all of its breadth to know what is common across related items of interest and to know what details vary from item to item. The items of interest are collectively called a family, and families—rather than individual applications—are the scope of architecture and design. We can use the commonality/variability model regardless of whether family members are modules, classes, functions, processes or types; it works for any paradigm. Commonality and variability are at the heart of most design techniques.

Think about the traditional object paradigm for a moment. As object-oriented developers, we're taught from early on to “identify the nouns” in the system and look for the things that make up a particular entity—to find all the things that pertain to being a “teacher” within the system, for example, and put them into a class called Teacher. But if several “nouns” have overlapping and related behavior—such as a “student” having some data and operations overlapping with a “person” but with some marked differences—then we're taught that rather than replicate the common code, we should elevate the commonality into a base class, and relate the types to one another through inheritance. In other words, commonality is gathered together within a class, and variability is captured by extending from that class and introducing the variations. Finding the commonalities and variabilities within a system, and expressing them, forms the heart of design.

Commonalities are often the parts that are difficult to explicitly identify, not because we don't recognize them, but because they're so easily and intuitively recognizable it's tough to spot them. For example, if I say, “Vehicle,” what image pops into your head? If we do this exercise with a group of people, each will have a different image, yet there will be vast commonality among them all. However, if we start listing the various vehicles imagined, the different kinds of variabilities begin to emerge and categorize themselves (we hope), such that we can still have some set of commonalities among the vehicles.

Positive and Negative Variability

Variability can come in two basic forms, one of which is easy to recognize and the other much more difficult. Positive variability is when the variability occurs in the form of adding to the basic commonality. For example, imagine the abstraction desired is that of a message, such as a SOAP message or e-mail. If we decide that a Message type has a header and body, and leave different kinds of messages to use that as the commonality, then a positive variability on this is a message that carries a particular value in its header, perhaps the date/time it was sent. This is usually easily captured in language constructs—in the object-oriented paradigm, for example, it's relatively trivial to create a Message subclass that adds the support for date/time sent.

Negative variability, however, is much trickier. As might be inferred, a negative variability removes or contradicts some facet of the commonality—a Message that has a header but no body (such as an acknowledgement message used by the messaging infrastructure) is a form of negative variability. And, as you can probably already guess, capturing this in a language construct is problematic—neither C# nor Visual Basic has a facility to remove a member declared in a base class. The best we could do in this case is return null or nothing from the Body member, which will clearly play havoc with any code that expects a Body to be present, such as verification routines that run a CRC on the Body to ensure it was transmitted correctly.

(Interestingly, XML Schema types offer negative variability in their schema-validation definitions, something that no mainstream programming language yet offers, which is one of the ways that the XML Schema Definition can mismatch against programming languages. Whether this will become a forthcoming feature in some as-yet-unwritten programming language, and whether it would be a Good Thing To Have, is an interesting discussion best had over beer.)

Figure 1 Partial Implementation of a Point Class

```
Public Class Point
    Public Sub New(ByVal XX As Integer, ByVal YY As Integer)
        Me.X = XX
        Me.y = YY
    End Sub

    Public Property X() As Integer
    Public Property y() As Integer

    Public Function Distance(ByVal other As Point) As Double
        Dim XDiff = Me.X - other.X
        Dim YDiff = Me.y - other.y
        Return System.Math.Sqrt((XDiff * XDiff) + (YDiff * YDiff))
    End Function

    Public Overrides Function Equals(ByVal obj As Object) As Boolean
        ' Are these the same type?
        If Me.GetType() = obj.GetType() Then
            Dim other As Point = obj
            Return other.X = Me.X And other.y = Me.y
        End If
        Return False
    End Function

    Public Overrides Function ToString() As String
        Return String.Format("{0},{1}", Me.X, Me.y)
    End Function
End Class
```

In many systems, negative variability is often handled using explicit code constructs at the client level—meaning, it's up to the users of the Message type to do some kind of if/else test to see what kind of Message it is before examining the Body, which renders the work put into the Message family all but irrelevant. Too much negative variability escaping the design is usually the underlying cause of calls from developers to “pitch it all and start over.”

Binding Commonality and Variability

The actual moment that commonality and variability are set varies with each paradigm, and in general, the closer to run time that we

Figure 2 A New Point Class with Floating-Point Members

```
Public Class PointD
    Public Sub New(ByVal XX As Double, ByVal YY As Double)
        Me.X = XX
        Me.y = YY
    End Sub

    Public Property X() As Double
    Public Property y() As Double

    Public Function Distance(ByVal other As Point) As Double
        Dim XDiff = Me.X - other.X
        Dim YDiff = Me.y - other.y
        Return System.Math.Sqrt((XDiff * XDiff) + (YDiff * YDiff))
    End Function

    Public Overrides Function Equals(ByVal obj As Object) As Boolean
        ' Are these the same type?
        If Me.GetType() = obj.GetType() Then
            Dim other As PointD = obj
            Return other.X = Me.X And other.y = Me.y
        End If
        Return False
    End Function

    Public Overrides Function ToString() As String
        Return String.Format("{0},{1}", Me.X, Me.y)
    End Function
End Class
```

can bind those decisions, the more control we give customers and users over the system's evolution and as a whole. When discussing a particular paradigm or technique within a paradigm, it's important to recognize in which of these four “bind times” the variability kicks in:

1. **Source time.** This is the time before the compiler fires up, when the developer (or some other entity) is creating the source files that will eventually be fed into the compiler. Code-generative techniques, such as the T4 template engine—and to a lesser degree the ASP.NET system—operate at a source-time binding.
2. **Compile time.** As its name implies, this binding occurs during the compiler's pass over the source code to render it into compiled bytecode or executable CPU instructions. A great deal of decision making is finalized here, though not all of it, as we'll see.
3. **Link/load time.** At the time the program loads and runs, an additional point of variability kicks in, based on the specific modules (assemblies, in the case of .NET; DLLs, in the case of native Windows code) that are loaded. This is commonly referred to as a plug-in- or add-in-style architecture when it's applied at a whole-scale program level.
4. **Run time.** During the program's execution, certain variabilities may be captured based on user input and decision making, and potentially different code executed (or even generated) based on those decisions/input.

In some cases, the design process will want to start from these “bind times” and work backward to figure out what language constructs can support the requirement; for example, a user may want to have the ability to add/remove/modify variability at run time (so that we don't have to go back through a compile cycle or reload code), which means that whatever paradigm the designer uses, it must support a runtime variability binding.

Challenge

In my previous article, I left readers with a question:

As an exercise, consider this: The .NET Framework 2.0 introduced generics (parameterized types). Why? From a design perspective, what purpose do they serve? (And for the record, answers of “It lets us have type-safe collections” are missing the point—Windows Communication Foundation uses generics extensively, clearly in ways that aren't just about type-safe collections.)

Taking this a little further, look at the (partial) implementation of a Point class in **Figure 1**, representing a Cartesian X/Y point, like pixel coordinates on a screen, or a more classical graph.

In and of itself, it's not really all that exciting. The rest of the implementation is left to the reader's imagination, because it's not central to the discussion.

Notice that this Point implementation has made a few assumptions about the way Points are supposed to be utilized. For example, the X and Y elements of the Point are integers, meaning that this Point class can't represent fractional Points, such as Points at (0.5,0.5). Initially, this may be an acceptable decision, but inevitably, a request will come up asking to be able to represent “fractional Points” (for whatever reason). Now, the developer has an interesting problem: How to represent this new requirement?

Starting from the basics, let's do the "Oh Lord don't do that" thing and simply create a new Point class that uses floating-point members instead of integral members, and see what emerges (see **Figure 2**; note that PointD is short for "Point-Double," meaning it uses Doubles). As is pretty clear, there's a lot of conceptual overlap here between the two Point types. According to the commonality/variability theory of design, that means we need to somehow capture the common parts and allow for variability. Classic object-orientation would have us do it through inheritance, elevating the commonality into a base class or interface (Point), then implementing that in subclasses (PointI and PointD, perhaps).

Interesting problems emerge from attempting this, however. First, the X and Y properties need a type associated with them, but the variability in the two different subclasses concerns how the X and Y coordinates are stored, and thus represented, to users. The designer could always simply opt for the largest/widest/most comprehensive representation, which in this case would be a Double, but doing so means having a Point that can only have Integer values is now lost as an option, and it undoes all of the work the inheritance was intended to permit. Also, because they're related by inheritance now, the two Point-inheriting implementations are now supposedly interchangeable, so we should be able to pass a PointD into a PointI Distance method, which may or may not be desirable. And is a PointD of (0.0, 0.0) equivalent (as in Equals) to a PointI of (0,0)? All these issues have to be considered.

Even if these problems are somehow fixed or made tractable, other problems emerge. Later, we might want a Point that accepts values larger than can be held in an Integer. Or only absolute-positive values (meaning the origin is in the lower-left corner) are deemed acceptable. Each of these different requirements will mean new subclasses of Point must be created.

Stepping back for a moment, the original desire was to reuse the commonality of the implementation of Point but allow for variability in the type/representation of the values that make up the

Figure 3 A Generic Constraint on Type


```
Public Class GPoint(Of Rep As {Comparable, IConvertible})
    Public Sub New(ByVal XX As Rep, ByVal YY As Rep)
        Me.X = XX
        Me.Y = YY
    End Sub

    Public Property X() As Rep
    Public Property Y() As Rep

    Public Function Distance(ByVal other As GPoint(Of Rep)) As Double
        Dim XDiff = (Me.X.ToDouble(Nothing)) - (other.X.ToDouble(Nothing))
        Dim YDiff = (Me.Y.ToDouble(Nothing)) - (other.Y.ToDouble(Nothing))
        Return System.Math.Sqrt((XDiff * XDiff) + (YDiff * YDiff))
    End Function

    Public Overrides Function Equals(ByVal obj As Object) As Boolean
        ' Are these the same type?
        If Me.GetType() = obj.GetType() Then
            Dim other As GPoint(Of Rep) = obj
            Return (other.X.CompareTo(Me.X) = 0) And (other.Y.CompareTo(Me.Y) = 0)
        End If
        Return False
    End Function

    Public Overrides Function ToString() As String
        Return String.Format("({0},{1}", Me.X, Me.Y)
    End Function
End Class
```



Discover **SCALABLE** Performance


SCALEOUT STATESERVER®


Simply Powerful Distributed Caching

As a software architect, you know the importance of combining simplicity with power. ScaleOut StateServer's distributed cache gives you both.

Its powerful architecture simplifies both configuration and management so that you can quickly - and easily - tap into blazing performance, scalability, and 24x7 availability.

Give your server farm and grid applications the performance boost they need. Download your **FREE** trial copy of ScaleOut StateServer today!




SCALEOUT SOFTWARE
www.scaleoutsoftware.com/eval | (503) 643-3422

Point. In the ideal, depending on the kind of graph we're working with, we should be able to choose the representation at the time the Point is created, and it would represent itself as an entirely distinct and different type, which is precisely what generics do.

Doing this, however, represents a problem: The compiler insists that "Rep" types won't necessarily have "+" and "-" operators defined for it, because it thinks we want to put any possible type here—Integers, Longs, Strings, Buttons, DatabaseConnections or whatever else comes to mind—and that's clearly a little too variable. So, once again, we need to express some commonality to the type that can be used here, in the form of a generic constraint on what types "Rep" can be (see **Figure 3**).

In this case, two constraints are imposed: one to ensure that any "Rep" type can be converted to double values (to calculate the distance between the two points), and the other to ensure that the constituent X and Y values can be compared to see if they're greater-than/equal-to/less-than one another.

And now the reason for generics becomes clearer: they support a different "axis" of variability for design, one that's drastically different from the traditional inheritance-based axis. It allows the designer to render the implementation as commonalities, and the types being operated upon by the implementation to be variabilities.

Note that this implementation assumes that the variability is occurring at compile time, rather than at link/load time or run time—if the user wants or needs to specify the type of the X/Y members of the Point at run time, then a different solution is needed.

Not Dead (or Done) Yet!

If all of software design is a giant exercise in commonality and variability, then the need to understand multiparadigmatic design becomes clear: Each of the different paradigms offers different ways to achieve this commonality/variability, and mixing the paradigms creates confusion and leads to calls for a complete rewrite. Just as the human brain starts to get confused when we try to map three-dimensional constructs in our head into four and five dimensions, too many dimensions of variability in software cause confusion.

In the next half-dozen or so articles, I'll be looking at the different ways that each paradigm supported by C# and Visual Basic—the structural, object-oriented, metaprogramming, functional and dynamic paradigms being the principals—provide functionality to capture commonality and allows for variability. When we're through all of that, we'll examine how some of them can be combined in interesting ways to make your designs more modular, extensible, maintainable and all that other stuff.

Happy coding! ■

TED NEWARD is a principal with Neward & Associates, an independent firm specializing in enterprise .NET Framework and Java platform systems. He's written more than 100 articles, is a C# MVP and INETA speaker and has authored and coauthored a dozen books, including "Professional F# 2.0" (Wrox, 2010). He also consults and mentors regularly. Reach him at ted@tedneward.com with questions or consulting requests, and read his blog at blogs.tedneward.com.

THANKS to the following technical expert for reviewing this article:
Anthony Green



Get Microsoft SharePoint® 2010
BCS Functionality in WSS 3.0 or Higher!

The first 100 buyers get it for
ONLY \$1,495

Free 30 Day Trial, unlimited users, money back
guarantee - what are you waiting for?

Plus check out our **Monthly Contests**
under the Developer Zone tab.

STONE BOND
TECHNOLOGIES

Enterprise Enabler®
Agile Integration Software

Microsoft
GOLD CERTIFIED
Partner

*Microsoft and SharePoint are registered
trade-marks of Microsoft Corporation.

www.enterpriseenabler.com



Multi-Touch and Inertia

UIs on computers seem to be most appealing when they mask the digital nature of the underlying technology, and instead mimic the analog feel of the real world. This trend began when graphical interfaces started replacing command lines, and then continued with the increased use of photographs, sound and other media.

The incorporation of multi-touch into video displays has given an extra boost to the process of making user controls more lifelike and intuitive. I think you've only begun to glimpse the potential of the touch revolution to revamp your relationship with the computer screen.

Perhaps someday we will even eliminate one of the ugliest vestiges of digital technology known to man: the push-button volume control. The basic dial that controlled volume (and other settings) on radios and televisions was delightfully simple, but has been replaced with awkward push buttons on remote controls and the appliances themselves.

Computer interfaces often offer sliders for volume control. These are almost as good as dials. But push-button volume controls are still persistent. Even the multi-touch screen of the Zune HD wants you to press plus and minus signs to increase and decrease volume. More preferable would be a dial that responds to touch.

I like the dial for multi-touch. Perhaps that's why I focused on simulating dials in my article, "Finger Style: Exploring Multi-Touch Support in Silverlight," in the March 2010 issue of *MSDN Magazine* (msdn.microsoft.com/magazine/ee336026), and why I'm returning to dials to talk about handling multi-touch inertia in the Windows Presentation Foundation (WPF).

The Inertia Event

One of the ways in which a multi-touch interface attempts to mimic the real world is by introducing inertia—the tendency for objects to maintain the same velocity unless acted upon by other forces, such as friction. In multi-touch interfaces, inertia can keep a visual object moving when the fingers have left the screen. The most common application of touch inertia is in navigating lists, and inertia is already built into the WPF ListBox.

Among the downloadable code for this article is a little program called *InertialListBoxDemo* that simply fills a WPF ListBox with a bunch of items so you can test it out on your multi-touch display.

In your own WPF controls, you'll need to decide how much inertia you want, if any. Handling inertia is easiest if you just want inertia to cause the same response from your program as direct manipulation. The hard part is getting a good understanding of the numeric values involved.

As you've seen in previous articles, a WPF element is informed of the onset of multi-touch manipulation with two events: *ManipulationStarting* (which is a good opportunity for performing some initialization) and *ManipulationStarted*. These two events are then followed by potentially very many *ManipulationDelta* events, which consolidate the action of one, two or more fingers into translation, scaling and rotation information.

You've only begun to
glimpse the potential of the
touch revolution.

When all the fingers have lifted from the element being manipulated, a *ManipulationInertiaStarting* event occurs. This is your program's opportunity to specify how much inertia you want. (I'll discuss how, shortly.) The effect of inertia takes the form of additional *ManipulationDelta* events until the object "runs down" and a *ManipulationCompleted* event signals the end. If you don't do anything with *ManipulationInertiaStarting*, it's followed immediately by *ManipulationCompleted*.

The use of the *ManipulationDelta* event to indicate both direct manipulations and inertia makes this whole scheme extremely easy to use. Basically, you can implement inertia with a single statement in the *ManipulationInertiaStarting* event. If necessary, you can discern the difference between direct manipulation and inertia by the *IsInertial* property of *ManipulationDeltaEventArgs*.

As you'll see, you specify how much inertia you want in one of two different ways, but they both involve deceleration—a decrease in velocity over a period of time until the velocity reaches zero and the object stops moving. This involves some concepts in physics that most programmers don't encounter often, so perhaps a little refresher course might help.

A Refresher on Acceleration

If something is moving, it is said to have a *speed* or *velocity* that can be expressed in units of distance-per-time. If the velocity is itself changing over the course of time, then the object has *acceleration*.

Code download available at code.msdn.microsoft.com/mag201010UIFrontiers.

A negative acceleration (indicating decreasing velocity) is often called *deceleration*.

Throughout this discussion, let's assume that the acceleration or deceleration itself is constant. A constant acceleration means that velocity changes linearly—an equal amount per unit of time. If acceleration is 0, the velocity remains constant.

For example, automobile manufacturers sometimes advertise their cars as being able to accelerate from 0 to 60 mph in a certain number of seconds, let's say, 8 seconds. The car is initially at rest but by the end of 8 seconds, it's going at 60 mph, as shown in **Figure 1**.

Notice that the velocity increases by the same amount every second. The acceleration is expressed as the change in velocity during a particular period of time, or in this case, 7.5 mph per second.

That acceleration value is a little awkward because there are two units of time involved: hours and seconds. Let's get rid of hours and just use seconds. Because 60 mph is 88 feet per second, the car could equivalently be said to go from 0 to 88 feet per second in 8 seconds. The acceleration is 11 feet per second per second, or (as it's often phrased) 11 feet per second *squared*.

We can go back to miles and hours, but the units get ridiculous as the acceleration is calculated as 27,000 mph squared. This implies that at the end of the hour, the car is traveling 27,000 mph. Of course it doesn't do that. In real life, as soon as the car gets up to 60 mph or thereabouts, the velocity levels off and the acceleration goes down to 0. That's a change in acceleration, which in engineering circles is referred to as a *jerk*.

You specify how much inertia you want in one of two different ways.

But for this exercise we're assuming constant acceleration. Starting at a velocity of 0, as a result of acceleration *a*, the distance *x* traveled during time *t* is:

$$x = \frac{1}{2}at^2$$

The $\frac{1}{2}$ and square are necessary because the velocity *v* is calculated as the first derivative of the distance with respect to time:

$$v = \frac{dx}{dt} = at$$

If *a* is 11 feet per second squared, then at *t* equal to 1 second, the velocity is 11 feet per second, but the car has only traveled 5.5 feet. At *t* equal to 2 seconds, the velocity is 22 feet per second, and the car has traveled a total of 22 feet. During each second, the car travels a distance based on the average velocity during that second. At *t* equal to 8 seconds, the velocity is 88 feet per second, and the car has traveled a total of 352 feet.

Multi-touch inertia is like watching the car in reverse: At the time the fingers lift from the screen, the object has a certain velocity. The application specifies a deceleration that causes the velocity of the object to decrease linearly down to 0. The higher the deceleration, the quicker the object stops. A deceleration of 0 causes the object to continue moving at the same velocity forever.

Let's focus on rotational inertia first, because it's an effect familiar from the real world.

Two Ways to Decelerate

The ManipulationDelta event arguments include a Velocities property of type ManipulationVelocities, which itself has three properties corresponding to the three types of manipulation:

- LinearVelocity of type Vector
- ExpansionVelocity of type Vector
- AngularVelocity of type double

The first two properties are expressed as device-independent units per millisecond; the third is in angles (in degrees) per millisecond. Of course, the millisecond is not an intuitively comprehensible period of time, so if you need to look at these values and get a sense of what they mean, you'll want to multiply them by 1,000 to convert to seconds.

Applications don't often make use of these Velocity properties, but they provide initial velocities for inertia. When the user's fingers leave the screen, a ManipulationInertiaStarting event occurs. The event arguments include these three properties that let you specify inertia separately for those same three types of manipulation:

- TranslationBehavior of type InertiaTranslationBehavior
- ExpansionBehavior of type InertiaExpansionBehavior
- RotationBehavior of type InertiaRotationBehavior

Each of those classes has an InitialVelocity property, a DesiredDeceleration property, and a third property named DesiredDisplacement, DesiredExpansion or DesiredRotation, depending on the class.

For translation and rotation, all the Desired properties have default values of Double.NaN, that special bit configuration that indicates "not a number." For expansion, the Desired properties are of type Vector with X and Y values of Double.NaN, but the concept is the same.

Let's focus on rotational inertia first, because it's an effect familiar from the real world (such as a playground roundabout), and you don't have to worry about objects flying off the screen.

By the time you get a ManipulationInertiaStarting event, the InertiaRotationBehavior object has been created, and the InitialVelocity value has been set from the previous ManipulationDelta event. If the InitialVelocity is 1.08, for example, that's 1.08 degrees per millisecond, or 1,080 degrees per second, or three revolutions per second, or 180 rpm.

Figure 1 Acceleration

Seconds	Velocity
0	0 mph
1	7.5
2	15
3	22.5
4	30
5	37.5
6	45
7	52.5
8	60

Are you kidding?

Visual Studio **LIVE!**
EXPERT SOLUTIONS FOR .NET DEVELOPERS

NOVEMBER 14-17, 2010
ORLANDO, FL

1105 MEDIA

You went to a developer conference and came back with no usable, real-world training? Come to Visual Studio Live! If it's Visual Studio, Sharepoint, Silverlight, Data Management, or Cloud education you need, we've got you covered. No kidding.

For more reasons on why you should attend Visual Studio Live!, visit www.vslive.com/kidding.

To keep the object spinning, you set either `DesiredRotation` or `DesiredDeceleration` but not both. If you try to set both, the last one you set will be valid and the other will be `Double.NaN`.

The first option is to set `DesiredRotation` to a value in degrees. This is the number of degrees the object will spin before it stops. If you set `DesiredRotation` to 360, for example, the spinning object will make just one additional revolution, slowing down and stopping in the process. The advantage is that you get the same amount of inertial activity regardless of the initial speed, so it's easy to predict what will happen. The disadvantage is that it's not entirely natural.

The alternative is to set `DesiredDeceleration` to a value in units of degrees per millisecond squared, and here's where it becomes a bit slippery because it's hard to guess at a good value.

If `InitialVelocity` is 1.08 degrees per millisecond and you set `DesiredDeceleration` to 0.01 degrees per millisecond squared, then the velocity will decrease by 0.01 degrees every millisecond: to 1.07 degrees per millisecond at the end of the first millisecond, 1.06 degrees per millisecond at the end of the second millisecond, and so forth linearly until the velocity gets down to 0. That whole process will take 108 ms, or a little more than one-tenth of a second.

You probably want to set `DesiredDeceleration` to something less than that, perhaps 0.001 degrees per millisecond squared, which will cause the object to continue spinning for 1.08 seconds.

When your finger is rotating the dial, the meter shows the current angular velocity.

Watch out if you want to convert the deceleration units to something a little easier for the human mind to contemplate. A velocity of 1.08 degrees per millisecond is the same as 1,080 degrees per second, but a deceleration of 0.001 degrees per millisecond squared is 1,000 degrees per second squared. To convert deceleration to seconds you need to multiply by 1,000 *twice* because time is squared.

If you combine the two formulas shown earlier and eliminate t , you get:

$$a = \frac{v^2}{2x}$$

This implies that the two methods for setting deceleration are equivalent and can be converted from one to the other. If the `InitialVelocity` is 1.08 degrees per millisecond, and you set the `DesiredDeceleration` to 0.001 degrees per millisecond squared, that's equivalent to setting `DesiredRotation` to 583.2 degrees. In either case, rotation stops after 1,080 ms.

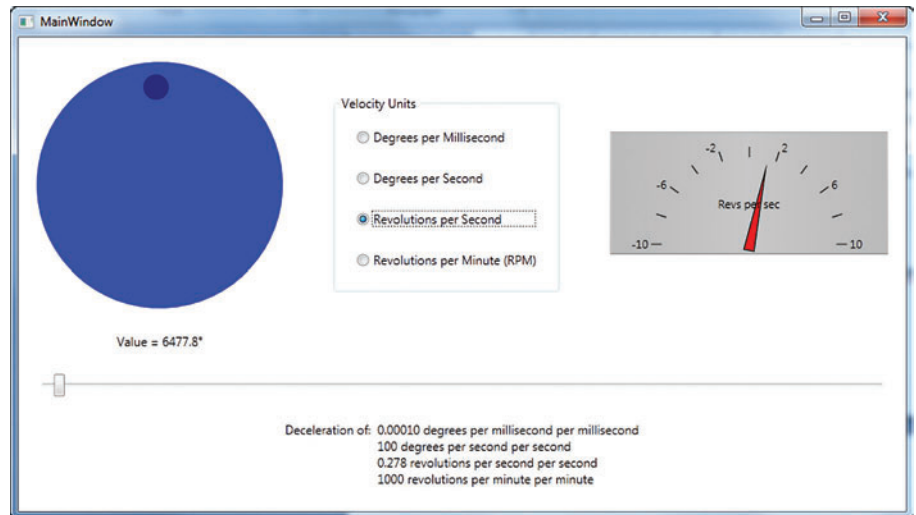


Figure 2 `RotationalInertiaDemo` Program in Action

Experimentation

To get a feel for rotational inertia, I built the `RotationalInertiaDemo` program shown in Figure 2.

The wheel on the left is what you turn with your finger, either clockwise or counter-clockwise. It's very simple: just a `UserControl` derivative with two `Ellipse` elements in a `Grid` with all the `Manipulation` events handled in `MainWindow`.

The `ManipulationStarting` event performs initialization by restricting the manipulation to rotation only, allowing single-finger rotation, and setting the center of rotation:

```
args.IsSingleTouchEnabled = true;
args.Mode = ManipulationModes.Rotate;
args.Pivot = new ManipulationPivot(
    new Point(ctrl.ActualWidth / 2,
              ctrl.ActualHeight / 2), 50);
```

The speedometer at the right is a class called `ValueMeter`, and shows the current velocity of the wheel. If it looks vaguely familiar, it's only because it's an enhanced version of a `ProgressBar` template from an article I wrote for this magazine more than three years ago. The enhancements involved some flexibility with the labels so I could use it to show velocity in four different units. The `GroupBox` in the middle of the window let you select those units.

When your finger is rotating the dial, the meter shows the current angular velocity available from the `Velocities.AngularVelocity` sub-property of the `ManipulationDelta` event arguments. But I found it wasn't possible to funnel the velocity of the dial directly to `ValueMeter`. The result was too jittery. I had to write a little `ValueSmoother` class to perform a weighted average of all the values from the past quarter second. The `ManipulationDelta` event handler also sets a `RotateTransform` object that actually rotates the dial:

```
rotate.Angle += args.DeltaManipulation.Rotation;
```

Finally, the `Slider` at the bottom lets you select a deceleration value. The value of the `Slider` is only read when the finger leaves the dial and a `ManipulationInertiaStarted` event is fired:

```
args.RotationBehavior.DesiredDeceleration = slider.Value;
```

That's the entire body of the `ManipulationInertiaStarted` event handler. During the inertial phase of the manipulation, the velocity

Figure 3 The OnManipulationDelta Event in BoundaryDemo

```
protected override void OnManipulationDelta(
    ManipulationDeltaEventArgs args) {

    FrameworkElement element =
        args.Source as FrameworkElement;
    MatrixTransform xform =
        element.RenderTransform as MatrixTransform;
    Matrix matx = xform.Matrix;
    Vector totalTranslate =
        args.DeltaManipulation.Translation;
    Vector usableTranslate = totalTranslate;

    if (args.IsInertial) {
        double xAttenuation = 0, yAttenuation = 0, attenuation = 0;

        if (matx.OffsetX < 0)
            xAttenuation = -matx.OffsetX;
        else
            xAttenuation = matx.OffsetX +
                element.ActualWidth - mainGrid.ActualWidth;

        if (matx.OffsetY < 0)
            yAttenuation = -matx.OffsetY;
        else
            yAttenuation = matx.OffsetY +
                element.ActualHeight - mainGrid.ActualHeight;

        xAttenuation = Math.Max(0, Math.Min(
            1, xAttenuation / (element.ActualWidth / 2)));
        yAttenuation = Math.Max(0, Math.Min(
            1, yAttenuation / (element.ActualHeight / 2)));
        attenuation = Math.Max(xAttenuation, yAttenuation);

        if (attenuation > 0) {
            usableTranslate.X =
                (1 - attenuation) * totalTranslate.X;
            usableTranslate.Y =
                (1 - attenuation) * totalTranslate.Y;

            if (totalTranslate != usableTranslate)
                args.ReportBoundaryFeedback(
                    new ManipulationDelta(totalTranslate -
                        usableTranslate, 0, new Vector(), new Vector()));

            if (attenuation > 0.99)
                args.Complete();
        }
    }
    matx.Translate(usableTranslate.X, usableTranslate.Y);
    xform.Matrix = matx;

    args.Handled = true;
    base.OnManipulationDelta(args);
}
```

values are much more regular and don't have to be smoothed, so the ManipulationDelta handler uses the IsInertial property to determine when to pass velocity values directly to the meter.

Boundaries and Bounce Back

The most common use of inertia with multi-touch is in moving objects around the screen, such as scrolling through a long list or flicking elements to the side. The big problem is that it's very easy to cause an element to go flying right off the screen!

But in the process of handling that little problem, you'll discover that WPF has a built-in feature that makes manipulation inertia even more realistic. You may have noticed earlier in the ListBoxDemo program that when you let inertia scroll to the end or beginning of the list, the entire window bounces a bit when the ListBox reaches the end. That's also an effect you can get in your own applications (if you want it).

It's very easy to cause an element
to go flying right off the screen!

The BoundaryDemo program consists of just one ellipse with an identity MatrixTransform set to its RenderTransform property sitting in a Grid named mainGrid. Only translation is enabled during the OnManipulationStarting override. The OnManipulationInertiaStarting method sets inertia deceleration like so:

```
args.TranslationBehavior.DesiredDeceleration = 0.0001;
```

That's 0.0001 device-independent units per millisecond squared, or 100 device-independent units per second squared, or about 1 inch per second squared.

The OnManipulationDelta override is shown in **Figure 3**. Notice the special handling when the IsInertial is true. The idea behind the

code is that the translation factors should be attenuated when the ellipse has drifted partially off the screen. The attenuation factor is 0 if the ellipse is within mainGrid, and goes up to 1 when the ellipse is halfway beyond the boundary of mainGrid. The attenuation factor is then applied to the translation vector delivered to the method (called totalTranslate) to calculate usableTranslate, which provides the values actually applied to the transform matrix.

The resulting effect is that the ellipse doesn't stop immediately when it hits the boundary, but slows down dramatically as if encountering some thick sludge.

The OnManipulationDelta override also calls the ReportBoundaryFeedback method, passing to it the unused portion of the translation vector, which is the vector totalTranslate minus usableTranslate. By default, this is processed to make the window bounce a bit as the ellipse slows down, demonstrating the physics principle that for every action there's an opposite and equal reaction.

This effect works best when the velocity on impact is fairly large. If the ellipse has slowed down appreciably, there's a vibration effect that's less satisfactory, but it's something you might want to control in more detail. If you don't want the effect at all (or only want it in some cases), you can simply avoid calling ReportBoundaryFeedback. Or you can handle the ManipulationBoundaryFeedback event yourself. You can inhibit default processing by setting the Handled property of the event arguments to true, or you can choose another approach. I've left an empty OnManipulationBoundaryFeedback method in the program for your experimentation. ■

CHARLES PETZOLD is a longtime contributing editor to MSDN Magazine. He's currently writing "Programming Windows Phone 7" (Microsoft Press), which will be published as a free downloadable e-book in the fall of 2010. A preview edition is currently available through his Web site charlespetzold.com.

THANKS to the following technical experts for reviewing this article:
Doug Kramer and Robert Levy



Devs and Designers Should Be Friends

Developers and designers don't usually get along well, and they need to. I first encountered this antipathy at Tech-Ed 2007 in Barcelona, Spain—the first time I'd seen a whole track of talks aimed at designers. The attendees hated this track, panning it so badly that management had to cancel it after the second day. I observed a few of these talks and found their quality decent. I didn't see any of the usual causes of terrible evaluations, such as demos not working or the speaker being more hungover than usual. Instead, I think that, at that time, the developer community was not willing to hear the fundamental message of the track: that these newfangled Windows Presentation Foundation (WPF) and Silverlight graphical environments required the services of new team members, with different skills but with status equal to the developers.

Developers and designers hold different worldviews.

Their attitude toward each other hasn't improved much. My keynote talk at Dev Days in Amsterdam, Netherlands, in 2008 pleased both communities, but then the designers went off into their own world and didn't rejoin the developers until evening, when the beer started flowing (funny how that works). The same thing happened with my keynote at ReMix in Milan in 2009, except that in Italy they served wine.

Developers and designers hold different worldviews, as do physicians and surgeons. Both pairs make different, incompatible uses of similar environments; both take different, incompatible approaches to similar problems. And both developers and designers are now necessary to any successful client program, as both physicians and surgeons are necessary to any successful medical practice.

The antipathy between developers and designers reminds me of the conflict between cowboys and farmers in "Oklahoma!"—the 1943 Rodgers and Hammerstein stage musical (exclamation point abuse didn't start with Yahoo!). In the song "The Farmer and the Cowman Should Be Friends," Aunt Eller has to pull a gun to force the two groups to mingle at a dance (you can see a YouTube clip at tinyurl.com/pvu93l for an excellent performance).

I'd like to encourage better cooperation between the developer and designer communities—ideally not requiring coercion with firearms. Perhaps I'd update the lyrics:

*Oh, the devs and the designers should be friends,
Oh, the devs and the designers should be friends.
One just bangs out code all day
The other wears a cool beret
But that's no reason why they can't be friends.*

*Software folks should stick together,
Software folks should all be friends.
Some get down with Visual C#
Others get high on Expression Blend.*

*Oh, the devs and the designers should be friends,
Oh, the devs and the designers should be friends.
One of them grinds his tooth enamel
The other knows how to think in XAML
But that's no reason why they can't be friends.*

When I teach WPF or Silverlight at a company, I insist that each class contain both developers and designers. And when I consult with companies on UI projects, I insist that the design team contain both. Usually the designer figures out what would make the user happy and the developer figures out how to implement those ideas efficiently, but a surprising amount of cross-fertilization runs both ways.

For example, at a recent session with a European client, I proposed a classic dialog box for looking up a customer, with a text box for government ID and a calendar control for date of birth, with labels identifying each. The designer said, "Fine, but this operation happens so frequently, how about a search box on the toolbar like Google? The user types in whatever information she has, and we'll take it from there, like a search engine." "Yes, I can parse out all the possible inputs," said the developer, "I have some good reusable classes, it won't take long." (It did, but *sic semper cum geeks*.) "And we can put a prompt string inside the text box, so the user knows what it's for," I added. And—zing!—we had a prototype in front of users for testing in just a few days.

That's what we can accomplish when developers and designers work together, assisted by a designated curmudgeon who keeps the pot stirred. Now start doing it, or I'll butcher that song again. ■

DAVID S. PLATT teaches Programming .NET at Harvard University Extension School and at companies all over the world. He is 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. Contact him at rollthunder.com.

New Version

GrapeCity PowerTools



for Windows Forms
for ASP.NET

SPREAD 5

Award-winning Microsoft® Excel® compatible
spreadsheet components for .NET and ASP.NET

Once Again,
**The Best Grid is
a Spreadsheet.**



Spread is the Winner of this year's CodeProject Members Choice Award for Best Grid Controls

"As before, The Code Project Members Choice Awards reflect the diversity and depth of the tools available to professional developers around the world. This past year has brought even more user interface (UI) tools but also more tools to support all aspects of the development process. Congratulations ..."

Jeff Hadfield,
President of The Code Project (USA)

Benefits

- World's best-selling .NET spreadsheet technology
- Hundreds of Chart styles for data visualization
- Full featured Formula support, including most Excel functions
- Full support for native Microsoft Excel files and data import/export
- Spreadsheet Designers, Quick-start Wizard and Chart Wizards
- Ultimate Dashboard component

**Are you using the Best
~~Grid~~ Spreadsheet?**

Act now and save up to \$300!

GCPowerTools.com/ActNow

1-800-645-5913 / 1-919-460-4551



WE ARE

GrapeCity

Excel  Report  Analyze



The Dashboard Platform for Developers like you

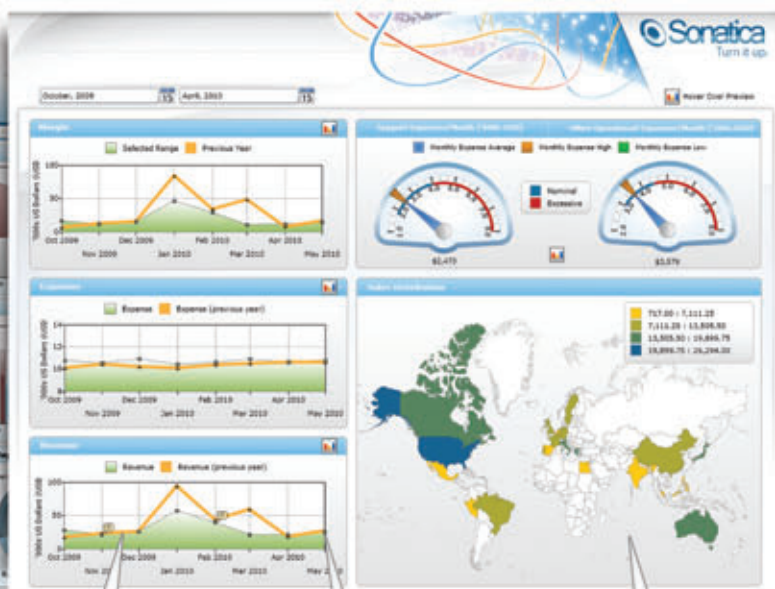
Build more **powerful** and **effective**
dashboards, **faster**.

Support For Numerous
Data Sources



Rapid Dashboard
Development

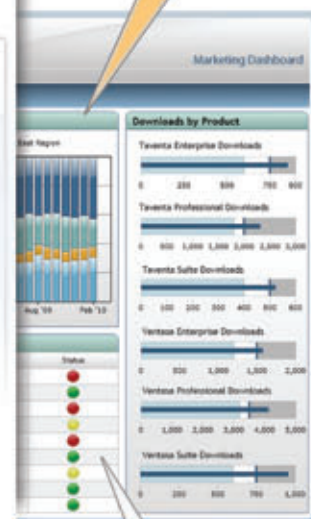
Leverages The Latest
Silverlight Technology



Full Scripting
Capabilities With
DundasScript™

More Data
Visualization Options

Extensible And
Customizable With
An Open API



OLAP Support

Dundas Dashboard is a ground-breaking, extensible solution that uses a revolutionary approach to dashboard creation, providing you with a unified view of key metrics and a new level of strategic insight and decision-making.



Powered by
Microsoft Silverlight



www.dundas.com/dashboard

(416) 467-5100 • (800) 463-1492

Silverlight is a trademark of Microsoft Corporation in the United States and/or other countries.