The Microsoft Journal for Developers

msdn magazine

MANAGED EXTENSIBILITY FRAMEWORK
Building Composable Apps in .NET 4 with the Managed Extensibility Framework
Glenn Block page 32

ROBOTICS
Writing and Testing VPL Services for Serial Communication
Trevor Taylor page 42

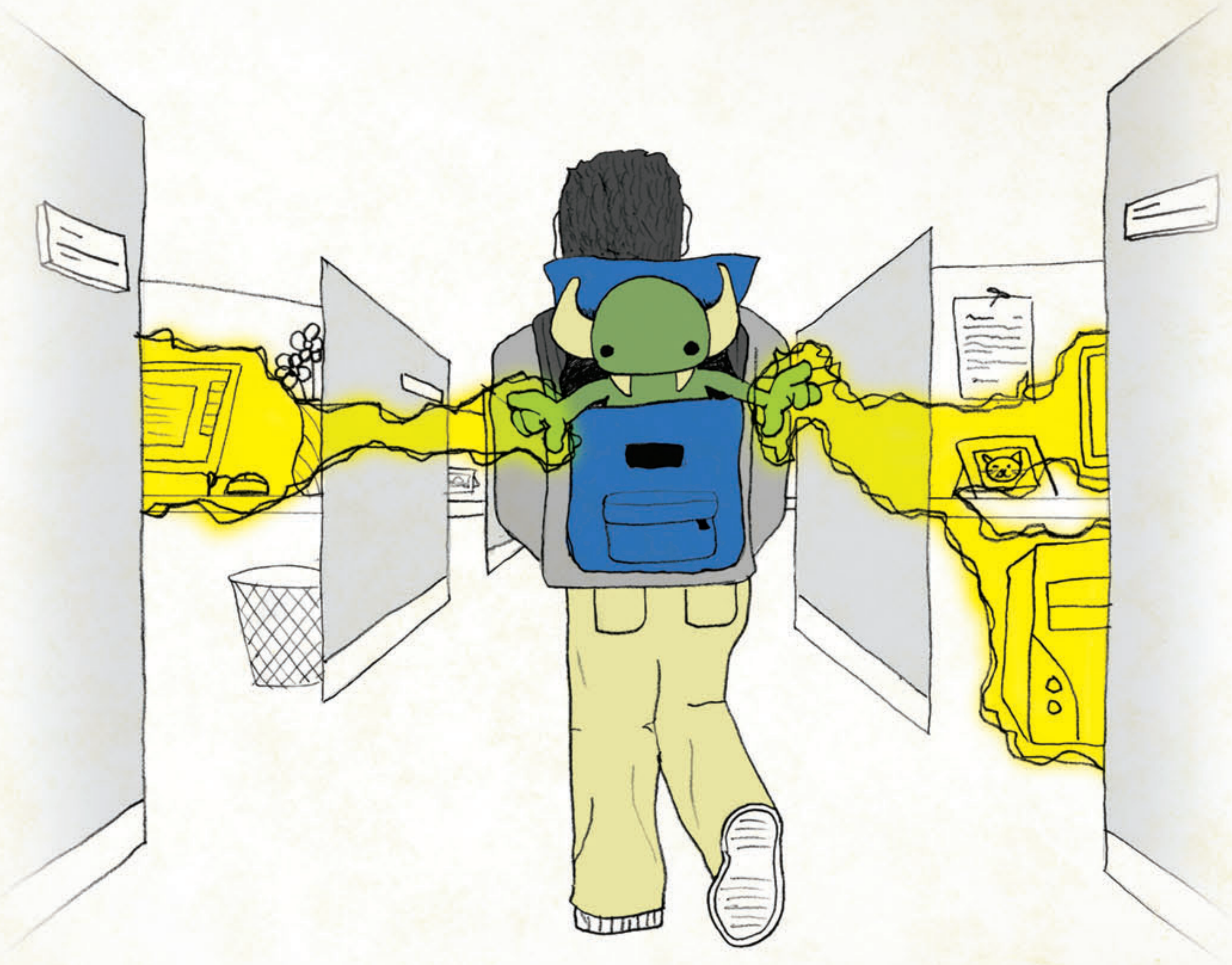
DYNAMIC .NET
Creating Interactive Bing Maps with Silverlight and IronRuby
Ashish Ghoda page 49

CLOUD COMPUTING
Windows Azure Platform for Enterprises
Hanu Kommalapati page 58

BEYOND MVP
Extending the MVP Pattern to Simplify UI Architecture
Zhe Ma page 70

COLUMNS

FEBRUARY 2010 VOL 25 NO 2



ADD SOME EXTREME TO YOUR TEAM

Infuse your team with the power to create user interfaces with extreme functionality, complete usability and the "wow-factor!" with NetAdvantage® in your .NET development toolbox. Featuring the most powerful and fastest data grids on the market for Windows Forms, ASP.NET, Silverlight and WPF, it'll be like having the strength of 10 developers on every desktop. Go to infragistics.com/killerapps to find out how you and your team can start creating your own Killer Apps.

Infragistics Sales 800 231 8588
Infragistics Europe Sales +44 (0) 800 298 9055
Infragistics India +91-80-6785-1111

Infragistics®

KILLER APPS. NO EXCUSES.

All cats can purr,
Syncfusion™
Deliver innovation with ease
www.syncfusion.com
1 888-9DOTNET

only lions roar.

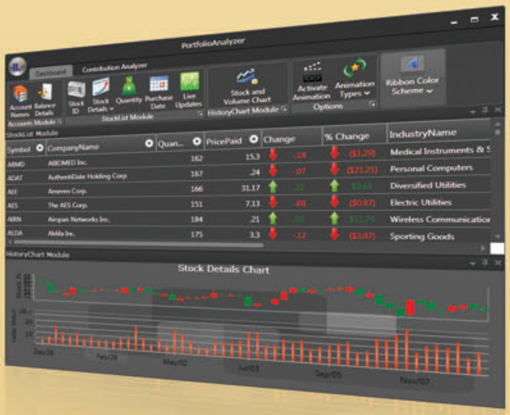
What does your grid do?

Although many grids share basic features, only Syncfusion's WPF grid has the power and performance to set your application apart. Our grid builds on years of Syncfusion's expertise with high-performance scenarios.

- Support for billions of rows
- Unmatched scrolling performance with large data sets
- Wide variety of styling options

Syncfusion Essential Grid WPF is the fastest WPF grid on the market.

Experience it for yourself at
www.syncfusion.com/wpf-gridperformance

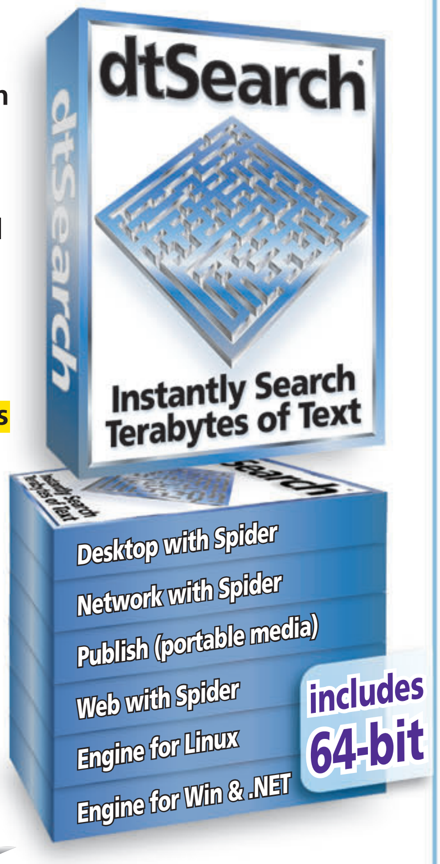


Symbol	CompanyName	Quan...	PricePaid	Change	% Change	IndustryName
ABMD	ABIOMED Inc.	162	15.3	-.18	($1.29)	Medical Instruments & S
ADAT	AuthentiDate Holding Corp	167	.24	-.07	($21.21)	Personal Computers
AEE	Ameren Corp.	166	31.17	.22	$0.65	Diversified Utilities
AES	The AES Corp.	151	7.13	-.08	($0.97)	Electric Utilities
AIRN	Airspan Networks Inc.	184	.21	.03	$11.76	Wireless Communicatior
ALDA	Aldila Inc.	175	3.3	-.12	($3.97)	Sporting Goods

ASP.NET MVC   ASP.NET   Windows Forms   Back Office   Silverlight   WPF   Business Intelligence

Instantly Search Terabytes of Text

- 25+ full-text and fielded data search options (with Unicode support for hundreds of international languages)
- Built-in file parsers / converters highlight hits in popular file types
- Spider supports static and dynamic web data; highlights hits with links, formatting and images intact
- API supports .NET, C++, Java, SQL, etc. .NET Spider API



The Smart Choice for Text Retrieval® since 1991

"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*

See www.dtsearch.com for hundreds more reviews, and hundreds of developer case studies

Fully-Functional Evaluations

1-800-IT-FINDS • www.dtsearch.com

Microsoft msdn magazine FEBRUARY 2010 VOLUME 25 NUMBER 2

LUCINDA ROWLEY Director
DIEGO DAGUM Editorial Director
KERI GRASSL Site Manager

KEITH WARD Editor in Chief
TERRENCE DORSEY Technical Editor
DAVID RAMEL Features Editor
WENDY GONCHAR Managing Editor

SCOTT SHULTZ Creative Director
JOSHUA GOULD Art Director
ALAN TAO Senior Graphic Designer

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

RedmondMediaGroup

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

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

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

1105 MEDIA INC

Neal Vitale President & Chief Executive Officer
Richard Vitale Senior Vice President & Chief Financial Officer
Michael J. Valenti Executive Vice President
Christopher M. Coates Vice President, Finance & Administration
Abraham M. Langer Vice President, Digital Media, Audience Marketing
Erik A. Lindgren Vice President, Information Technology & Web 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 U.S. funds: U.S. $35; Canada $45; International $60. 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, e-mail MSDNmag@1105service.com or call 847-763-9560. **POSTMASTER:** Send address changes to *MSDN Magazine*, P.O. Box 1081, Skokie, IL 60076-8081. Canada Publications Mail Agreement No: 40612608. Return Undeliverable Canadian Addresses to Circulation Dept. or Bleuchip International, P.O. Box 25542, London, ON N6C 6B2.

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

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

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

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

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

BPA WORLDWIDE
Printed in the USA

Your best source for software development tools!

programmer's paradise®

LEADTOOLS Recognition SDK v16.5

by LEAD Technologies

Develop robust 32/64 bit document imaging and recognition functionality into your applications with accurate and high-speed multi-threaded Forms, OCR, OMR, and 1D/2D barcode engines.

- Supports text, OMR, image, and barcode fields
- Auto-registration and clean-up to improve recognition results
- Provided as both high and low level interface
- Includes comprehensive confidence reports to assess performance

Paradise #
L05 26301A01
$3,214.99

programmers.com/LEAD

Multi-Edit 2008

by Multi Edit Software

Multi-Edit 2008 delivers, a powerful IDE, with its speed, depth, and support for over 50 languages. Enhanced search functions include Perl 5 Regular Expressions and definable filters. Supports large DOS/Windows, UNIX, binary and Mac files. File Sync Integration for: Delphi 6, 7, 2005, C++ Builder 6, BDS 2006 and RadStudio 2007, VB 6, VC 6, VS 2003, 2005 and 2008. Includes file compare, code beautifying, command maps, and much more.

1-49 Users
Paradise #
A30 01201A01
$179.99

programmers.com/multiedit

VMware vSphere

Put time back into your day.

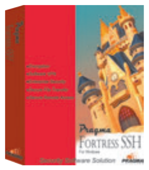
Your business depends on how you spend your time. You need to manage IT costs without losing time or performance. With proven cost-effective virtualization solutions from VMware, you can:

- Increase the productivity of your existing staff three times over
- Control downtime—whether planned or not
- Save more than 50% on the cost of managing, powering and cooling servers

Make your time (and money) count for more with virtualization from VMware.

VMware Advanced Acceleration Kit for 6 processors
Paradise #
V55 78101A01
$9,234.99

programmers.com/vSphere



Pragma Fortress SSH—SSH Server & Client for Windows

by Pragma Systems

Contains SSH, SFTP, SCP servers and clients 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
- Offers FIPS mode
- 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

Unify SQLBase Embedded Database

by Unify

SQLBase is an easy to deploy database for building desktop and Web applications in server environments without a DBA.

- Small footprint
- Zero admin deployment
- GUI admin
- Embed it and forget it

For applications with encrypted security, we offer SQLBase Treasury.

Reduce your database costs and simplify maintenance with Unify SQLBase.

NEW RELEASE!

Paradise #
C15 03101A01
$143.99

programmers.com/unify

TX Text Control 15.1

Word Processing Components

NEW RELEASE!

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

Professional Edition
Paradise #
T79 02101A02
$848.99

Download a demo today.

programmers.com/theimagingsource

FarPoint Spread for Windows Forms

Now with Charting! The Best Grid is a Spreadsheet. Give your users the look, feel, and power of Microsoft® Excel®, without needing Excel installed on their machines. Join the professional developers around the world who consistently turn to FarPoint Spread to add powerful, extendable spreadsheet solutions to their COM, ASP.NET, .NET, BizTalk Server and SharePoint Server applications.

- World's #1 selling development spreadsheet
- Read/Write native Microsoft Excel Files
- Charts with 85 new chart types
- Fully extensible models
- Royalty-free, run-time free

NEW VERSION 5!

Paradise #
F02 01101A01
$936.99

programmers.com/grapecity

Enterprise Architect 7.5

Visualize, Document and Control Your Software Project

by Sparx Systems

Enterprise Architect is a comprehensive, integrated UML 2.1 modeling suite providing key benefits at each stage of system development. Enterprise Architect 7.5 supports UML, SysML, BPMN and other open standards to analyze, design, test and construct reliable, well understood systems. Additional plug-ins are also available for Zachman Framework, MODAF, DoDAF and TOGAF, and to integrate with Eclipse and Visual Studio 2005/2008.

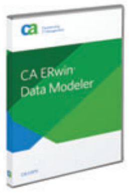
Corporate Edition
1-4 Users
Paradise #
SP6 03101A02
$182.99

programmers.com/sparxsystems

CA ERwin r7.3

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
$3,951.99

programmers.com/ca

FREE ON DEMAND WEBINAR SERIES: MORE Maximum Data Modeling with CA ERwin 7.3

In our last webinar series, we looked at CA ERwin's core functionality. In this second series, we provide a grounding in how CA ERwin r7.3's new features help you with Master Data Management, Metadata Management, Data Warehousing, Data Governance and Business Intelligence.

There are six sessions in the on demand series:

- What's New in CA ERwin 7.3
- MDM (Master Data Management) with CA ERwin and Data Profiling tool
- Collaborative model management with CA ERwin ModelManager
- Validate the integrity of your model with CA ERwin Validator
- Reporting: Crystal Reports, PDF, HTML
- SAPHIR Option: light at the end of the metadata tunnel

VIEW TODAY: programmers.com/MDM_2009

AdminStudio & Application Virtualization Pack

by Flexera Software

One Application Software Deployment Tool for Reliable MSI Packaging, Application Virtualization, and Windows 7 Migration. Top choice of Microsoft®, Novell®, LANDesk® and other software management solutions. Cut MSI packaging time by up to 70%, Deploy software to desktops with 99% success or better. AdminStudio is the only MSI packaging solution to support multiple virtualization formats, including Microsoft® App-V™, VMware® ThinApp™ and Citrix® XenApp™.

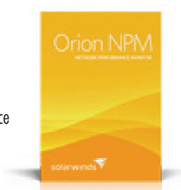
Professional Upgrade from any Active AS Pro + Silver Mtn
Paradise #
I21 09401S05
$4,228.99

programmers.com/flexera

Orion Network Performance Monitor

by Solarwinds

Orion Network Performance Monitor is a comprehensive fault and network performance management platform that scales with the rapid growth of your network and expands with your network management needs. It offers out-of-the-box network-centric views that are designed to deliver the critical information network engineers need. Orion NPM is the easiest product of its kind to use and maintain, meaning you will spend more time actually managing networks, not supporting Orion NPM.



Paradise #
S4A 08201E02
$4,606.99

programmers.com/solarwinds

866-719-1528 **programmersparadise.com**

Prices subject to change. Not responsible for typographical errors.

Not Your Father's *MSDN*

As *MSDN Magazine* Editorial Director Diego Dagum mentioned last December, changes are coming to your favorite developer publication. This month is when they start to kick in.

To begin with, we've made some changes to our column lineup. The changes will be implemented gradually over the next few months to a year. The biggest change is that many of the columns you now enjoy on a quarterly or other irregular basis will start to appear more often—most of them monthly.

We'll also be starting a few new monthly columns with this issue. And it begins with the end in this case—specifically, the new back-page column. February is the unveiling of "Don't Get Me Started," by (literal) software legend David S. Platt.

The literal part is that Platt was named a Software Legend by Microsoft in 2002. You may know him better, however, by his landmark book, "Why Software Sucks" (Addison-Wesley, 2006). It's one of 11 software development tomes he has penned, but is by far the most well-known. The title of that book gives good insight into who Platt is, and how he writes. There are no sacred cows with him; only heifers ripe for the slaughterhouse. His humor shines through, but it's never without a serious point.

Beyond those strengths, what really makes Platt good is his ability to think like a non-programmer; to put himself in the place of the end-user trying to muddle through lousy UI design, cryptic error messages and other development no-nos that leave folks like my Mom in a state of confused depression.

His first column is a perfect example of that. Floating menu bars in (earlier versions of) Microsoft Word? How did that one get through QA? Word's auto-correct feature, on the other hand, is a thing of beauty, something that's useful for my Mom—and me.

(Side note: I was having breakfast recently with two friends of mine who are veteran developers and longtime *MSDN Magazine* readers. I told them Platt was coming onboard. They knew his work well, and were ecstatic about it. That's Miles Davis to my ears.)

Another new column kicking off is from an old friend of yours. Charles Petzold is just as legendary as David Platt, having written numerous programming books over the years, teaching generations of coders how to do their jobs better. He's one of the original contributors to this magazine; in fact, he first appeared in Volume 1, Issue 1 in October 1986, when the magazine was called *Microsoft Systems Journal*.

More recently, Petzold has been a contributor to the Foundations column, but his stuff is so good that having him in the magazine three or four times per year wasn't enough for me. So, after much begging, pleading and threats of physical and psychological torture, he agreed to go monthly with a new column called UI Frontiers.

Petzold's vision for the column is that it will, in his words, "Explore the new presentation and UI capabilities enabled by the XAML-based programming environments of the Windows Presentation Foundation and Silverlight, with a special emphasis on cross-platform compatibility. Modern UI design is characterized by an extensive use of 2-D and 3-D graphics and animation, but also (where appropriate) sound, music, speech and touch."

Also going monthly is Test Run, by Dr. James McCaffrey. I had the pleasure of meeting Dr. McCaffrey recently in Redmond, Wash., at Microsoft's headquarters. He's just a great guy, and gave me and my colleagues tremendous insight into the readership of *MSDN Magazine*, and many of the issues that are near and dear to your hearts.

He's also a brilliant writer and developer. It probably gives him headaches to talk down to my level of comprehension, but with the help of him and others, I'm getting better.

More changes are in the offing, including more monthly columns, a re-focus of content to better serve our day-to-day developer audience, and other goodies that we'll be launching in the near future. I hope you like them, and will tell me about it. I'm at kward@1105media.com.

[illegible]

Visit us at msdn.microsoft.com/magazine. Questions, comments or suggestions for *MSDN Magazine*? Send them to the editor: kward@1105media.com.

"Best Project Management Software"
Readers' Choice - 4 years in a row

READERS' CHOICE 2006 asp.net PRO WINNER · READERS' CHOICE 2007 asp.net PRO WINNER · READERS' CHOICE 2008 asp.net PRO WINNER · READERS' CHOICE 2009 asp.net PRO WINNER

FREE! 1-User License

Axosoft customers have saved

$1,985,575,8

by shipping software OnTime!

We've saved dev teams a couple of billion bucks... but we're not making a big deal about it.

See how at axosoft.com

Project Management • Bug Tracking • Scrum

Track Everything
Track everything from bugs to software requirements, team member tasks, and more. 360-degree visibility.

Team Wiki
Finally, one place to collaborate on projects in a clean, organized, hierarchical structure.

Help Desk
Use OnTime for tracking Help Desk incidents - even monitor email accounts to generate help tickets.

Scrum
Harness the power of scrum within the OnTime client. Use Sprints, Burndowns, Trending Charts and more.

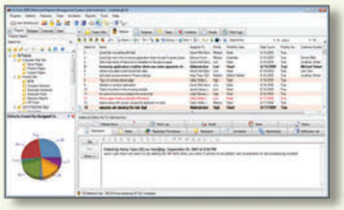
Workflow
Define a hyper-fast, lightweight, agile workflow or a robust process-enforcement system.

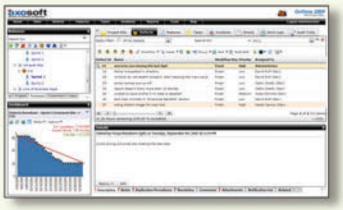
Notifications
Ensure team members are in synch with automatic email notifications and alerts.

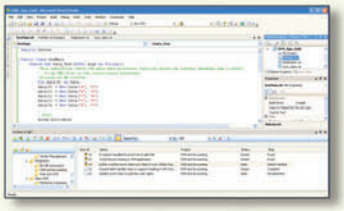
Customize
Unlimited custom fields, customize all field names, and decide when fields are visible, editable or required.

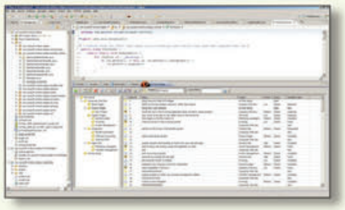
Security
Secure every aspect of the system by project, user roles, and even fields.

Starts at: FREE
1-user: free.
5-user teams start at: $395.
Visit axosoft.com for an exact quote.

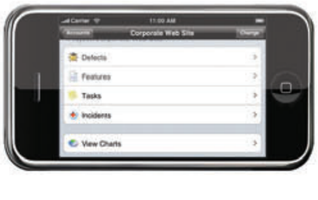

Windows


Web


Visual Studio


Eclipse

axosoft
Software for Software Development™



800.653.0024
www.axosoft.com

WINDOWS 7 IS HERE... VISUAL STUDIO 2010 IS COMING
PREPARE FOR THE WORLD OF
TOMORROW
WITH STUDIO ENTERPRISE 2009 v3

100'S OF CONTROLS, 7 PLATFORMS, FEATURING

WinForms • WPF • ASP.NET • Silverlight • iPhone • Mobile • ActiveX

Grids • Charts • Reports • Schedules • Menus • Toolbars • Ribbon • Data Input • Editors • PDF

DEVELOPER APPROVED FOR 20+ YEARS!

	Location	Deposits	Withdrawls
09	NEW YORK CITY, NY	3307	
009	DUBAI, INDIA		1593
010	PITTSBURGH, PA		3977
009	BEIJING, CHINA		4745
08	DENVER, CO		3294
09	CHICAGO, IL	3587	
0	PARIS, FRANCE		4222

4 General Checking	$1108.50
Savings	$11108.50
a	$3108.30
e Funds	$9104.10
ncy Funds	$5108.80
	$4878.70

DOWNLOAD YOUR FREE TRIAL AT
WWW.COMPONENTONE.COM/PREPARE

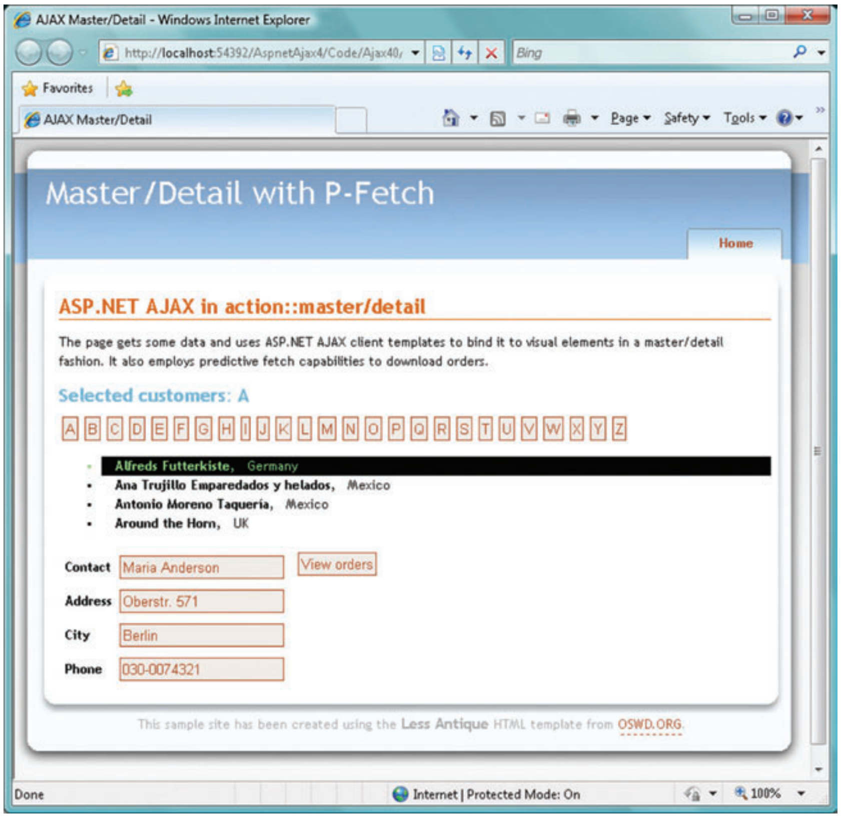
ComponentOne **ComponentOne Sales:** 1.800.858.2739 or 1.412.681.4343

Predictive Fetch with jQuery and the ASP.NET Ajax Library

Last month I discussed the implementation of master-detail views using the new features coming with the ASP.NET Ajax Library. The list of new features includes a syntax for client-side live data binding and a rich rendering component, exemplified by the DataView client control. By putting these features together, you can easily build nested views to represent one-to-many data relationships.

In the ASP.NET Ajax Library, the mechanics of master-detail views are largely defined in the logic of the DataView component and in the way the component handles and exposes its events.

This month I'll go one step further and discuss how to implement a common and popular AJAX design pattern—predictive fetch—on top of the ASP.NET Ajax Library. Basically, I'll extend last month's example—a relatively standard drill-down view into customer details—to automatically and asynchronously download and display related orders, if any exist. In doing so, I'll touch on some jQuery stuff and take a look at the new jQuery integration API in the ASP.NET Ajax Library. Without further ado, let's review the context and build a first version of the example.


AJAX Master/Detail - Windows Internet Explorer
http://localhost:54392/AspnetAjax4/Code/Ajax40/
Bing
Favorites
AJAX Master/Detail
Page
Safety
Tools
Home
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
View orders
Done
Internet | Protected Mode: On
100%

Master/Detail with P-Fetch

ASP.NET AJAX in action::master/detail

The page gets some data and uses ASP.NET AJAX client templates to bind it to visual elements in a master/detail fashion. It also employs predictive fetch capabilities to download orders.

Selected customers: A

- Alfreds Futterkiste, Germany
- Ana Trujillo Emparedados y helados, Mexico
- Antonio Moreno Taqueria, Mexico
- Around the Horn, UK

Contact	Maria Anderson
Address	Oberstr. 57
City	Berlin
Phone	030-0074321

This sample site has been created using the Less Antique HTML template from OSWD.ORG

Figure 1 **The Initial Stage of the Sample Application**

The Demo to Expand

Figure 1 shows the application scenario on top of which I'll add predictive fetch capabilities.

The menu bar allows the user to filter customers by initial. Once a selection is made, a smaller list of customers is displayed through an HTML bulleted list. This is the master view.

Each rendered item has been made selectable. Clicking on one causes the details of the customer to be displayed in the adjacent detail view. This is where I left off last month. As you can see in **Figure 1**, the user interface now shows a button to view orders. I proceed from here onward.

The first decision to be made is architectural and relates to the use-case you are considering. How would you load order information? Is it information you have downloaded already along with customer information? Are orders attached to customers? Is lazy loading an option here?

The code we're considering is expected to run on the client side, so you can't rely on lazy loading facilities built into some object/relational modeling (O/RM) tools such as Entity Framework or NHibernate. If orders are to be lazy-loaded, then any code is up to you. On the other hand, if you can assume that orders are already available on the client—that is, orders have been downloaded along with the customer information—then you're pretty much done. All you need to do is bind orders data to some HTML template and go.

Obviously, things get much more interesting if lazy loading is what you want. Let's work out this scenario, then.

As a side note, you should know that lazy loading is fully supported if you get your data through the AdoNetDataContext object. (I'll cover this in future articles.) For more information,

This column is based on a pre-release version of the ASP.NET Ajax Library. All information is subject to change.

Send your questions and comments for Dino to cutting@microsoft.com.

Code download available at code.msdn.microsoft.com/mag201002CuttingEdge.

/n software

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

Microsoft .NET, | Java | ActiveX | C++ | Delphi | C++ Builder | PHP | Objective-C | Windows | Windows Mobile | Mac OS X | Linux/Unix | iPhone be sure to look at asp.net/ajaxlibrary/Reference.Sys-Data-AdoNetServiceProxy-fetchDeferredProperty-Method.ashx.

A New Way to Load Script Libraries

For years, Web developers had been left alone to figure out which script files a page would need. That wasn't a daunting task, because the limited amount of simple JavaScript code that was used made it quite easy to check whether a required file was missing. Increased amounts of complicated JavaScript code in Web pages introduced the problem of splitting scripts among distinct files and, subsequently, referring to them properly to avoid nasty runtime "undefined object" errors.

You can easily build nested views to represent one-to-many data relationships.

Many popular JavaScript libraries have been providing facilities in this regard for years now. For example, the jQuery UI library has a modular design and allows you to download and link only the pieces you really need. This same capability is also offered by the scripts that make up the ASP.NET Ajax Library. The script loader, however, is something more.

The script loader provides a number of extra services and builds on the partitioning of large script libraries into smaller pieces. Once you tell the loader about the libraries you're interested in, you delegate to the loader any tasks related to the correct ordering of required files. The script loader loads all required scripts in parallel and then executes them in the right order. In this way, the loader saves you from any "missing object" exceptions and provides the fastest way to handle scripts. All you need to do is list the scripts you want.

Hey, wait a moment. If I have to list all scripts I need, what are the benefits of using a loader? Well, what the loader requires is nowhere near to what is required in the well-known process of linking assemblies to a project. You link assembly A and let the Visual Studio 2008 loader figure out any static dependencies. Here's a code snippet that shows how to deal with the script loader:

```
Sys.require([Sys.components.dataView, Sys.scripts.jQuery]);
```

The Sys.require method takes an array of references to the scripts you want to link to your page. In the preceding example, you are instructing the loader to take care of two scripts—dataView and jQuery.

As you can see, however, the call made to the method Sys.require doesn't include any Web server path to any physical .js files. Where's the path, then?

Scripts that will work with the ASP.NET Ajax Library loader are required to define themselves to the loader and inform it when they load completely. Registering a script with the loader doesn't cause any round trip, but is simply a way to let the loader know that it may be called upon to manage a new script. **Figure 2** includes an excerpt from MicrosoftAjax.js that shows how jQuery and jQuery.Validate are registered with the loader.

Of course, you can use this approach with custom scripts and client controls as well. In that case, you need to reference the loader-specific definition of the script in addition to your actual script. A loader-specific definition includes release and debug server paths of the script, a public name used to reference it, dependencies and an expression to be evaluated in order to test whether the library loaded correctly.

In order to use the script loader component, you need to reference a new JavaScript file named start.js. Here's an excerpt from the sample application that uses a mix of old and new script-loading techniques:

```
<asp:ScriptManagerProxy runat="server" ID="ScriptManagerProxy1">
    <Scripts>
        <asp:ScriptReference Path="~/Scripts/Ajax40/Preview6/start.js"/>
        <asp:ScriptReference Name="MicrosoftAjax.js"
                       Path="~/Scripts/MicrosoftAjax.js"/>
        <asp:ScriptReference Path=
                                  "~/Scripts/MicrosoftAjaxTemplates.js"/>
     <asp:ScriptReference Path="~/MasterDetail4.aspx.js"/>
    </Scripts>
</asp:ScriptManagerProxy>
```

You reference the start.js file using a classic <script> element. Other scripts can be referenced using the ScriptManager control, plain <script> elements or the Sys.require method. As you can see from the code snippet above, there's no reference to the jQuery library. In fact, the jQuery library is referenced programmatically

Figure 2 **Register jQuery and jQuery.Validate with the Script Loader**

```
loader.defineScripts(null, [
     { name: "jQuery",
       releaseUrl: ajaxPath + "jquery/jquery-1.3.2.min.js",
       debugUrl: ajaxPath + "jquery/jquery-1.3.2.js",
       isLoaded: !!window.jQuery
     },
     { name: "jQueryValidate",
       releaseUrl: ajaxPath +
                     "jquery.validate/1.5.5/jquery.validate.min.js",
       debugUrl: ajaxPath + "jquery.validate/1.5.5/jquery.validate.js",
       dependencies: ["jQuery"],
       isLoaded: !!(window.jQuery && jQuery.fn.validate)
     }
    ]);
```

Figure 3 **The Code to Fetch Orders**

```
function fetchOrders(elem)
{
    // Set the customer ID
    var id = elem["commandargument"];
    currentCustomer = id;

    // Check the jQuery cache first
    var cachedInfo = $('#viewOfCustomers').data(id);
    if (typeof (cachedInfo) !== 'undefined')
        return;

    // Download orders asynchronously
    $.ajax({
        type: "POST",
        url: "/mydataservice.asmx/FindOrders",
        data: "id=" + id,
        success: function(response) {
            var output = response.text;
            $('#viewOfCustomers').data(id, output);
            if (id == currentCustomer)
                $("#listOfOrders0").html(output);
        }
    });
}
```

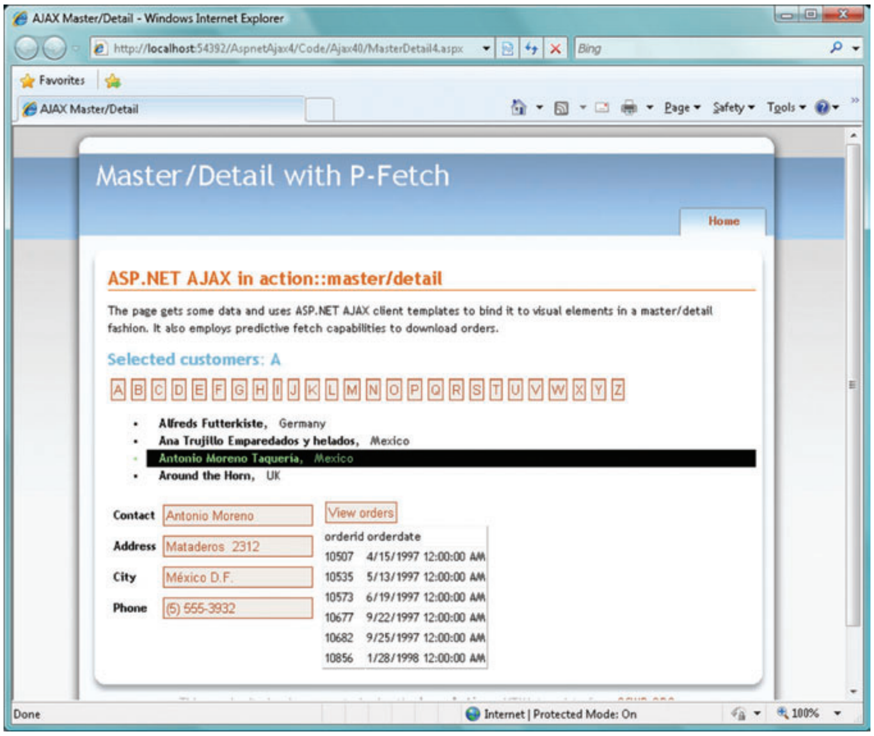
from the page-specific JavaScript file linked via the ScriptManager.

Another interesting feature in the ASP.NET Ajax Library is the availability of jQuery features through the Sys namespace and, conversely, the exposure of Microsoft client components as jQuery plugins. This means, for example, that you can register an event handler for the ready event—a typical jQuery task—using the Sys.onReady function, as shown here:

```
    Sys.onReady(
      function() {
         alert("Ready...");
      }
  );
```

Given all these new features, the typical startup of a JavaScript file to be used as an extension of a Web page looks like the following:

```
// Reference external JavaScript files
Sys.require([Sys.scripts.MicrosoftAjax,
             Sys.scripts.Templates,
             Sys.scripts.jQuery]);
Sys.onReady(
    function() {
        // Initialize scriptable elements
        // of the page.
    }
);
```


AJAX Master/Detail - Windows Internet Explorer

http://localhost:54392/AspnetAjax4/Code/Ajax40/MasterDetail4.aspx

Master/Detail with P-Fetch

ASP.NET AJAX in action::master/detail

The page gets some data and uses ASP.NET AJAX client templates to bind it to visual elements in a master/detail fashion. It also employs predictive fetch capabilities to download orders.

Selected customers: A

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

- Alfreds Futterkiste, Germany
- Ana Trujillo Emparedados y helados, Mexico
- Antonio Moreno Taqueria, Mexico
- Around the Horn, UK

Contact	Antonio Moreno
Address	Mataderos 2312
City	México D.F.
Phone	(5) 555-3932

View orders

orderid	orderdate
10507	4/15/1997 12:00:00 AM
10535	5/13/1997 12:00:00 AM
10573	6/19/1997 12:00:00 AM
10677	9/22/1997 12:00:00 AM
10682	9/25/1997 12:00:00 AM
10856	1/28/1998 12:00:00 AM

Done

Internet | Protected Mode: On

Figure 4 **Fetching and Displaying Orders**

An even simpler approach is possible, however. You can use Sys.require to load a control like a DataView instead of the files that implement it. The script loader would load these files automatically based on the dependencies defined for the DataView. Let's focus on predictive fetch.

Handling the Customer Selection

To obtain the user interface shown in **Figure 1**, you use HTML templates and attach data to data-bound placeholders using the DataView component. Customer details are automatically shown due to DataView-based data binding when you click on a listed customer. Orders, however, are not bound directly via the DataView. This is because of the requirements we set at the beginning of the article—by design, orders are not downloaded with the customer information.

To fetch orders, therefore, you need to handle the change of selection within the template associated with the DataView. Currently, the DataView doesn't fire a selection-changed event. The DataView does provide great support for master-detail scenarios, but much of it happens automatically, even though you can create custom commands and handlers (see asp.net/ajaxlibrary/Reference.Sys-UI-DataView-onCommand-Method.ashx). In particular, you set the sys:command attribute to "select" on any clickable elements that can trigger the details view, as shown here:

```
<li>
   <span sys:command="Select"
         id="itemCustomer"
         class="normalitem">
   <span>{binding CompanyName}</span>
   <span>{binding Country}</span>
   </span>
</li>
```

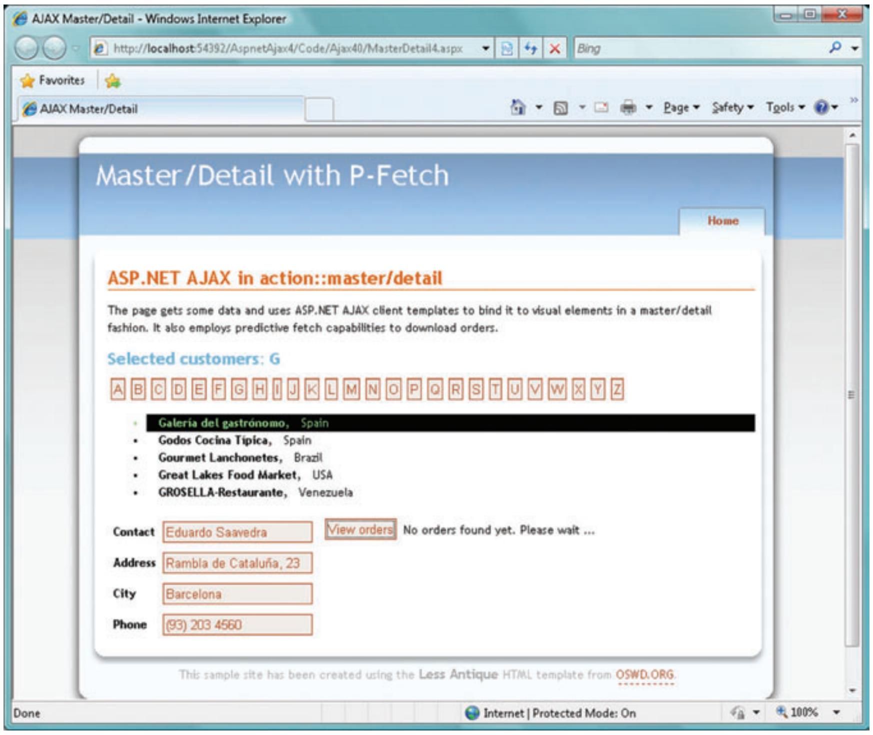
When the element is clicked, it fires an onCommand event within the DataView and, as a result, the content of the selectedData property is updated to reflect the selection. Subsequently, any parts of the template that are bound to selectedData are refreshed. Data binding, however, entails updating displayed data, not executing any code.

As mentioned, when a command is fired within the DataView, the onCommand event is raised internally. As a developer, you can register your own handler for the event. Unfortunately, at least with the current prerelease version of the DataView component, the command handler is invoked before the selected index property is updated. The net effect is that you can intercept when the details view is about to show, but you have no clue about the new content being shown. The only goal of the event seems to be giving developers a way to prevent the change of selection should some critical conditions not be verified.

An approach that works today, and that will continue working in the future regardless of any improvements to the DataView component, is the following. You attach an onclick handler to any clickable elements of the master view and bind an extra attribute to contain any key information that is helpful. Here's the new markup for the repeatable portion of the master view:

```
<li>
   <span sys:command="Select"
         sys:commandargument="{binding ID}"
         onclick="fetchOrders(this)"
         id="itemCustomer"
         class="normalitem">
   <span>{binding CompanyName}</span>
   <span>{binding Country}</span>
   </span>
</li>
```

The markup presents two changes. First, it now includes a new sys:commandargument attribute; second, it has a handler for the click event. The sys:commandargument attribute contains the ID


AJAX Master/Detail - Windows Internet Explorer

http://localhost:54392/AspnetAjax4/Code/Ajax40/MasterDetail4.aspx

Master/Detail with P-Fetch

Home

ASP.NET AJAX in action::master/detail

The page gets some data and uses ASP.NET AJAX client templates to bind it to visual elements in a master/detail fashion. It also employs predictive fetch capabilities to download orders.

Selected customers: G

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

- Galería del gastrónomo, Spain
- Godos Cocina Típica, Spain
- Gourmet Lanchonetes, Brazil
- Great Lakes Food Market, USA
- GROSELLA-Restaurante, Venezuela

Contact: Eduardo Saavedra View orders No orders found yet. Please wait ...

Address: Rambla de Cataluña, 23

City: Barcelona

Phone: (93) 203 4560

This sample site has been created using the Less Antique HTML template from OSWD.ORG.

Done Internet | Protected Mode: On 100%

Figure 5 **Orders are Not Yet Available**

of the customer that has been selected. The ID is emitted through data binding. The attribute where you park the ID doesn't have to be sys:commandargument necessarily; you can use any custom attribute as well.

The click handler is responsible for fetching orders according to whatever loading policy you have set. **Figure 3** shows the source code of the orders loader.

The fetchOrders function receives the DOM element that was clicked. First, it retrieves the value of the agreed attribute that contains the customer ID. Next, it checks whether orders already exist in the jQuery client cache. If not, it finally proceeds with an asynchronous download. It uses a jQuery AJAX method to arrange a POST request to a Web service. I'm assuming in this example that the Web service employs the "HTML Message" AJAX pattern and returns plain HTML ready to be merged with the page. (Note that this is not necessarily the best approach and mostly works in legacy scenarios. From a pure design perspective, querying an endpoint for JSON data would generate a much lighter payload.)

If the request is successful, the orders markup is first cached and then displayed where expected (see **Figure 4**).

Figure 4 shows only a screenshot and doesn't really explain what's going on. As you click to select a customer to drill down, the request for orders fires asynchronously. Meanwhile, the details of the customer are displayed. As you may recall, there's no need to download customer information on demand, as that information is downloaded in chunks as the user clicks on the high-level menu of initials.

Downloading orders may take a while and is an operation that doesn't give (or require) any feedback to the user. It just happens, and it's completely transparent to the user. The whole point of the predictive fetch pattern is that you fetch information in advance that the user may possibly request. To represent a true benefit, this feature has to be implemented asynchronously and, from a usability perspective, it's preferable if it's invisible to the user.

Let's focus on the most common tasks a user would perform on the user interface in **Figure 4**. A user would typically click to select a customer. Next, the user would likely spend a few moments reading the displayed information. As the user views the display, the orders for the selected customer are silently downloading.

The user may, or may not, request to view orders immediately. For example, the user may decide to switch to another customer, read the information and then switch back to the first one, or perhaps navigate to yet another. In any case, by simply clicking to drill down on a customer, the user triggers the fetch of related orders.

What happens to downloaded orders? What would be the recommended way of dealing with them upon download?

Dealing with Fetched Orders

Frankly, I can't find a clearly preferred way to deal with preloaded data in such a scenario. It mostly depends on the input you get from your stakeholders and end users.

However, I would suggest that orders be automatically displayed if the user is still viewing the customer for which the download of orders has just completed.

The $.ajax method works asynchronously and is attached to its own success callback. The callback receives orders downloaded for a given customer, but at the time the callback runs, the displayed customer may be different. The policy I used ensures that orders are displayed directly if they refer to the current customer. Otherwise, orders are cached and made available for when the user comes back and clicks the "View orders" button.

Let's have a second look at the success callback for the fetch procedure:

```
  function(response)
{
  // Store orders to the cache
  $('#viewOfCustomers').data(id, response.text);

  // If the current customer is the customer for which orders
  // have been fetched, update the user interface
  if (id == currentCustomer)
     $("#listOfOrders0").html(response.text);
}
```

The id variable is local to the $.ajax method and is set with the ID of the customer for which orders are being fetched. The currentCustomer variable, though, is a global variable that is set any time the fetch procedure is executed (see **Figure 3**). The trick is that a global variable may be updated from multiple points, so the check at the end of the download callback makes sense.

What's the role of the "View orders" button that you see in **Figure 1** and **Figure 4**? The button is there for users wanting to see orders for a given customer. By design, in this example displaying orders is an option. Hence, a button that triggers the view is a reasonable element to have in the user interface.

When the user clicks to view orders, order information may, or may not, be available at that time. If orders are not available, it means that—by design—the download is pending, or it has failed for some reason. The user is therefore presented with the user interface shown in **Figure 5**.

If the user remains on the same page, orders display automatically as the download completes successfully, as in **Figure 4**.

The DataView is a formidable instrument for data binding in the context of Web client applications.

The First Displayed Customer

One thing remains to finish the demo of this master-detail scenario enriched with predictive fetch capabilities. The DataView component allows the specification of a particular data item to be rendered in selected mode on display. You control the item to be initially selected via the initialselectedindex attribute of the DataView component. This is shown below:

```
<ul class="sys-template" sys:attach="dataview" id="masterView"
  dataview:dataprovider="/aspnetajax4/mydataservice.asmx"
  dataview:fetchoperation="LookupCustomers"
  dataview:selecteditemclass="selecteditem"
  dataview:initialselectedindex="0">
```

In this case, the first customer retrieved for the selected initial is automatically displayed. Because the user doesn't need to click, no automatic fetch of orders occurs. You can still access the orders of the first customer by clicking on it again. In this way, the first customer will be processed as any other displayed customer. Is there any way to avoid such behavior?

For the user, clicking to view orders wouldn't be sufficient for the first customer. In fact, the button handler limits the information that can be displayed to what's in the cache. This is done to avoid duplicated behavior in code and to try to do everything once and only once. This is shown here:

```
function display()
{
    // Attempt to retrieve orders from cache
    var cachedInfo = $('#viewOfCustomers').data(currentCustomer);
    if (typeof (cachedInfo) !== 'undefined')
        data = cachedInfo;
    else
        data = "No orders found yet. Please wait ...";

    // Display any data that has been retrieved
    $("#listOfOrders0").html(data);
}
```

The preceding display function has to be slightly improved to trigger order fetches in the case of no current customer selection. This is another reason for having a global currentCustomer variable. Here's the edited code of the display function:

```
function display()
{
    if (currentCustomer == "")
    {
        // Get the ID of the first item rendered by the DataView
        currentCustomer = $("#itemCustomer0").attr("commandargument");

        // The fetchOrders method requires a DOM element.
        // Extract the DOM element from the jQuery result.
        fetchOrders($("#itemCustomer0")[0]);
    }

    // Attempt to retrieve orders from cache
    ...

    // Display any data that has been retrieved
    ...
}
```

If no customer has been manually selected, the sys:commandargument of the first rendered item is read. The quickest way of doing that is leveraging the naming convention for the ID of items rendered through the DataView. The original ID is appended with a progressive number. If the original ID is itemCustomer, then the ID of the first element will be itemCustomer0. (This is an aspect of DataView that may change in the final release version of the ASP.NET Ajax Library.) Note also that fetchOrders requires you to pass in a DOM element. A jQuery query returns a collection of DOM elements. That's why in the code above you need to add an item selector.

Finally, note that another solution is also possible, if it's acceptable to you that no customer is initially displayed after data binding. If you set the initialselectedindex attribute of the DataView to -1, no customer would be initially selected. As a result, to see order details, you need to click on any customer, which would trigger the fetch of associated orders.

Wrapping Up

The DataView is a formidable instrument for data binding in the context of Web client applications. It's specifically designed for common scenarios such as master-detail views. It doesn't support every possible scenario, however. In this article, I showed some code that extends a DataView solution through the implementation of the "predictive fetch" pattern.

[*The ASP.NET Ajax Library beta is available for download at* ajax.codeplex.com. *It is expected to be released at the same time as Visual Studio 2010.—Ed.*] ■

DINO ESPOSITO *is the author of the upcoming "Programming ASP.NET MVC" from Microsoft Press and co-authored "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:
Stephen Walther

XCEED

DataGrid

for WPF

The most advanced WPF data presentation control

Microsoft®
Visual Studio® Team System 2010

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

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

IBM®
U2 SystemBuilder™

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

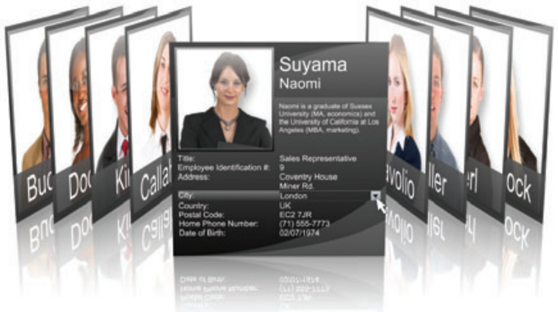
Vincent Smith
U2 Tools Product Manager at IBM

Microsoft® Visual Studio® PARTNER

XCEED
MULTI-TALENTED COMPONENTS

Datagrids, transformed

- Display & edit data in *stunning 2D* or *3D*
- *Highest-performance* WPF datagrid
- Most *adopted*, most *mature* WPF control
- *150* features, *10* major releases in *3* years



Suyama
Naomi

Naomi is a graduate of Sussex University (MA, economics) and the University of California at Los Angeles (MBA, marketing).

Title:	Sales Representative
Employee Identification #:	9
Address:	Coventry House Miner Rd.
City:	London
Country:	UK
Postal Code:	EC2 7JR
Home Phone Number:	(71) 555-7773
Date of Birth:	02/07/1974

The Coverflow™ 3D view provides a true 3D experience to add to your application. Also offered is a classic 2D card view.

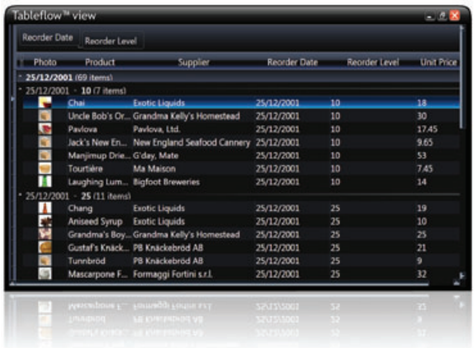


Photo	Product	Supplier	Reorder Date	Reorder Level	Unit Price
25/12/2001 (69 items)					
25/12/2001 - **10** (7 items)					
	Chai	Exotic Liquids	25/12/2001	10	18
	Uncle Bob's Or...	Grandma Kelly's Homestead	25/12/2001	10	30
	Pavlova	Pavlova, Ltd.	25/12/2001	10	17.45
	Jack's New En...	New England Seafood Cannery	25/12/2001	10	9.65
	Manjimup Drie...	G'day, Mate	25/12/2001	10	53
	Tourtière	Ma Maison	25/12/2001	10	7.45
	Laughing Lum...	Bigfoot Breweries	25/12/2001	10	14
25/12/2001 - **25** (11 items)					
	Chang	Exotic Liquids	25/12/2001	25	19
	Aniseed Syrup	Exotic Liquids	25/12/2001	25	10
	Grandma's Boy...	Grandma Kelly's Homestead	25/12/2001	25	25
	Gustaf's Knäck...	PB Knäckebröd AB	25/12/2001	25	21
	Tunnbröd	PB Knäckebröd AB	25/12/2001	25	9
	Mascarpone F...	Formaggi Fortini s.r.l.	25/12/2001	25	32

The new smooth-scrolling Tableflow™ view provides an amazingly rich, fluid, and high-performance user experience.

Try it live on xceed.com

XCEED
MULTI-TALENTED COMPONENTS

Surprise!

Gesture Magic

Touchable screens have been synonymous with Windows Mobile since the first devices appeared back in 2002; however, Windows Mobile 6.5 is the first version to claim any form of gesture support that is exposed to developers. So what is a gesture and why all the fuss?

The traditional touch screens found on Windows Mobile Professional devices provide a mouse simulation surface producing mouse-left-button and mouse-move messages through the screen driver interface. These messages are processed and delivered as if the screen and stylus were a physical mouse, and there are definite similarities: a mouse produces a stream of location coordinates in a linear fashion and can be used as a very precise pointing device, just like a stylus on a screen.

There are differences, as well. For example, a mouse sends position information independent of button information, but the touch screen always simulates the left button being pressed and sends position information only when there is contact with the screen. This paradigm can continue as long as the similarities remain strong. However, with the ever-increasing screen sizes on modern phones, the most natural and intuitive stylus rapidly becomes the user's index finger. For consumer markets, reliance on a fiddly and easily lost stylus is fast going out of fashion, replaced with the demand for a bold and interactive interface that shouts "touch me!" to encourage an emotional connection with a user.

Fingers and thumbs present a different profile, in total contrast to the precision of a stylus tip, so we see the similarities to mouse input break down. The input data is no longer pinpoint accurate, and often the linear input is more akin to a lunar orbit than a straight line. And it's not just the data that's different; input is expected to result in a smooth, animated response, proportional to the input sequence. At this point, it's clear the mouse paradigm no longer fits and we need something new and different to help describe this type of input and understand how to respond. Enter gestures.

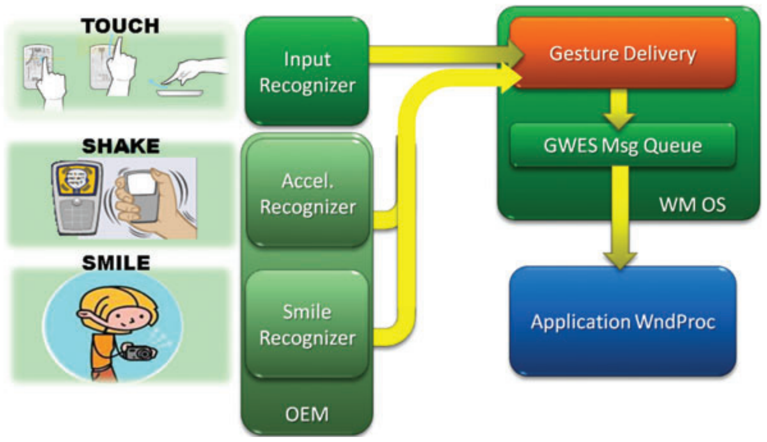


Figure 1 **General Gesture Architecture**

Although we may have only touch gesture recognition today, new gestures can be delivered through the system once the sensor and recognition components are present.

It's Not Just About Touch Gestures

Before we get to the details of touch gestures, let's take a moment to step back and think on a broader level about gestures in general. A gesture can mean lots of different things. It might be a finger movement on a computer screen, but shaking your head is also a gesture, as is waving your arm or shaking hands with someone. My point is that it would be shortsighted to just consider the input on a screen as the only source of gestures. Many devices today have multiple sensors, including touch screens, accelerometers, compasses, GPS instruments and cameras. Shaking a device, turning it over, turning it around in a circle or even just smiling at the camera could all be interpreted as gestures to which the software needs to respond, and that's just with the sensors we know about today.

With this in mind, the architecture in Windows Mobile 6.5 was designed to separate a gesture's source and recognition process

Send your questions and comments for Marcus to goplaces@microsoft.com.

Code download available at code.msdn.microsoft.com/mag201002Gesture.

from the routing, delivery and response to that gesture. Although we may have only touch gesture recognition today, new gestures can be delivered through the system once the sensor and recognition components are present. New sensors and recognition software can be added by hardware manufacturers and integrated into the existing gesture delivery architecture—see **Figure 1**. I'll come back later to look more closely at gesture targeting and delivery.

Touch Recognizer

Figure 2 shows the new touch gesture components in Windows Mobile 6.5: Gesture Recognizer, Gesture Delivery, Physics Engine, and Window Auto Gesture (WAG). We will take a look at each, starting with the Gesture Recognizer.

The Gesture Recognizer component connects directly to the input from the existing touch driver. The input information provided by the driver remains unchanged in Windows Mobile 6.5 in order to keep OEM development costs low and to encourage adoption.

There are five recognized touch gestures in Windows Mobile 6.5 (see **Figure 3**):

- *Select:* touch location stays within a movement threshold (that is, the maximum allowed movement of the finger), and touch duration is less than the threshold time.
- *Hold:* touch location stays within a more lenient threshold and exceeds the select threshold time.
- *Double Select:* two correctly recognized select gestures are recognized within a threshold time, and occur within a distance tolerance.
- *Pan:* touch point movement exceeds the select threshold. This gesture is slightly different and is classed as continuous because it generates more than one gesture event.
- *Scroll:* the most complex gesture to recognize, as it has speed, angle deviation and distance thresholds.

You might wonder why we have some of these gestures, because the mouse behavior appears sufficient to acquire the same information. For example, the Select gesture seems just like clicking on a button, and Pan is just like a mouse move. There are two main reasons why all five of these gestures are important.

Consistency: A mouse click is received as two messages, down and up of the mouse button. The exact behavior for recognizing a click is specific to the control that recognizes it. For example, a button control recognizes the mouse down and mouse up as a click when both locations are within the windows bounds. In contrast, the ListView control recognizes the same event, but for each item in its list. The Select gesture is recognized independently of the control, using consistent parameters. The distance thresholds used for gesture recognition are resolution-aware (or more accurately, dots-per-inch-aware) and are set in order to work with the broadest range of finger profiles (there is a surprising range of finger shapes). So the same physical distances are used on different-sized screens to provide consistency among devices.

Routing: A finger is not an accurate pointing device, especially when the user is moving or walking around, so it's vital that applications maximize the touchable target area. The Gesture Delivery component implements some specific rules to assist with this task and increase the value of these simple gestures.

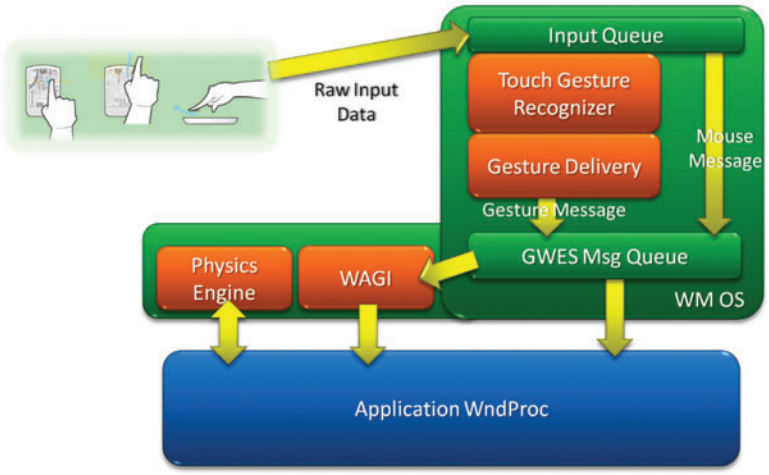


Figure 2 **Touch Gesture Components**

Routing

Gesture information is delivered via the new WM_GESTURE message, and as with all window messages, there are associated parameters—DWORD wParam and LONG lParam—that contain the details of the message. The WM_GESTURE message parameters contain the gesture ID as a wParam to indicate which gesture is being delivered, and a handle to the full gesture information as an lParam. A mouse message is always sent to the topmost window at the location of the mouse coordinates (discounting mouse capture scenarios), but for gestures the rules are different. Gesture messages are different and are always sent to the topmost window

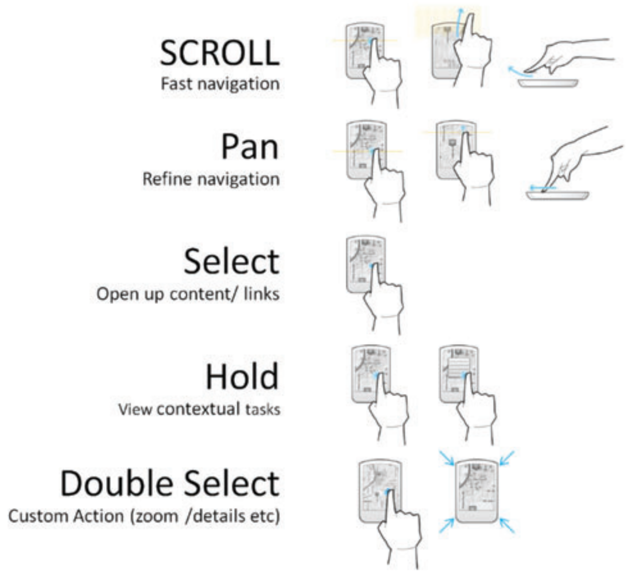


Figure 3 **Five Core Gestures**

under the very first touch point of the sequence that makes up the full gesture sequence. This subtlety doesn't make much of an impact for Select, Hold and Double Select gestures, which have only small screen movement tolerances. However, the Pan gesture is quite different. When you start panning, all Pan messages are sent to the window in which the panning starts, even if the panning movement takes the touch point outside of that original window.

In the same way, the Scroll gesture is recognized many pixels from its original touch-point location. But it makes sense that the Scroll should be routed to the same window as the preceding Pan messages, as the user started the input sequence in that original control and intended to target it. Considering that the Pan gesture is often associated with direct manipulation—moving content around the screen as if it were a piece of paper on a desktop—this routing makes a lot of sense, because the control or screen point under the finger on the initial touch should remain under the finger as the content is moved around the screen.

Unhandled Message Routing

Another unusual aspect of gesture message routing is what happens to unhandled gesture messages. Like all unhandled messages, they end up being sent to DefWindowProc for default processing. When DefWindowProc receives a gesture message, it attempts to find the window's parent and send the message on to that window. This is done to maximize the touchable area available to the user.

To help explain, consider a scrollable window with a number of child label controls. The parent window implements Pan and Scroll gesture response logic to move the child label controls up and down on the visible surface. However, the label controls are unmodified and know nothing about gesture support. If the user happens to start a gesture by touching on a label control instead of the parent window, the user's expectation is the same—that the form will move in response to input movement. By forwarding the unhandled gesture messages from the label control to the parent window, the user's expectation is met and the content moves as if the user had touched on the form directly. This behavior is illustrated in **Figure 4**.

There is a small gotcha to call out here: Never send gesture messages from parent to child window or you risk invoking an infinite loop and an inevitable stack overflow crash. There is some basic loop detection implemented in DefWindowProc to try to prevent this situation, but it may not detect all occurrences.

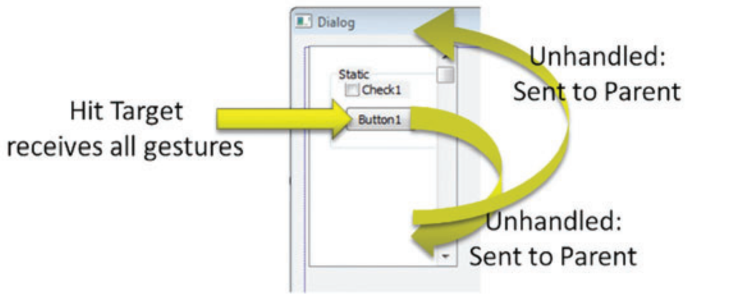

Dialog
Static
Check1
Button1
Hit Target receives all gestures
Unhandled: Sent to Parent
Unhandled: Sent to Parent

Figure 4 **Message Routing**

Gesture Messages

Windows Mobile 6.5 recognizes five gestures, but applications can receive seven gesture types. The extra two gesture types are BEGIN and END, sent at the beginning and end of a gesture sequence (all gesture types are prefixed with GID_ to indicate Gesture IDentifier, so these are GID_BEGIN and GID_END). For example, if a Select gesture is recognized, the application will receive three gesture messages: GID_BEGIN, GID_SELECT and GID_END. For a Pan sequence ending in a Scroll gesture, the application will receive GID_BEGIN, GID_PAN, GID_PAN …, GID_SCROLL and finally GID_END.

GID_BEGIN is useful as it contains the screen coordinates of the original touch point. GID_END is handy as it indicates when the user input has ended and no further gestures will be sent for the current sequence.

Never send gesture messages from parent to child window or you risk invoking an infinite loop and an inevitable stack overflow crash.

To help introduce the basic gesture recognition and delivery system in Windows Mobile 6.5, I've included a Visual Studio project in the attached samples called SimpleGestureCapture. This sample shows a listbox and adds a new line for every gesture message received by the main window, including location information for all gestures and the angle and speed of scroll gestures. You will need Visual Studio 2005 or Visual Studio 2008 plus the Windows Mobile 6 Professional SDK and the Windows Mobile 6.5 Developer Tool Kit installed. From this sample you can see how the gesture message is received and the data extracted.

Physics

The most exciting part of gesture support is the natural response users experience when manipulating screen content. The key part of this response is the consistent, predictable and natural experience across the device. To achieve this consistency, a new component has been added to the OS called the Physics Engine. This module provides a suite of number-crunching algorithms that take input information, such as the angle and speed from a Scroll gesture, and decay the speed over time using a specific deceleration coefficient. Also, the Physics Engine can be used to apply boundary animations when the input speed is sufficient to move the animation point outside a bounding rectangle.

To use the Physics Engine in Windows Mobile 6.5, a new instance of the Physics Engine must first be created and initialized. Then, at regular time intervals, it's polled to retrieve the current animation location and the calling application redraws its client region appropriately. The Physics Engine will continue to decay the speed of the animation until it falls below a minimum threshold value, at which point it's marked as complete and can be released.

As part of the initialization data, the application must specify the bounding rectangle of the data space as well as the view rectangle for the display space (see **Figure 5**). If the view rectangle moves outside the bounding rectangle, the Physics Engine will use the selected boundary animation (again, part of the initialization data) to bring the view rectangle back inside. The Physics Engine initialization is flexible enough to allow animation in just one axis or to have different boundary animation for each axis if required.

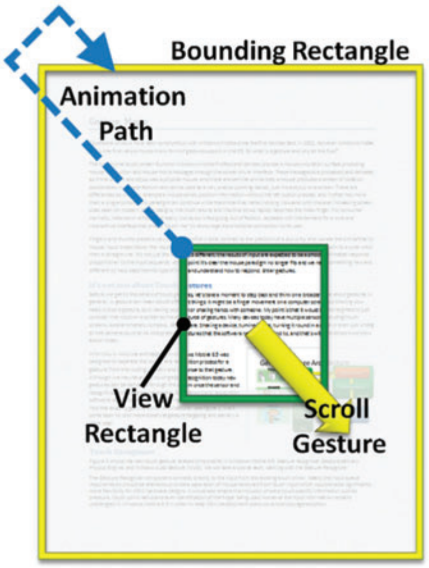


Figure 5 **How the Physics Engine Handles Bounding and Display Rectangles**

By default the Physics Engine decays the speed based on a time delta taken from the point of initialization to the time of each location retrieval call. The calling app can override this by specifying a "user time" value and have the Physics Engine calculate the location at that time. This can be useful for finding the screen position where an animation will complete.

Another interesting Physics Engine configuration is that of item size. This information is used to impose a grid of valid stopping positions over the data space, forcing the Physics Engine to allow the view location final position to end only at one of these grid coordinates. This behavior is helpful when an application is displaying a list of items on the screen and doesn't want a partial item to be displayed at the top of the screen. The behavior works in either or both axes and will adjust the animation decay and stop algorithms to extend or contract the duration of the animation so it hits the required stopping points.

Putting It Together

For an application to fully support touch gestures, it needs to be enhanced to recognize the appropriate gesture messages and respond appropriately. Where necessary, it needs to create and query a Physics Engine instance to drive the screen redraw. Moreover, the application needs to consider what should happen if an animation or gesture sequence is interrupted by further user input or other events, and ensure that it's handled in an efficient way. Although all of this is relatively straightforward to achieve, it does require a reasonable amount of boilerplate code that must be created for each window that responds to gestures. So in Windows Mobile 6.5, a number of steps have been taken to simplify this task.

First, a number of the inbuilt controls have already been updated to support gestures, including the LISTVIEW, LISTBOX, TREEVIEW and WEBVIEW controls (some modes don't support gestures). If you are already using any of these controls, your app is already gesture-enabled.

For applications that don't make use of the inbuilt controls, there is a new API that significantly simplifies the work required to enable gesture support in the most common scenarios, called Window Auto Gesture (WAG).

Window Auto Gesture

The WAG logic is tightly bound to the DefWindowProc() processing to provide a default gesture response available for any window. When enabled, WAG will automatically respond to GID_PAN and GID_SCROLL gestures, create a Physics Engine instance and send the relevant positioning data back to the application through notification messages. WAG also implements gesture interruption by monitoring the input queue when a pan or scroll gesture is in progress, providing appropriate transitions to and from an animation state.

The default configuration for WAG is to ignore gesture messages, so any window that wants to use the WAG behavior must enable it first. To turn gesture support on, the application must call TKSetWindowAutoGesture for each window that requires support and pass the configuration settings required. As I said earlier, WAG is intended to simplify the most common scenarios for gesture support, and in order for WAG to drive your window, it must have been created with the WS_VSCROLL and/or VS_HSCROLL style

The most exciting part of gesture support is the natural response users experience when manipulating screen content.

set in the axes that can be manipulated by touch gestures. Additionally, the application is required to correctly manage the scroll bar, maintaining the range, min/max and page size as appropriate. This is required so that WAG can calculate the data area size that your window is displaying.

WAG has a number of options worth calling out:

- WAG will handle both GID_PAN and GID_SCROLL gestures, but either can be disabled if required.
- Like the Physics Engine, WAG also supports setting item width and height. This information is used not only to set the snapping points, but also to expand the scroll range values from an item count to a pixel count. For example, if the scroll bar range is 0 to 9 for a list of 10 items, and each item requires 20 pixels vertically to display its content, then the item height should be set to 20. WAG will multiply the scroll range (10) by the pixel

height (20) to identify the full pixel range of the data (200 pixels).

- WAG supports a special mode that will drive the window movement by generating WM_xSCROLL messages to the application instead of the more common owner animation messages. This is useful if you have a legacy application and want touch gesture support with the absolute minimum changes to its code. This mode is enabled by setting the nOwnerAnimateMessage value that is part of the TKSetWindowAutoGesture() initialization data to 0 instead of the normal WM_USER + x value. Some functionality is limited in this mode, such as no support for pixel-by-pixel manipulation—the control can only be manipulated item by item. Also, there is no way to go outside the scroll range in this mode, so the extent values are ignored. This option doesn't work well for scrolling in both axes at the same time because each axis must be moved independently.
- Extents describe the distance the display area can be dragged beyond the data range and is expressed as a percentage of the display size. Take care when enabling extents, because this allows the user to drag the display beyond the scroll limits and expose a screen area that many existing applications aren't capable of handling correctly. Ensure the application is correctly clearing the screen when space appears beyond the top or to the left of the data range.

Typically an application will configure WAG with nOwnerAnimateMessage as a value in the range WM_USER to WM_APP. WAG will use this value in the message sent back to the application each time the application needs to redraw its display area. The first animation message in a sequence will be preceded by a status message indicating that the control is now responding to gesture input. WAG automatically aggregates GID_PAN gesture messages and only sends an animation message to the application at a maximum frequency of 24 times per second (regulated using the GESTURE_ANIMATION_FRAME_DELAY_MS timer duration found in gesturephysics.h from the Windows Mobile 6.5 Developer Tool Kit). The same applies for scroll animations, where WAG uses the same timer to query its Physics Engine a maximum of 24 times per second.

The status message option for WAG is especially useful if your control supports focus or changes visually without user interaction, for example via asynchronous updates. Status messages tell the control when the user is interacting through the touch interface. They should be used as a trigger to halt any updates that might change the visual aspects of the control or its content, or unnecessarily take resources from the screen animation. Producing a full-screen animated effect can be resource-intensive, so it's important to halt any unnecessary background processing and concentrate the resources to provide smooth and timely response to the user. Once the touch interaction is done, use the status message to trigger a data refresh and update, if required.

For more information on the WAG API, see the MSDN documentation for Windows Mobile 6.5 (msdn.microsoft.com/library/ee220917).

Tips and Tricks

Using the gesture API to accept and process gesture information is straightforward. However, it can be a little trickier to produce smooth animation in response to the gestures. Here are some tips that may help.

First frame time is vital. It's surprising how sensitive the human eye can be to user interface latency. For example, a delay of more than 100ms between a screen touch and a graphical response can result in a feeling of sluggishness, even if the application then maintains a steady 24 frames per second (fps). Work to ensure the first frame response is fast, ideally below 50ms. It's worth noting that the overhead of the Gesture Recognizer and Gesture Delivery have been carefully optimized, resulting in only 1ms or 2ms from touch to application.

Prefer a consistent frame rate. In our testing, users preferred a slightly slower but more consistent frame rate over a faster but more variable rate. We applied this information by making a timer to regulate the frame update frequency, and tuning the timer to ensure some free CPU time in each frame to handle other tasks.

Remove unnecessary overhead during animation. It's obvious that the less work there is per frame, the more frames can be drawn per second. However, it's sometimes harder to identify exactly what work can be left out. During touch manipulation, and especially during scroll animations, the user is less interested in detail and more interested in broad indicators. For example, while scrolling a list of e-mail messages, the user might be less interested in a preview of each message but more interested in its location in the list and its title. So it may be okay to stop updating or retrieving the preview text in order to allow extra time for smooth animation.

In our testing, users preferred a slightly slower but more consistent frame rate over a faster but more variable rate.

Judicious use of off-screen buffers. Double-buffering can be an excellent way of improving drawing performance and reducing fragmented drawing of the screen. However, it must be applied carefully, as an off-screen buffer is costly in resources. Ensure the buffer is held for the shortest possible time and is kept to a minimum size. Using the ScrollWindowEx API can often achieve similar results without the memory overhead of an off-screen buffer.

Measure first and then apply appropriate improvements. It's standard performance-analysis practice to ensure you're fixing something that is actually broken. So before changing any code, make sure you understand where the costs are in your animation loop by measuring them first, and then apply your effort to the areas that will yield the most significant benefits to your application.

Doing It Managed

Managed code applications that use common controls (such as LISTBOX, LISTVIEW, WEBVIEW and TREEVIEW) will automatically benefit from the touch gesture support added to these controls without any code changes. For applications that have custom controls, the control code will need to be modified to make use of gestures through the API exposed in the Windows Mobile 6.5 Developer Tool Kit. The tool kit contains C++ headers and samples and is aimed at native code developers. However, the APIs are designed to be easy to use through a simple interop from managed code.

It's surprising how sensitive the human eye can be to user interface latency.

The trickiest part of implementing gesture support is being able to receive the new WM_GESTURE message and the WAG animation messages, because unlike the desktop, the compact framework doesn't expose the WndProc handler. To get at these messages requires the common technique of sub-classing the window to get a first look at all messages sent to it and filter out the ones you need. This can be done by using a native helper DLL or by simply calling directly to the native APIs. In the sample code available with this article on the MSDN online site, I've included some examples that show how this might be achieved, along with three projects showing touch gestures, the Physics Engine and WAG all in use with managed code. You'll also find several solutions available in the community.

Next Steps

To get started with gestures on Windows Mobile 6.5, be sure to download the Developer Tool Kit from microsoft.com/downloads/details.aspx?FamilyID=20686a1d-97a8-4f80-bc6a-ae010e085a6e. It includes emulators and samples to explore many of the possibilities. Also, the MSDN documentation for this native API is available at msdn.microsoft.com/library/ee220920. If you're looking for managed code solutions, take a look at the sample code attached to this article on the MSDN page or at Maarten Struys' blog (dotnetfordevices.com/forum.html?monthidx=8&yearidx=2009) or Alex Yakhnin's blog (blogs.msdn.com/priozersk/archive/2009/08/28/managed-wrapper-of-the-gesture-apis.aspx).

There are also more of my ramblings about touch gestures on my blog:

- Let's Talk About Touch (Part 1): blogs.msdn.com/marcpe/archive/2009/06/29/let-s-talk-about-touch-part1.aspx
- Let's Talk About Touch (Part 2): blogs.msdn.com/marcpe/archive/2009/06/29/let-s-talk-about-touch-part2.aspx ■

MARCUS PERRYMAN *has worked at Microsoft for more than 10 years in various technical roles, including developer evangelist and developer consultant. At present, Perryman works as a software design engineer in the Windows Mobile product group designing and developing the next generation of mobile operating systems.*

THANKS to the following technical experts for reviewing this article: *Tim Benton, David Goon, John Lawrence, Stewart Tootill and Marcin Stankiewicz*

InishTech Software Licensing and Protection Services

LICENSE MANAGE PROTECT

Developed the Best Product? That's only Half the Battle!

Exploit your product's sales potential to the max and protect your core IP with the industry's strongest code and license protection. Make it easy for customers to buy with configurable features and flexible business terms including subscription and pay-as-you-go with InishTech's Software Licensing and Protection Online Service (SLPS).

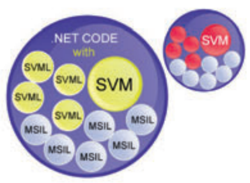
License your Products

Configure products, features and business models independent of code.

Manage your Customers

Monitor usage and manage upgrade cycles to reduce support costs and increase sales.

Protect your Hard Work



Patented Secure Virtual Machine technology protects licenses and IP with no impact on code.

SLPS is powerful, yet easy to setup and manage. Subscriptions start at just $45 per month and most customers see a return on their investment within the first six months. Contact us at sales@inishtech.com or check out our website on **www.inishtech.com/redmond** You'll be glad you did!

RON PETRUSHA

Formatting and Parsing Time Intervals in the .NET Framework 4

In the Microsoft .NET Framework 4, the TimeSpan structure has been enhanced by adding support for both formatting and parsing that is comparable to the formatting and parsing support for DateTime values. In this article, I'll survey the new formatting and parsing features, as well as provide some helpful tips for working with TimeSpan values.

Figure 1 Time Interval Formatted Using "G" Format String

Visual Basic

```
Imports System.Globalization

Module Example
   Public Sub Main()
      Dim interval As New TimeSpan(1, 12, 42, 30, 566)
      Dim cultures() As CultureInfo = { New CultureInfo("en-US"),
                                        New CultureInfo("fr-FR") }
      For Each culture As CultureInfo In cultures
         Console.WriteLine("{0}: {1}", culture, interval.ToString(
            "G", culture))
      Next
   End Sub
End Module
```

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      TimeSpan interval = new TimeSpan(1, 12, 42, 30, 566);
      CultureInfo[] cultures = { new CultureInfo("en-US"),
                                 new CultureInfo("fr-FR") };
      foreach (CultureInfo culture in cultures)
         Console.WriteLine("{0}: {1}", culture, interval.ToString( _
            "G", culture));
   }
}
```

Formatting in the .NET Framework 3.5 and Earlier Versions

In the Microsoft .NET Framework 3.5 and earlier versions, the single formatting method for time intervals is the parameterless TimeSpan.ToString method. The exact format of the returned string depends on the TimeSpan value. At a minimum, it includes the hours, minutes and seconds components of a TimeSpan value. If it is non-zero, the day component is included as well. And if there is a fractional seconds component, all seven digits of the ticks component are included. The period (".") is used as the separator between days and hours and between seconds and fractional seconds.

Expanded Support for Formatting in the .NET Framework 4

While the default TimeSpan.ToString method behaves identically in the .NET Framework 4, there are now two additional overloads. The first has a single parameter, which can be either a standard or custom format string that defines the format of the result string. The second has two parameters: a standard or custom format string, and an IFormatProvider implementation representing the culture that supplies formatting information. This method, incidentally, provides the IFormattable implementation for the TimeSpan structure; it allows TimeSpan values to be used with methods, such as String.Format, that support composite formatting.

In addition to including standard and custom format strings and providing an IFormattable implementation, formatted strings can now be culture-sensitive. Two standard format strings, "g" (the general short format specifier) and "G" (the general long format specifier) use the formatting conventions of either the current culture or a specific culture in the result string. The example formats in **Figure 1** provide an illustration by displaying the result string for a time interval formatted using the "G" format string and the en-US and fr-FR cultures. The example in **Figure 1** displays the following output:

```
en-US: 1:12:42:30.5660000
fr-FR: 1:12:42:30,5660000
```

Parsing in the .NET Framework 3.5 and Earlier Versions

In the .NET Framework 3.5 and earlier versions, support for parsing time intervals is handled by the static System.TimeSpan.Parse and System.TimeSpan.TryParse methods, which support a limited number of invariant formats. The example in **Figure 2** parses the string representation of a time interval in each format recognized by the method. The example in **Figure 2** displays the following output:

```
Converted 12 to 12.00:00:00
Converted 12.16:07 to 12.16:07:00
Converted 12.16:07:32 to 12.16:07:32
Converted 12.16:07:32.449 to 12.16:07:32.4490000
Converted 12.16:07:32.4491522 to 12.16:07:32.4491522
Converted 16:07 to 16:07:00
Converted 16:07:32 to 16:07:32
Converted 16:07:32.449 to 16:07:32.4490000
```

This column is based on a pre-release version of the Microsoft .NET Framework 4. All information is subject to change.

Post your questions on the CLR Team blog at blogs.msdn.com/clrteam.

Now v4.0!

Why is Amyuni PDF so interesting?

Proven

Choose a PDF technology that is integrated into thousands of applications behind millions of desktops worldwide.

High-Performance

Develop with the fastest PDF conversion on the market, designed to perform in multithreaded and 64-bit Windows environments.

Rapid Integration

Integrate PDF conversion, creation and editing into your .NET and ActiveX applications with just a few lines of code.

Expertise

Produce accurate and stable PDF documents using reliable tools built by experts with over ten years of experience.

OEM Licenses

License and distribute products quickly and easily with a PDF technology that does not rely on external open-source libraries.

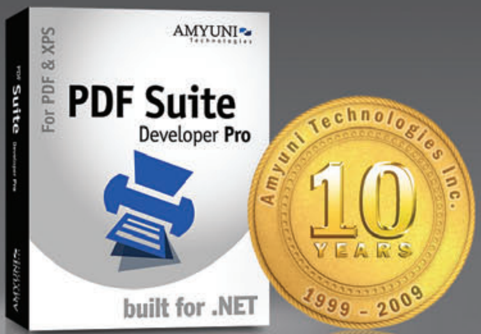
Customization

Let our experienced consultants help you turn your software requirements into customized PDF solutions.

SD Times 2009 100

Windows 7

Certified for Windows Vista



We understand the challenges that come with PDF integration. From research and development, through design and implementation, we work with you every step of the way.

Get 30 days of FREE technical support with your trial download!

www.amyuni.com

USA and Canada
Toll Free: 1 866 926 9864
Support: (514) 868 9227
Info: sales@amyuni.com

Europe
Sales: (+33) 1 30 61 07 97
Support: (+33) 1 30 61 07 98
Customizations: management@amyuni.com

AMYUNI Technologies

As the output shows, the method can parse a single integer, which it interprets as the number of days in a time interval (more about this later). Otherwise, it requires that the string to be parsed includes at least an hour and a minute value.

Expanded Support for Parsing in the .NET Framework 4

In the .NET Framework 4 and Silverlight 4, support for parsing the string representations of time intervals has been enhanced and is now comparable to support for parsing date and time strings. The TimeSpan structure now includes a new overload for the Parse and TryParse methods, as well as completely new ParseExact and TryParseExact methods, each of which has four overloads. These parsing methods support standard and custom format strings, and offer some support for culture-sensitive formatting. Two standard format strings ("g" and "G") are culture-sensitive, while the remaining standard format strings ("c", "t" and "T") as well as all custom format strings are invariant. Support for parsing and formatting time intervals will be further enhanced in future releases of the .NET Framework.

The example in **Figure 3** illustrates how you can use the ParseExact method to parse time interval data in the .NET Framework 4. It defines an array of seven custom format strings; if the string representation of the time interval to be parsed does not conform to one of these formats, the method fails and throws an exception.

The example in **Figure 3** displays the following output:

```
Converted '16' to 16:00:00
Converted '1' to 01:00:00
Converted '16:03' to 16:03:00
Converted '1:12' to 01:12:00
Converted '1.13:34:15' to 1.13:34:15
Converted '41237' to 00:00:00.4123700
Converted '0609' to 06:09:00
```

Instantiating a TimeSpan with a Single Numeric Value

Interestingly, if these same seven time interval strings were passed to the TimeSpan.Parse(String) method in any version of the .NET Framework, they would all parse successfully, but in four cases, they

Figure 2 **Parsing Time Interval String in Multiple Formats**

Visual Basic

```
Module Example
   Public Sub Main()
      Dim values() As String = {"12", "12.16:07", "12.16:07:32", _
                                "12.16:07:32.449", "12.16:07:32.4491522",
_
                                "16:07", "16:07:32", "16:07:32.449" }

      For Each value In values
         Try
            Console.WriteLine("Converted {0} to {1}", _
                              value, TimeSpan.Parse(value))
         Catch e As OverflowException
            Console.WriteLine("Overflow: {0}", value)
         Catch e As FormatException
            Console.WriteLine("Bad Format: {0}", value)
         End Try
      Next
   End Sub
```

C#

```
using System;

public class Example
{
   public static void Main()
   {
      string[] values = { "12", "12.16:07", "12.16:07:32",
                          "12.16:07:32.449", "12.16:07:32.4491522",
                          "16:07", "16:07:32", "16:07:32.449" };

      foreach (var value in values)
         try {
            Console.WriteLine("Converted {0} to {1}",
                              value, TimeSpan.Parse(value));}
         catch (OverflowException) {
            Console.WriteLine("Overflow: {0}", value); }
         catch (FormatException) {
            Console.WriteLine("Bad Format: {0}", value);
         }
   }
}
```

Figure 3 **Parsing Time Interval Data with ParseExact Method**

Visual Basic

```
Module modMain
   Public Sub Main()
      Dim formats() As String = { "hh", "%h", "h\:mm", "hh\:mm",
                                  "d\.hh\:mm\:ss", "fffff", "hhmm" }
      Dim values() As String = { "16", "1", "16:03", "1:12",
                                 "1.13:34:15", "41237", "0609" }
      Dim interval As TimeSpan

      For Each value In values
         Try
            interval = TimeSpan.ParseExact(value, formats, Nothing)
            Console.WriteLine("Converted '{0}' to {1}",
                              value, interval)
         Catch e As FormatException
            Console.WriteLine("Invalid format: {0}", value)
         Catch e As OverflowException
            Console.WriteLine("Overflow: {0}", value)
         Catch e As ArgumentNullException
            Console.WriteLine("No string to parse")
         End Try
      Next
   End Sub
End Module
```

C#

```
using System;

public class Example
{
   public static void Main()
   {
      string[] formats = { "hh", "%h", @"h\:mm", @"hh\:mm",
                           @"d\.hh\:mm\:ss", "fffff", "hhmm" };
      string[] values = { "16", "1", "16:03", "1:12",
                          "1.13:34:15", "41237", "0609" };
      TimeSpan interval;

      foreach (var value in values)
      {
         try {
            interval = TimeSpan.ParseExact(value, formats, null);
            Console.WriteLine("Converted '{0}' to {1}", value,
                              interval); }
         catch (FormatException) {
            Console.WriteLine("Invalid format: {0}", value); }
         catch (OverflowException) {
            Console.WriteLine("Overflow: {0}", value); }
         catch (ArgumentNullException) {
            Console.WriteLine("No string to parse");
         }
      }
   }
}
```

would return a different result. Calling TimeSpan.Parse(String) with these strings produces the following output:

```
Converted '16' to 16.00:00:00
Converted '1' to 1.00:00:00
Converted '16:03' to 16:03:00
Converted '1:12' to 01:12:00
Converted '1.13:34:15' to 1.13:34:15
Converted '41237' to 41237.00:00:00
Converted '0609' to 609.00:00:00
```

Support for parsing the string representations of time intervals has been enhanced.

The major difference in the TimeSpan.Parse(String) and TimeSpan.ParseExact(String, String[], IFormatProvider) method calls lies in the handling of strings that represent integer values. The TimeSpan.Parse(String) method interprets them as days. The interpretation of integers by the TimeSpan.ParseExact(String, String[], IFormatProvider) method depends on the custom format strings supplied in the string array parameter. In this example, strings that have only one or two integer digits are interpreted as the number of hours, strings with four digits are interpreted as the number of

Figure 4 **Representations of Integers with 1 to 5 Digits**

Visual Basic

```vb
Module Example
   Public Sub Main()
      Dim formats() As String = { "%h", "hh", "fff", "ffff", "fffff" }
      Dim values() As String = { "3", "17", "192", "3451",
                                 "79123", "01233" }

      For Each value In values
         Dim interval As TimeSpan
         If TimeSpan.TryParseExact(value, formats, Nothing, interval)
Then
            Console.WriteLine("Converted '{0}' to {1}",
                              value, interval.ToString())
         Else
            Console.WriteLine("Unable to parse {0}.", value)
         End If
      Next
   End Sub
End Module
```

C#

```csharp
using System;

public class Example
{
   public static void Main()
   {
      string[] formats = { "%h", "hh", "fff", "ffff", "fffff" };
      string[] values = { "3", "17", "192", "3451", "79123", "01233" };

      foreach (var value in values)
      {
         TimeSpan interval;
         if (TimeSpan.TryParseExact(value, formats, null, out interval))
            Console.WriteLine("Converted '{0}' to {1}",
                              value, interval.ToString());
         else
            Console.WriteLine("Unable to parse {0}.", value);
      }
   }
}
```

What are you looking for in a PDF solution?

Amyuni **PDF Suite 4.0** New Features

- Support for a wider range of PDF files such as PDF 1.7 and the compressed XRef table format.
- The addition and extraction of attachments to and from PDF files.
- Exporting capabilities of PDF files into XAML to view from a Web page with Silverlight controls.
- Updated PDF/A engine ensures Acrobat 9 compatibility.
- Improved file processing speed of PDF files that contain large graphics.
- JPEG2000 compression in addition to decompression.

PDF **Converter**

- Certified for Windows 7 and Windows 2008.
- Generation of layered PDF files for CAD and other applications.
- Improved support for CJK languages and custom CMAPs.
- Generation of industry standard PDF/X-1 and PDF/X-3 PDF files.
- Resaving capabilities of PDF files using PDF 1.4, 1.5, or PDF/A formats.

PDF **Creator**

- Loading, creation, and display of layered PDF files.
- CJK file support, with the ability to specify the location of custom CMAPs.
- Text anti-aliasing for better on-screen legibility.
- Internal PDF file structure access through comprehensive class architecture.
- The ability to print a single page to multiple sheets or multiple pages to a single sheet.

CERTIFIED FOR Windows Vista

Windows 7

SD Times 2009 100

www.amyuni.com

AMYUNI Technologies

hours and minutes, and strings that have five integer digits are interpreted as a fractional number of seconds.

In many cases, .NET Framework applications receive strings containing time interval data in an arbitrary format (such as integers representing a number of milliseconds, or integers representing a number of hours). In previous versions of the .NET Framework, it was necessary to manipulate this data so that it would be in an acceptable format before passing it to the

Figure 5 **Handling Nonstandard Time Interval Strings**

Visual Basic

```
Module Example
   Public Sub Main()
      Dim values() As String = { "37:16:45.33", "0:128:16.324",
                                 "120:08" }
      Dim interval As TimeSpan
      For Each value In values
         Try
            interval = ParseIntervalWithoutOverflow(value)
            Console.WriteLine("'{0}' --> {1}", value, interval)
         Catch e As FormatException
            Console.WriteLine("Unable to parse {0}.", value)
         End Try
      Next
   End Sub

   Private Function ParseIntervalWithoutOverflow(value As String)
                    As TimeSpan
      Dim interval As TimeSpan
      If Not TimeSpan.TryParse(value, interval) Then
         Try
            ' Handle failure by breaking string into components.
            Dim components() As String = value.Split( {"."c, ":"c } )
            Dim offset As Integer = 0
            Dim days, hours, minutes, seconds, milliseconds As Integer
            ' Test whether days are present.
            If value.IndexOf(".") >= 0 AndAlso
                     value.IndexOf(".") < value.IndexOf(":") Then
               offset = 1
               days = Int32.Parse(components(0))
            End If
            ' Call TryParse to parse values so no exceptions result.
            hours = Int32.Parse(components(offset))
            minutes = Int32.Parse(components(offset + 1))
            If components.Length >= offset + 3 Then
               seconds = Int32.Parse(components(offset + 2))
            End If
            If components.Length >= offset + 4 Then
               milliseconds = Int32.Parse(components(offset + 3))
            End If
            ' Call constructor.
            interval = New TimeSpan(days, hours, minutes,
                                    seconds, milliseconds)
         Catch e As FormatException
            Throw New FormatException(
                      String.Format("Unable to parse '{0}'"), e)
         Catch e As ArgumentOutOfRangeException
            Throw New FormatException(
                      String.Format("Unable to parse '{0}'"), e)
         Catch e As OverflowException
            Throw New FormatException(
                      String.Format("Unable to parse '{0}'"), e)
         Catch e As ArgumentNullException
            Throw New ArgumentNullException("value cannot be null.",
                                            e)
         End Try
      End If
      Return interval
   End Function
End Module
```

C#

```
using System;

public class Example
{
   public static void Main()
   {
      string[] values = { "37:16:45.33", "0:128:16.324", "120:08" };
      TimeSpan interval;
      foreach (var value in values)
      {
         try {
            interval = ParseIntervalWithoutOverflow(value);
            Console.WriteLine("'{0}' --> {1}", value, interval);
         }
         catch (FormatException) {
            Console.WriteLine("Unable to parse {0}.", value);
         }
      }
   }

   private static TimeSpan ParseIntervalWithoutOverflow(string value)
   {
      TimeSpan interval;
      if (! TimeSpan.TryParse(value, out interval))
      {
         try {
            // Handle failure by breaking string into components.
            string[] components = value.Split(
                                  new Char[] {'.', ':' } );

            int offset = 0;
            int days = 0;
            int hours = 0;
            int minutes = 0;
            int seconds = 0;
            int milliseconds = 0;
            // Test whether days are present.
            if (value.IndexOf(".") >= 0 &&
                     value.IndexOf(".") < value.IndexOf(":"))
            {
               offset = 1;
               days = Int32.Parse(components[0]);
            }
            // Call TryParse to parse values so no exceptions result.
            hours = Int32.Parse(components[offset]);
            minutes = Int32.Parse(components[offset + 1]);
            if (components.Length >= offset + 3)
               seconds = Int32.Parse(components[offset + 2]);

            if (components.Length >= offset + 4)
               milliseconds = Int32.Parse(components[offset + 3]);

            // Call constructor.
            interval = new TimeSpan(days, hours, minutes,
                                    seconds, milliseconds);
         }
         catch (FormatException e) {
            throw new FormatException(
                      String.Format("Unable to parse '{0}'"), e);
         }
         catch (ArgumentOutOfRangeException e) {
            throw new FormatException(
                      String.Format("Unable to parse '{0}'"), e);
         }
         catch (OverflowException e)
         {
            throw new FormatException(
                      String.Format("Unable to parse '{0}'"), e);
         }
         catch (ArgumentNullException e)
         {
            throw new ArgumentNullException("value cannot be null.",
                                            e);
         }
      }
      return interval;
   }
}
```

Get the most comprehensive file coverage from any of Aspose's 4 Award Winning Suites

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

•DOC	•DOCX	•WordML	•OOXML	•ODF	•RTF	•SpreadsheetML	•XLS	
•XLSX	•CSV	•TAB	•PDF	•FO	•PPT	•POT	•SWF	•& More...

The ever expanding list of features now includes the ability to Read, Write, Convert, Print, View, and Render for a variety of platforms including .NET, Java, SQL Server and JasperReports. Extensitve File Format Processing coupled with unmatched performance, reliability and an award winning customer support makes sure you perform... beyond the rest!

Logon to www.aspose.com and get your free evaluation copy right now!

ASPOSE®
The .NET & Java Component Publisher™

TimeSpan.Parse method. In the .NET Framework 4, you can use custom format strings to define the interpretation of time interval strings that contain only integers, and preliminary manipulation of string data is not necessary. The example in **Figure 4** illustrates this by providing different representations for integers that have from one to five digits.

The example in **Figure 4** displays the following output:

```
Converted '3' to 03:00:00
Converted '17' to 17:00:00
Converted '192' to 00:00:00.1920000
Converted '3451' to 00:00:00.3451000
Converted '79123' to 00:00:00.7912300
Converted '01233' to 00:00:00.0123300
```

Handling OverflowExceptions When Parsing Time Intervals

These new TimeSpan parsing and formatting features introduced in the .NET Framework 4 retain one behavior that some customers have found inconvenient. For backward compatibility, the TimeSpan parsing methods throw an OverflowException under the following conditions:

- If the value of the hours component exceeds 23.
- If the value of the minutes component exceeds 59.
- If the value of the seconds component exceeds 59.

There are a number of ways to handle this. Instead of calling the TimeSpan.Parse method, you could use the Int32.Parse method to convert the individual string components to integer values, which you can then pass to one of the TimeSpan class constructors. Unlike the TimeSpan parsing methods, the TimeSpan constructors do not throw an OverflowException if the hours, minutes or seconds value passed to the constructor is out of range.

This is an acceptable solution, although it does have one limitation: It requires that all strings be parsed and converted to integers before calling the TimeSpan constructor. If most of the data to be parsed does not overflow during the parsing operation, this solution involves unnecessary processing.

Another alternative is to try to parse the data, and then handle the OverflowException that is thrown when individual time interval components are out of range. Again, this is an acceptable solution, although unnecessary exception handling in an application can be expensive.

The best solution is to use the TimeSpan.TryParse method to initially parse the data, and then to manipulate the individual time interval components only if the method returns false. If the parse operation fails, you can use the String.Split method to separate the string representation of the time interval into its individual components, which you can then pass to the TimeSpan(Int32, Int32, Int32, Int32, Int32) constructor. The example in **Figure 5** provides a simple implementation:

As the following output shows, the example in **Figure 5** successfully handles hour values that are greater than 23, as well as minute and second values that are greater than 59:

```
'37:16:45.33' --> 1.13:16:45.0330000
'0:128:16.324' --> 02:08:16.3240000
'120:08' --> 5.00:08:00
```

Application Compatibility

Paradoxically, enhanced formatting support for TimeSpan values in the .NET Framework 4 has broken some applications that formatted TimeSpan values in previous versions of the .NET Framework. The following code, for example, executes normally in the .NET Framework 3.5, but throws a FormatException in the .NET Framework 4:

```
string result = String.Format("{0:r}", new TimeSpan(4, 23, 17));
```

To format each argument in its parameter list, the String.Format method determines whether the object implements IFormattable. If it does, it calls the object's IFormattable.ToString implementation. If it does not, it discards any format string supplied in the index item and calls the object's parameterless ToString method.

Paradoxically, enhanced formatting support for TimeSpan values in the .NET Framework 4 has broken some applications that formatted TimeSpan values in previous versions.

In the .NET Framework 3.5 and earlier versions, TimeSpan does not implement IFormattable, nor does it support format strings. Therefore, the "r" format string is ignored, and the parameterless TimeSpan.ToString method is called. In the .NET Framework 4, on the other hand, TimeSpan.ToString(String, IFormatProvider) is called and passed the unsupported format string, which causes the exception.

If possible, this code should be modified by calling the parameterless TimeSpan.ToString method, or by passing a valid format string to a formatting method. If that is not possible, however, a <TimeSpan_LegacyFormatMode> element can be added to the application's configuration file so that it resembles the following:

```
<?xml version ="1.0"?>
<configuration>
  <runtime>
    <TimeSpan_LegacyFormatMode enabled="true"/>
  </runtime>
</configuration>
```

By setting its enabled attribute to true, you can ensure that TimeSpan uses legacy formatting behavior. ■

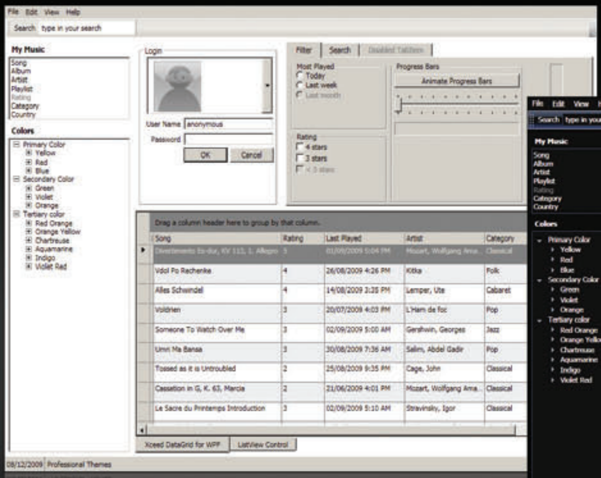
RON PETRUSHA *is a programming writer on the .NET Framework Base Class Library team. He is also the author of numerous programming books and articles on Windows programming, Web programming and programming with VB.NET.*

THANKS to the following technical experts for reviewing this article: *Melitta Andersen and Josh Free*

XCEED
Professional Themes
for WPF

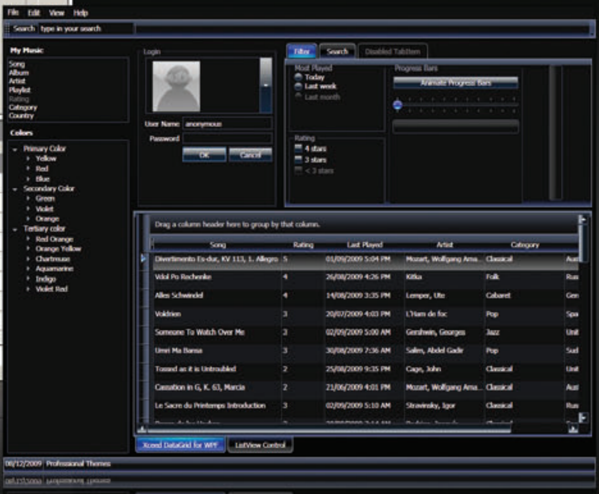
Beautifully themed WPF apps in under 2 minutes.

4:58 PM - Ewww!



Song	Rating	Last Played	Artist	Category
Divertimento Es-dur, KV 113, 1. Allegro	5	01/09/2009 5:04 PM	Mozart, Wolfgang Ama…	Classical
Vdol Po Rechenke	4	26/08/2009 4:26 PM	Kitka	Folk
Alles Schwindel	4	14/08/2009 3:35 PM	Lemper, Ute	Cabaret
Voldrien	3	20/07/2009 4:03 PM	L'Ham de foc	Pop
Someone To Watch Over Me	3	02/09/2009 5:00 AM	Gershwin, Georges	Jazz
Umri Ma Bansa	3	30/08/2009 7:36 AM	Salim, Abdel Gadir	Pop
Tossed as it is Untroubled	2	25/08/2009 9:35 PM	Cage, John	Classical
Cassation in G, K. 63, Marcia	2	21/06/2009 4:01 PM	Mozart, Wolfgang Ama…	Classical
Le Sacre du Printemps Introduction	3	02/09/2009 5:10 AM	Stravinsky, Igor	Classical

5:00 PM - Awesome!



Flawless WPF themes for styling all official Microsoft WPF controls. Requires as few as a couple of lines of XAML code to quickly apply beautiful, uniform styles to your entire WPF application.

Try it live on xceed.com

XCEED
MULTI-TALENTED COMPONENTS

Building Composable Apps in .NET 4 with the Managed Extensibility Framework

Glenn Block

With the upcoming Microsoft .NET Framework 4, you'll find an exciting new technology at your doorstep that will greatly simplify your application development. If you've struggled with how to design applications that are easier to maintain and extend, keep reading.

The Managed Extensibility Framework (MEF) is a new library shipping in the .NET Framework 4 and in Silverlight 4 that simplifies the design of composable systems that can be extended by third parties after they have been deployed. MEF opens up your applications, allowing new functionality to be incrementally introduced by application developers, framework authors and third-party extenders.

This article is based on a pre-release version of the Microsoft .NET Framework 4. All information is subject to change.

This article discusses:

- The Attributed Programming Model
- Decoupling implementation with an interface
- Contract assemblies
- Importing lazy exports and accessing metadata
- Recomposition
- Stable composition, rejection and diagnostics
- MEF in Silverlight 4

Technologies discussed:

Managed Extensibility Framework, .NET Framework 4, Silverlight 4, IronRuby

Why We Built It

Several years ago, within Microsoft, a number of groups were working to find solutions to a problem—how to build applications from reusable components that can be discovered, reused and composed dynamically:

- Visual Studio 2010 was building a new extensible code editor. The editor's core capabilities, as well as third-party capabilities, were all to be deployed as binaries that would be discovered at runtime. One of the core requirements was to support lazily loading extensions in order to improve startup time and memory consumption.
- "Oslo" introduced "Intellipad," a new extensible text editor for working with MEF. In Intellipad, plugins were to be authored in IronPython.
- Acropolis was delivering a framework for building composite applications. The Acropolis runtime discovered application component "parts" at runtime and provided those parts with services in a loosely coupled manner. Acropolis made heavy use of XAML for component authoring.

This problem was not specific to Microsoft. Customers have been implementing their own custom extensibility solutions for ages. Here was clear opportunity for the platform to step in and provide a more general-purpose solution to help both Microsoft and customers.

Did We Need Something New?

MEF is not by any means the first solution in this problem space. There have been many proposed solutions—a long list of ventures that cross platform boundaries and include efforts like EJB, CORBA, Eclipse's OSGI implementation and Spring on the Java side. On Microsoft's platform, there are the Component model and System.Addin within the .NET Framework itself. And there are several open-source solutions, including SharpDevelop's SODA architecture and Inversion of Control containers like Castle Windsor, Structure Map and patterns & practices' Unity.

With all the existing approaches, why come up with something new? We realized none of our current solutions were ideal for general third-party extensibility. They were either too heavyweight for general use or required too much effort on the part of either the host or the extension developer. MEF represents the culmination of learning from each of these solutions, and an attempt to address the pain points just mentioned.

Let's take a look at MEF's core concepts, illustrated in **Figure 1**.

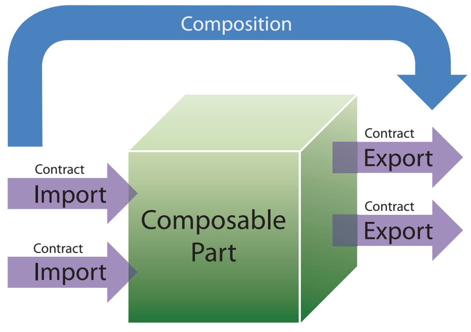


Figure 1 **Core Concepts in the Managed Extensibility Framework**

Concepts

At the heart of MEF are a few essential concepts:

Composable part (or, simply, part)—A part provides services to other parts and consumes services provided by other parts. Parts in MEF can come from anywhere, from within the application or externally; from an MEF perspective, it makes no difference.

Export—An export is a service that a part provides. When a part provides an export, it is said that the part *exports* it. For example, a part may export a logger, or in the case of Visual Studio, an editor extension. Parts can provide multiple exports, though most parts provide a single export.

Import—An import is a service that a part consumes. When a part consumes an import, the part *imports* it. Parts can import single services, such as the logger, or import multiple services, such as an editor extension.

Contracts—A contract is an identifier for an export or an import. An exporter specifies a string contract that it provides, and an importer specifies the contract that it needs. MEF derives contract names from the types that are being exported and imported, so in most cases you don't have to think about it.

Composition—Parts are composed by MEF, which instantiates them and then matches up exporters to importers.

Programming Models—the Face(s) of MEF

Developers consume MEF through a programming model. A programming model provides a means to declare components as MEF parts. Out of the box, MEF provides an attributed programming model, which will be the main focus of this article. That model is just one of many possible programming models that MEF enables. MEF's core API is completely agnostic to attributes.

Diving into the Attributed Programming Model

In the attributed programming model, parts (known as attributed parts) are defined with a set of .NET attributes, which live in the System.ComponentModel.Composition namespace. In the sections that follow, I'll explore building an extensible Windows Presentation Foundation (WPF) sales order management application using this model. This application allows customers to add new, customized views within their environments simply by deploying a binary to the bin folder. I'll look at how this can be implemented through MEF. I'll incrementally improve the design as I go, and explain more about MEF's capabilities and what the attributed programming model provides in the process.

Exporting a Class

The order management application allows plugging in new views. To export something to MEF, you export it by using the Export attribute as shown here:

```
[Export]
public partial class SalesOrderView : UserControl
{
public SalesOrderView()
  {
InitializeComponent();
  }
}
```

The above part exports the SalesOrderView contract. By default, the Export attribute uses the concrete type of the member (in this case, the class) as the contract. You can also explicitly specify the contract by passing a parameter to the attribute's constructor.

Importing Through Properties and Fields

Attributed parts can express the things they need by using the import attribute on a property or field. The application exports a ViewFactory part, which other parts can use to get to views. That ViewFactory imports SalesOrderView using a property import. Importing a property simply means decorating a property with an Import attribute:

```
[Export]
public class ViewFactory
{
  [Import]
  public SalesOrderView OrderView { get; set; }
}
```

Importing Through Constructors

Parts can also import through constructors (commonly known as constructor injection) by using the ImportingConstructor attribute as shown below. When using an importing constructor, MEF assumes all parameters are imports, making the import attribute unnecessary:

```
[Export]
public class ViewFactory
{
  [ImportingConstructor]
  public ViewFactory(SalesOrderView salesOrderView)
{
}
}
```

In general, importing via constructors rather than properties is a matter of preference, though there are times when it's appropriate to use property imports, particularly when there are parts that aren't instantiated by MEF, as in the WPF App example. Recomposition is also not supported on constructor parameters.

Composition

With SalesOrderView and ViewFactory in place, you can now start composition. MEF parts don't automatically get discovered or created. Instead, you need to write some bootstrapping code that will enact composition. A common place to do this is in your application's entry point, which in this case is the App class.

To bootstrap MEF involves a few steps:

- Add imports of the contracts you need the container to create.
- Create a catalog that MEF uses to discover parts.
- Create a container that composes instances of parts.
- Compose by calling the Composeparts method on the container and passing in the instance that has the imports.

As you can see here, I added a ViewFactory import on the App class. Then I created a DirectoryCatalog pointing to the bin folder and created a container that uses the catalog. Finally, I called Composeparts, which caused an App instance to be composed and the ViewFactory import to be satisfied:

```
public partial class App : Application
{
  [Import]
public ViewFactory ViewFactory { get; set; }

public App()
  {
this.Startup += new StartupEventHandler(App_Startup);
  }

void App_Startup(object sender, StartupEventArgs e)
  {
var catalog = new DirectoryCatalog(@".\");
var container = new CompositionContainer(catalog);
container.Composeparts(this);
  }
}
```

During composition, the container will create the ViewFactory and satisfy its SalesOrderView import. This will result in SalesOrderView being created. Finally, the Application class will have its ViewFactory import satisfied. In this way, MEF has assembled the entire object graph based on declarative information, rather than manually requiring imperative code to do the assembly.

Exporting Non-MEF Items to MEF Through Properties

When integrating MEF into an existing application, or with other frameworks, you will often find non-MEF related class instances (meaning they are not parts) that you will want to make available to importers. These may be sealed framework types such as System.String; application-wide singletons such as Application.Current; or instances retrieved from a factory, such as a logger instance retrieved from Log4Net.

To support this, MEF allows property exports. To use property exports, you create an intermediary part with a property that is decorated with an export. That property is essentially a factory and executes whatever custom logic is necessary to retrieve the non-MEF value. In the following code sample, you can see that Loggerpart exports a Log4Net logger, allowing other parts such as the App to import it rather than depending on accessing the static accessor method:

```
public class Loggerpart
{
  [Export]
public ILog Logger
  {
get { return LogManager.GetLogger("Logger"); }
  }
}
```

Property exports are like Swiss Army knives in their functionality, allowing MEF to play well with others. You will find them extremely useful for integrating MEF into your existing apps and for talking to legacy systems.

Decoupling Implementation with an Interface

Going back to the SalesOrderView example, a tightly coupled relationship has been formed between ViewFactory and SalesOrderView. The factory expects a concrete SalesOrderView that limits extensibility options as well as testability of the factory itself. MEF allows imports to be decoupled from the exporter's implementation by using an interface as the contract:

```
public interface ISalesOrderView{}

[Export(typeof(ISalesOrderView))]
public partial class SalesOrderView : UserControl, ISalesOrderView
{
   ...
}

[Export]
public class ViewFactory
{
  [Import]
ISalesOrderView OrderView{ get; set; }
}
```

In the preceding code, I changed SalesOrderView to implement ISalesOrderView and explicitly export it. I also changed the factory on the importer side to import ISalesOrderView. Notice the importer doesn't have to specify the type explicitly, as MEF can derive it from the property type, which is ISalesOrderView.

This raises the question of whether ViewFactory should also implement an interface like IViewFactory. This isn't a requirement, though it may make sense for mocking purposes. In this case I'm not expecting anyone to replace ViewFactory, and it's designed in a testable fashion, so it's fine. You can have multiple exports on a part in order to have the part imported under multiple contracts. SalesOrderView, for example, can export both UserControl and ISalesOrderView by having an additional export attribute:

```
[Export (typeof(ISalesOrderView))]
[Export (typeof(UserControl))]
public partial class SalesOrderView : UserControl, ISalesOrderView
{
   ...
}
```

Contract Assemblies

As you start to create contracts, you will need a way to deploy those contracts to third parties. A common way to do this is by having a contract assembly that contains interfaces for the contracts that will be implemented by extenders. The contract assembly becomes a form of SDK that the parts will reference. A common pattern is to name the contract assembly as the application name + .Contracts, such as SalesOrderManager.Contracts.

Importing Many Exports of the Same Contract

Currently ViewFactory imports only a single view. Hardcoding a member (property param) for each view works for a very small number of predefined types of views that aren't changing frequently. However, with such an approach, adding new views requires the factory to be recompiled.

If many types of views are expected, MEF offers a better approach. Instead of using a specific view interface, you can create a generic IView interface that all views export. The factory then imports a collection of all available IViews. To import a collection in the attributed model, use the ImportMany attribute:

```
[Export]
public class ViewFactory
{
  [ImportMany]
IEnumerable<IView> Views { get; set; }
}

[Export(typeof(IView))]
public partial class SalesOrderView : UserControl, IView
{
}
//in a contract assembly
public interface IView{}
```

Here you can see that ViewFactory now imports a collection of IView instances rather than a specific view. SalesOrder implements IView and exports it rather than ISalesOrderView. With this refactoring, ViewFactory can now support an open set of views.

MEF also supports importing using concrete collections such as ObservableCollection<T> or List<T>, as well as custom collections that provide a default constructor.

Controlling Part Creation Policy

By default, all part instances in the container are singletons, thus they are shared by any parts that import them within the container. For this reason, all importers of SalesOrderView and ViewFactory will get the same instance. In many cases this is desirable, as it replaces having static members that other components depend on. However, sometimes it's necessary for each importer to get its own instance, for example, to allow multiple SalesOrderView instances to be viewed on the screen at the same time.

Part creation policy in MEF can be one of three values: CreationPolicy.Shared, CreationPolicy.NonShared or CreationPolicy.Any. To specify creation policy on a part, you decorate it with the partCreationPolicy attribute, as shown here:

```
[partCreationPolicy(CreationPolicy.NonShared)]
[Export(typeof(ISalesOrderView))]
public partial class SalesOrderView : UserControl, ISalesOrdderView
{
public SalesOrderView()
  {
  }
}
```

PartCreationPolicy can also be specified on the importer side by setting the RequiredCreationPolicy property on the import.

Distinguishing Exports with Metadata

ViewFactory now works with an open set of views, but I have no way to distinguish one view from another. I could add a member to IView called ViewType, which the view would provide, and then filter against that property. An alternative is to use MEF's export metadata facilities to annotate the view with its ViewType. Using the metadata provides an additional advantage of allowing the view's instantiation to be delayed until it's needed, which can conserve resources and improve performance.

Defining Export Metadata

To define metadata on an export, you use the ExportMetadata attribute. Below, SalesOrderView has been changed to export an IView marker interface as its contract. It then adds additional metadata of "ViewType" so that it can be located among other views that share the same contract:

```
[ExportMetadata("ViewType", "SalesOrder")]
[Export(typeof(IView)]
public partial class SalesOrderView : UserControl, IView
{
}
```

ExportMetadata has two parameters, a key that is a string and a value of type object. Using magic strings as in the preceding example can be problematic because this isn't compile-safe. Instead of a magic string, we can supply a constant for the key and an enum for the value:

```
[ExportMetadata(ViewMetadata.ViewType, ViewTypes.SalesOrder)]
[Export(typeof(IView)]
public partial class SalesOrderView : UserControl, IView
{
  ...
}
//in a contract assembly
public enum ViewTypes {SalesOrderView}

public class ViewMetadata
{
public const string ViewType = "ViewType";
}
```

Using the ExportMetadata attribute provides a lot of flexibility, but there are several caveats when using it:

- Metadata keys aren't discoverable in the IDE. The part author must know which metadata keys and types are valid for the export.
- The compiler won't validate metadata to ensure it's correct.
- ExportMetadata adds more noise to the code, hiding the intent.

MEF provides a solution to address the above issues: custom exports.

Custom Export Attributes

MEF allows the creation of custom exports that include their own metadata. Creating a custom export involves creating a derived ExportAttribute that also specifies metadata. We can use custom exports to create an ExportView attribute that includes metadata for ViewType:

```
[MetadataAttribute]
[AttributeUsage(AttributeTargets.Class, AllowMultiple=false)]
public class ExportViewAttribute : ExportAttribute {
public ExportViewAttribute()
:base(typeof(IView))
  {}

public ViewTypes ViewType { get; set; }
}
```

ExportViewAttribute specifies that it exports IView by calling Export's base constructor. It's decorated with a MetadataAttribute, which specifies that the attribute provides metadata. This attribute tells MEF to look at all of the public properties and create associated metadata on the export using the property name as the key. In this case, the only metadata is ViewType.

The last important thing to note about the ExportView attribute is that it's decorated with an AttributeUsage attribute. This specifies that the attribute is valid only on classes and that only a single ExportView attribute can be present.

In general, AllowMultiple should be set to false; if it's true, the importer will be passed an array of values rather than a single value. AllowMultiple should be left as true when there are multiple exports with different metadata of the same contract on the same member.

Applying the new ExportViewAttribute to the SalesOrderView now results in the following:

```
[ExportView(ViewType = ViewTypes.SalesOrder)]
public partial class SalesOrderView : UserControl, IView
{
}
```

As you can see, custom exports ensure the correct metadata is provided for a particular export. They also reduce noise in the code, are more discoverable through IntelliSense and better express intent through being domain-specific.

Now that metadata has been defined on the view, the ViewFactory can import it.

Importing Lazy Exports and Accessing Metadata

To allow the accessing of metadata, MEF leverages a new API of the .NET Framework 4, System.Lazy<T>. It allows delaying the instantiation of an instance until the value property of the Lazy is accessed. MEF further extends Lazy<T> with Lazy<T,TMetadata> to allow accessing export metadata without instantiating the underlying export.

TMetadata is a metadata view type. A metadata view is an interface that defines read-only properties that correspond to keys in the exported metadata. When the metadata property is accessed, MEF will dynamically implement TMetadata and will set the values based on the provided metadata from the export.

This is how ViewFactory looks when the View property is changed to import using Lazy<T,TMetadata>:

```
[Export]
public class ViewFactory
{
  [ImportMany]
IEnumerable<Lazy<IView, IViewMetadata>> Views { get; set; }
}

public interface IViewMetadata
{
ViewTypes ViewType {get;}
}
```

Once a collection of lazy exports with metadata has been imported, you can use LINQ to filter against the set. In the following code snippet, I've implemented a GetViews method on ViewFactory to retrieve all views of the specified type. Notice it accesses the Value property in order to manufacture the real view instances only for the views that match the filter:

```
[Export]
public class ViewFactory
{
  [ImportMany]
IEnumerable<Lazy<IView, IViewMetadata>> Views { get; set; }

public IEnumerable<View> GetViews(ViewTypesviewType) {
return Views.Where(v=>v.Metadata.ViewType.Equals(viewType)).Select(v=>v.
Value);
  }
}
```

With these changes, ViewFactory now discovers all views that are available *at the time the factory is composed by MEF*. If new implementations appear in the container or catalogs after that initial composition, they won't be seen by the ViewFactory, as it was already composed. Not only that, but MEF will actually prevent the views from being added to the catalog by throwing a CompositionException—that is, unless recomposition is enabled.

Recomposition

Recomposition is a feature of MEF that allows parts to automatically have their imports updated as new matching exports appear in the system. One scenario where recomposition is useful is for downloading parts from a remote server. SalesOrderManager can be changed so that when it starts up, it initiates a download for several optional views. As the views show up, they will appear in the view factory. To make the ViewFactory recomposable, we set the AllowRecomposition property on the ImportMany attribute of the Views property to true, as shown here:

```
[Export]
public class ViewFactory
{
[ImportMany(AllowRecomposition=true)]
IEnumerable<Lazy<IView, IViewMetadata>> Views { get; set; }

public IEnumerable<View>GetViews(ViewTypesviewType) {
return Views.Where(v=>v.Metadata.ViewType.Equals(viewType)).Select(v=>v.
Value);
  }
}
```

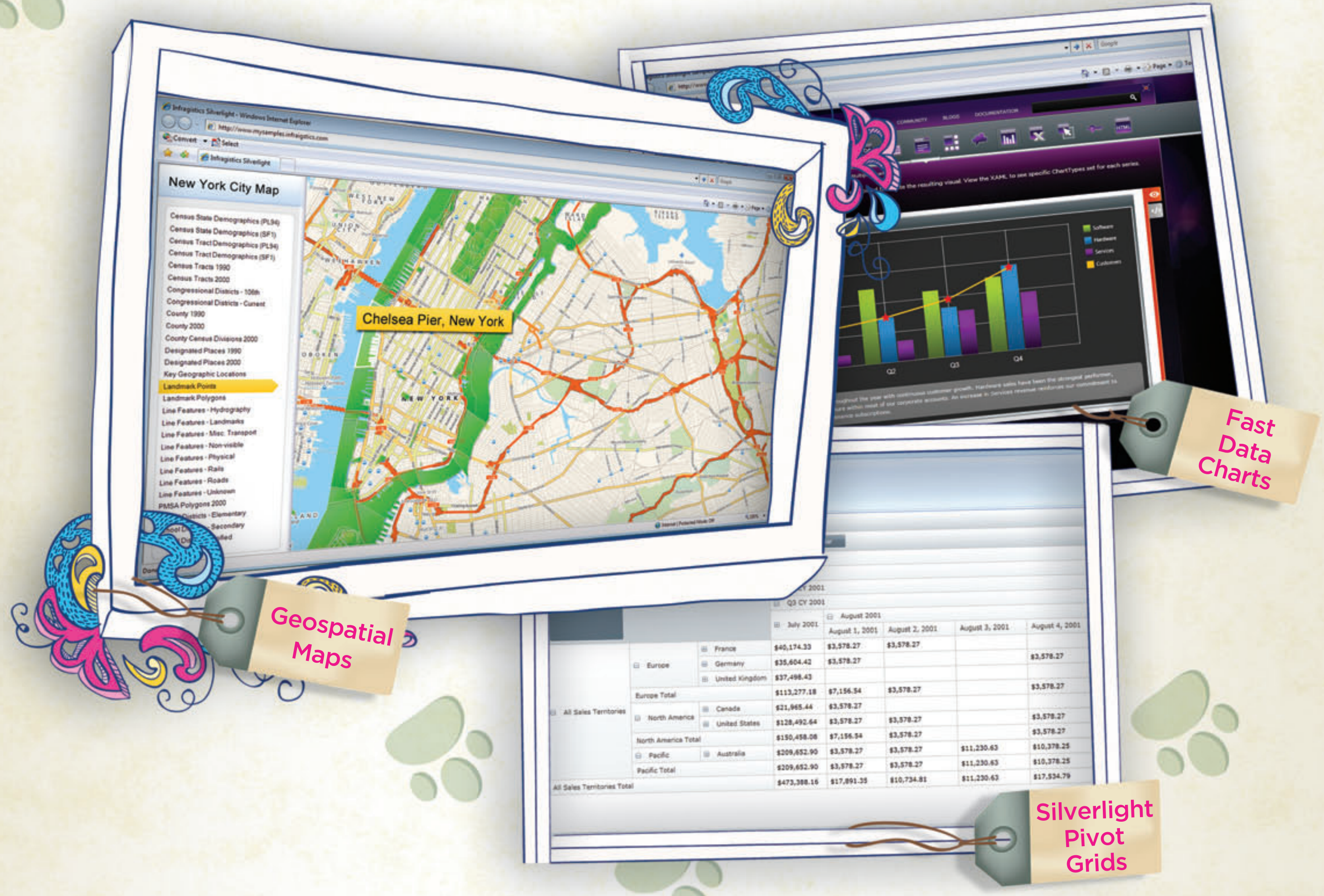
When recomposition occurs, the Views collection will instantly be replaced with a new collection that contains the updated set of views.

With recomposition enabled, the app can download additional assemblies from the server and add them to the container. You can do this through MEF's catalogs. MEF offers several catalogs, two of which are recomposable. DirectoryCatalog, which you have already seen, is one that is recomposed by calling its Refresh method. Another recomposable catalog is AggregateCatalog, which is a catalog of catalogs. You add catalogs to it by using the Catalogs collection property, which starts recomposition. The last catalog I'll use is an AssemblyCatalog, which accepts an assembly upon which it then builds a catalog. **Figure 2** shows a sample illustrating how you can use these catalogs together for dynamic download.

The container in **Figure 2** is created with an AggregateCatalog. It then has a DirectoryCatalog added to it in order to grab the local parts in the bin folder. The aggregate catalog is passed to the DownloadAssemblies method, which asynchronously downloads assemblies and then calls AddAssemblies. That method creates a new AggregateCatalog, to which it adds AssemblyCatalogs for each download assembly. AddAssemblies then adds the AggregateCatalog containing the assemblies for the main aggregate. The reason it adds in this fashion is to have recomposition occur in one shot, rather than over and over again, which is what would happen if we added assembly catalogs directly.

When recomposition occurs, the collection is immediately updated. Depending on the collection property type, the result is different. If the property is of type IEnumerable<T>, it's replaced with a new instance. If it's a concrete collection that inherits from List<T> or ICollection, then MEF will call Clear and then Add for each item. In either case, it means you will have to consider thread-

INSPIRE! EMPOWER! IMPRESS!



			Q3 CY 2001				
				August 2001			
			July 2001	August 1, 2001	August 2, 2001	August 3, 2001	August 4, 2001
All Sales Territories	Europe	France	$40,174.33	$3,578.27	$3,578.27		
		Germany	$35,604.42	$3,578.27			$3,578.27
		United Kingdom	$37,498.43				
	Europe Total		$113,277.18	$7,156.54	$3,578.27		$3,578.27
	North America	Canada	$21,965.44	$3,578.27			
		United States	$128,492.64	$3,578.27	$3,578.27		$3,578.27
	North America Total		$150,458.08	$7,156.54	$3,578.27		$3,578.27
	Pacific	Australia	$209,652.90	$3,578.27	$3,578.27	$11,230.63	$10,378.25
	Pacific Total		$209,652.90	$3,578.27	$3,578.27	$11,230.63	$10,378.25
All Sales Territories Total			$473,388.16	$17,891.35	$10,734.81	$11,230.63	$17,534.79

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**, you can create Web-based data visualizations and dashboard-driven applications on Microsoft Silverlight (and coming soon for 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

safety when using Recomposition. Recomposition not only relates to adds, it also relates to removals. If catalogs are removed from the container, those parts will also be removed .

Stable Composition, Rejection and Diagnostics

Sometimes a part may specify an import that is missing, as it isn't present in the catalog. When this happens, MEF prevents the part missing the dependency—or anything that depends on it—from being discovered. MEF does this in order to stabilize the system and prevent runtime failures that would surely occur if the part were created.

Here, SalesOrderView has been changed to import an ILogger though there's no logger instance present:

```
[ExportView(ViewType = ViewTypes.SalesOrder)]
public partial class SalesOrderView : UserControl, IView
{
[Import]
public ILogger Logger { get; set; }
}
```

Because there isn't an ILogger export available, SalesOrderView's export won't appear to the container. This won't throw an exception; instead SalesOrderView will just be ignored. If you check ViewFactory's Views collection, it will be empty.

Rejection will also happen in cases where there are multiple exports available for a single import. In those cases, the part that imports the single export is rejected:

```
[ExportView(ViewType = ViewTypes.SalesOrder)]
public partial class SalesOrderView : UserControl, IView
{
[Import]
public ILogger Logger { get; set; }
}
 [Export(typeof(ILogger))]
public partial class Logger1 : ILogger
{
}
 [Export(typeof(ILogger))]
public partial class Logger2 : ILogger
{
}
```

In the preceding example, SalesOrderView will be rejected because there are multiple ILogger implementations, but a single implementation is imported. MEF does provide facilities for allowing a default export in the presence of multiples. For more on this, see codebetter.com/blogs/glenn.block/archive/2009/05/14/customizing-container-behavior-part-2-of-n-defaults.aspx.

You might ask why MEF doesn't create SalesOrderView and throw an exception. In an open extensible system, if MEF throws an exception, it would be very difficult for the application to handle it, or to have the context to know what to do, because the part might be missing or the import might be nested very deeply in the composition. Without proper handling, the app would be in an invalid state and unusable. MEF rejects the part, thus ensuring application stability is maintained. For more on stable composition, see: blogs.msdn.com/gblock/archive/2009/08/02/stable-composition-in-mef-preview-6.aspx.

Diagnosing Rejection

Rejection is a very powerful feature, but it can sometimes be difficult to diagnose, especially when the entire dependency graph is rejected. In the first early example, ViewFactory directly imports a SalesOrderView. Let's say MainWindow imported ViewFactory and SalesOrderView is rejected. Then ViewFactory and MainWindow will also get rejected. You might be scratching your head if you see this occur, as you know that MainWindow and ViewFactory actually are present; the reason for the rejection is a missing dependency.

MEF doesn't leave you in the dark. To assist with diagnosing this problem, it provides tracing. In the IDE, all rejection messages are traced to the output window, though they can also be traced to any valid trace listener. For example, when the app attempts to import MainWindow, the trace messages in **Figure 3** will be outputted.

The trace output shows the root cause of the problem: SalesOrderView requires an ILogger and one cannot be located. We can then see that rejecting it caused the factory to be rejected, and ultimately the MainWindow.

Inspecting Parts in the Debugger

You can go one step further and actually inspect the available parts in the catalog, which I'll discuss in the section on hosting. In **Figure 4** you can see in the watch window the available parts (in the green circles) as well as the required ILogger import (in the blue circle).

Diagnosing Rejection at the Command Line

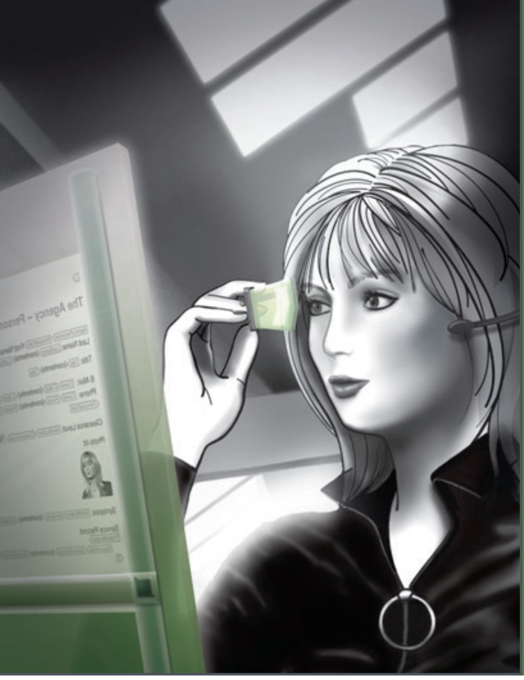
One of the goals of MEF was to support static analyzability, thus allowing composition to be analyzed outside of the runtime

Figure 2 **Using MEF Catalogs for Dynamic Download**

```
void App_Startup(object sender, StartupEventArgs e)
{
var catalog = new AggregateCatalog();
catalog.Catalogs.Add(newDirectoryCatalog((@"\.")));
var container = new CompositionContainer(catalog);
container.Composeparts(this);
base.MainWindow = MainWindow;
this.DownloadAssemblies(catalog);
}

private void DownloadAssemblies(AggregateCatalog catalog)
{
//asynchronously downloads assemblies and calls AddAssemblies
}

private void AddAssemblies(Assembly[] assemblies, AggregateCatalog catalog)
{
var assemblyCatalogs = new AggregateCatalog();
foreach(Assembly assembly in assemblies)
assemblyCatalogs.Catalogs.Add(new AssemblyCatalog(assembly));
catalog.Catalogs.Add(assemblyCatalogs);
}
```

Figure 3 **MEF Trace Messages**

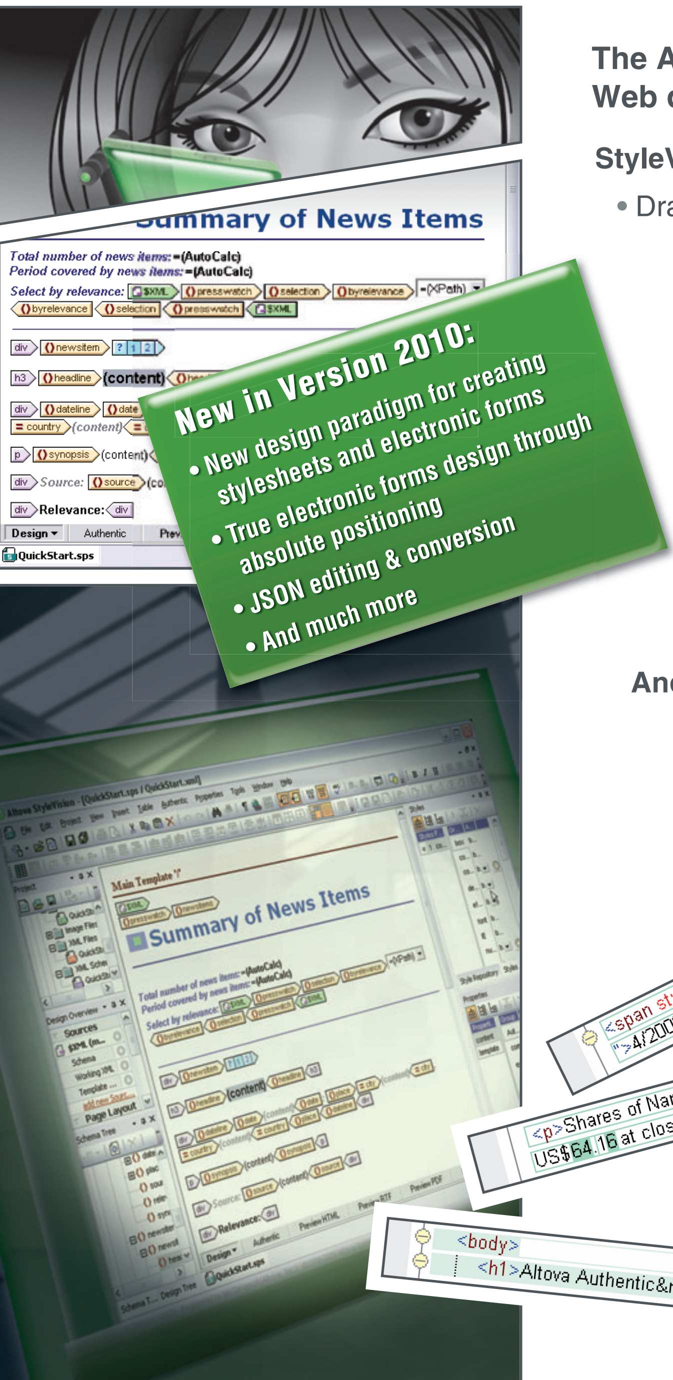
```
System.ComponentModel.Composition Warning: 1 : The
ComposablepartDefinition 'Mef_MSDN_Article.SalesOrderView' has been
rejected. The composition remains unchanged. The changes were rejected
because of the following error(s): The composition produced a single
composition error. The root cause is provided below. Review the
CompositionException.Errors property for more detailed information.

1) No valid exports were found that match the constraint
'((exportDefinition.ContractName == "Mef_MSDN_Article.ILogger") AndAlso
(exportDefinition.Metadata.ContainsKey("ExportTypeIdentity") AndAlso
"Mef_MSDN_Article.ILogger".Equals(exportDefinition.Metadata.get_
Item("ExportTypeIdentity"))))', invalid exports may have been rejected.

Resulting in: Cannot set import 'Mef_MSDN_Article.SalesOrderView.Logger
(ContractName="Mef_MSDN_Article.ILogger")' on part 'Mef_MSDN_Article.
SalesOrderView'.
Element: Mef_MSDN_Article.SalesOrderView.logger (ContractName="Mef_MSDN_
Article.ILogger") -->Mef_MSDN_Article.SalesOrderView -->TypeCatalog
(Types='Mef_MSDN_Article.MainWindow, Mef_MSDN_Article.SalesOrderView,
...').
```



ALTOVA® missionkit®

Uncover
Web development
simplicity with the
complete set of tools
from Altova®

Experience how the Altova MissionKit® suite of integrated XML and database tools can simplify even the most advanced Web 2.0 development projects.


Summary of News Items
Total number of news items: =(AutoCalc)
Period covered by news items: =(AutoCalc)
Select by relevance:
Design
Authentic
QuickStart.sps

New in Version 2010:

- New design paradigm for creating stylesheets and electronic forms
- True electronic forms design through absolute positioning
- JSON editing & conversion
- And much more

The Altova MissionKit includes powerful Web development tools:

StyleVision® – graphical stylesheet & electronic forms designer

- Drag-and-drop XSLT 1.0/2.0 and XSL-FO stylesheet design
- Advanced CSS and JavaScript functionality
- Precise electronic forms design

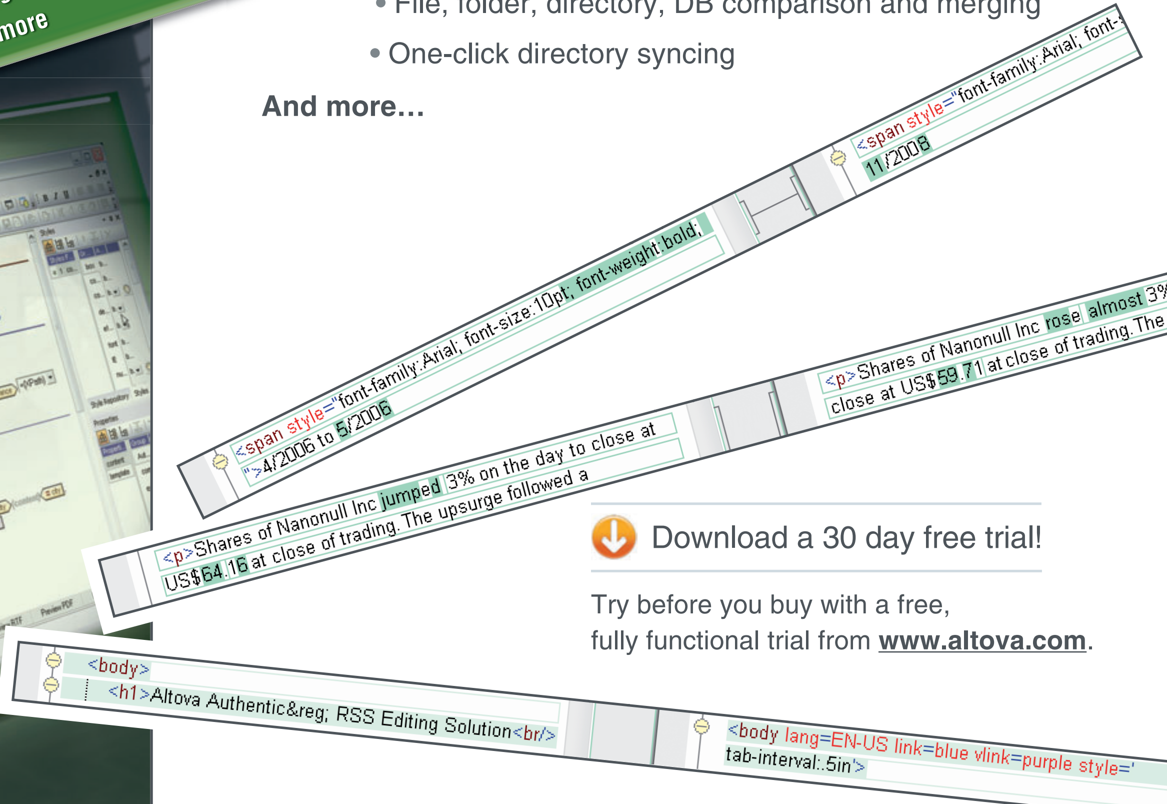
XMLSpy® – advanced XML editor

- XSLT 1.0/2.0 editing, debugging, and profiling
- XSLT editing help, support for program code in stylesheets

DiffDog® – XML-aware diff / merge utility

- File, folder, directory, DB comparison and merging
- One-click directory syncing

And more…



```
<span style="font-family:Arial; font-
11/2008
<span style="font-family:Arial; font-size:10pt; font-weight:bold;
">4/2006 to 5/2006
<p>Shares of Nanonull Inc rose almost 3%
close at US$59.71 at close of trading. The
<p>Shares of Nanonull Inc jumped 3% on the day to close at
US$64.16 at close of trading. The upsurge followed a
<body>
<h1>Altova Authentic&reg; RSS Editing Solution<br/>
<body lang=EN-US link=blue vlink=purple style='
tab-interval:.5in'>
```

Download a 30 day free trial!

Try before you buy with a free, fully functional trial from **www.altova.com**.

environment. We don't yet have such tooling support in Visual Studio, however Nicholas Blumhardt authored MEFX.exe (mef.codeplex.com/Release/ProjectReleases.aspx?ReleaseId=33536), a command-line tool that does the trick. MEFX analyzes assemblies and determines which parts are being rejected and why.

If you run MEFX.exe at the command line, you will see a host of options; you can list specific imports, exports or all parts available. For example, here you can see using MEFX to display the list of parts:

```
C:\mefx>mefx.exe /dir:C:\SalesOrderManagement\
bin\debug /parts
SalesOrderManagement.SalesOrderView
SalesOrderManagement.ViewFactory
SalesOrderManagement.MainWindow
```

This is useful for getting a part inventory, but MEFX can also track down rejections, which is our interest here, as shown in **Figure 5**.

Dissecting the output in **Figure 6** reveals the root cause of the problem: ILogger can't be located. As you can see, in large systems with many parts, MEFX is an invaluable tool. For more on MEFX, see blogs.msdn.com/nblumhardt/archive/2009/08/28/analyze-mef-assemblies-from-the-command-line.aspx.

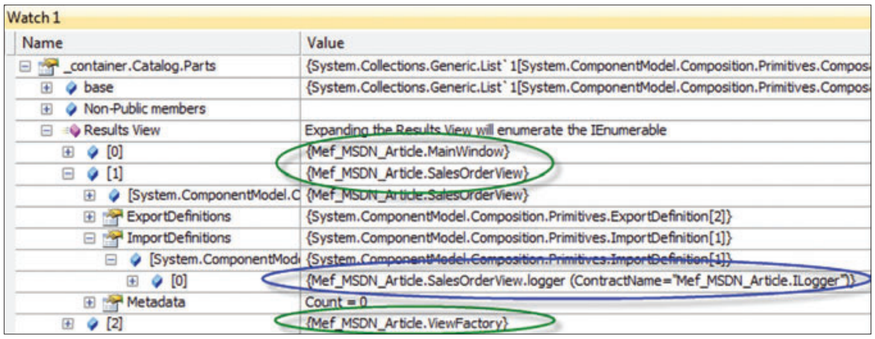
Summarizing, there are several advantages of the attributed model:

- It provides a universal way for parts to declare their exports and imports.
- It allows systems to dynamically discover available parts rather than requiring preregistration.
- It's statically analyzable, allowing tools like MEFX to determine failures ahead of time.

I'll now take a quick tour of the architecture and see what it enables. At a high level, MEF's architecture is broken into layers: Programming Models, Hosting and Primitives.

Figure 5 **Tracking Down Rejections with MEFX.exe**

```
C:\mefx>mefx.exe /dir:C:\SalesOrderManagement\bin\debug /rejected /
verbose

[part] SalesOrderManagement.SalesOrderView from: DirectoryCatalog
(Path="C:\SalesOrderManagement\bin\debug")
  [Primary Rejection]
  [Export] SalesOrderManagement.SalesOrderView (ContractName="SalesOrder
Management.IView")
  [Export] SalesOrderManagement.SalesOrderView (ContractName="SalesOrder
Management.IView")
  [Import] SalesOrderManagement.SalesOrderView.logger (ContractName="Sal
esOrderManagement.ILogger")
    [Exception] System.ComponentModel.Composition.
ImportCardinalityMismatchException: No valid exports were found
that match the constraint '((exportDefinition.ContractName ==
"SalesOrderManagement.ILogger") AndAlso (exportDefinition.Metadata.Con
tainsKey("ExportTypeIdentity") AndAlso "SalesOrderManagement.ILogger".
Equals(exportDefinition.Metadata.get_Item("ExportTypeIdentity"))))',
invalid exports may have been rejected.
at System.ComponentModel.Composition.Hosting.
ExportProvider.GetExports(ImportDefinition definition,
AtomicCompositionatomicComposition)
at System.ComponentModel.Composition.Hosting.ExportProvider.
GetExports(ImportDefinition definition)
at Microsoft.ComponentModel.Composition.Diagnostics.CompositionInfo.Ana
lyzeImportDefinition(ExportProvider host, IEnumerable`1 availableparts,
ImportDefinition id)
```


Watch 1

Name	Value
_container.Catalog.Parts	{System.Collections.Generic.List` 1[System.ComponentModel.Composition.Primitives.Compos
base	{System.Collections.Generic.List` 1[System.ComponentModel.Composition.Primitives.Compos
Non-Public members	
Results View	Expanding the Results View will enumerate the IEnumerable
[0]	{Mef_MSDN_Article.MainWindow}
[1]	{Mef_MSDN_Article.SalesOrderView}
[System.ComponentModel.C	{Mef_MSDN_Article.SalesOrderView}
ExportDefinitions	{System.ComponentModel.Composition.Primitives.ExportDefinition[2]}
ImportDefinitions	{System.ComponentModel.Composition.Primitives.ImportDefinition[1]}
[System.ComponentMode	{System.ComponentModel.Composition.Primitives.ImportDefinition[1]}
[0]	{Mef_MSDN_Article.SalesOrderView.logger (ContractName="Mef_MSDN_Article.ILogger")}
Metadata	Count = 0
[2]	{Mef_MSDN_Article.ViewFactory}

Figure 4 **Available Parts and Required ILogger Shown in a Watch Window**

Programming Models Revisited

The attributed model is simply one implementation of those primitives that uses attributes as the means of discovery. The primitives can represent non-attributed parts or even parts that aren't statically typed, as in the Dynamic Language Runtime (DLR). In **Figure 6** you can see an IronRuby part that exports an IOperation. Notice that it uses IronRuby's native syntax for declaring a part rather than the attributed model, as attributes aren't supported in the DLR.

MEF does not ship with an IronRuby programming model, though it's likely we will add dynamic language support in the future.

You can read more about experiments in building a Ruby programming model in this blog series: blogs.msdn.com/nblumhardt/archive/tags/Ruby/default.aspx.

Hosting: Where Composition Happens

Programming models define parts, imports and exports. In order to actually create instances and object graphs, MEF provides hosting APIs that are primarily located in the System.ComponentModel.Composition.Hosting namespace. The hosting layer offers a lot of flexibility, configurability and extensibility. It's the place where much of the "work" in MEF is done and where discovery in MEF begins. Most folks who are simply authoring parts will never touch this namespace. If you are a hoster, however, you'll be using them as I did earlier in order to bootstrap composition.

Catalogs provide part definitions (ComposablepartDefinition), which describe the available exports and imports. They are the main unit for discovery in MEF. MEF provides several catalogs in the System.ComponentModel.Composition namespace, some of which you already saw, including DirectoryCatalog, which scans a directory; AssemblyCatalog, which scans an assembly; and TypeCatalog, which scans through a specific set of types. Each of these catalogs is specific to the attributed programming model. The AggregateCatalog, however, is agnostic to programming models. Catalogs inherit from ComposablepartCatalog and are an extensibility point in MEF. Custom catalogs have several uses, from providing a completely new programming model to wrapping and filtering existing catalogs.

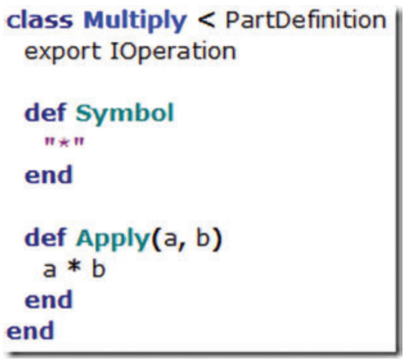
Figure 7 shows an example of a filtered catalog, which accepts a predicate to filter against the inner catalog from which parts will be returned.

The CompositionContainer composes, meaning it creates parts and satisfies imports of those parts. In satisfying the imports, it will grab from the pool of available exports. If those exports also have imports, the container will satisfy them first. In this way, the container assembles entire object graphs on demand. The primary source for the pool of exports is a catalog, but the container can also have existing part instances directly added to it and composed. It's customary to manually add the entry point class to the container, combined with parts pulled from the catalog, though in most cases parts will come from the catalog.

Containers can also be nested in a hierarchy in order to support scoping scenarios. Child containers by default will query the parent, but they can also provide their own catalogs of child parts, which will be created within the child container:

```
var catalog = new DirectoryCatalog(@".\");
var childCatalog = new DirectoryCatalog(@".\Child\";
var rootContainer = new CompositionContainer(rootCatalog));
var childContainer = new CompositionContainer(childCatalog,
rootContainer);
```

In the preceding code, childContainer is arranged as a child of rootContainer. Both rootContainer and childContainer provide their own catalogs.



```
class Multiply < PartDefinition
  export IOperation

  def Symbol
    "*"
  end

  def Apply(a, b)
    a * b
  end
end
```

Figure 6 **A Part Example in IronRuby**

The Primitives: Where Parts and Programming Models Are Born

The primitives located at System.ComponentModel.Composition. Primitives are the lowest level in MEF. They are the quantum universe of MEF, so to speak, and its über extensibility point. Up until now, I've covered the attributed programming model. MEF's container, however, isn't at all bound to attributes; instead, it's bound to the primitives. The primitives define an abstract representation of parts, which includes definitions such as ComposablepartDefinition, ImportDefinition and ExportDefinition, as well as Composablepart and Export, which represent actual instances.

Exploring the primitives is a journey of its own, and one I'll likely take in a future article. For now, you can find out more about it at blogs.msdn.com/dsplaisted/archive/2009/06/08/a-crash-course-on-the-mef-primitives.aspx.

Figure 7 **A Filtered Catalog**

```
public class FilteredCatalog : ComposablepartCatalog,
{
private readonly composablepartcatalog _inner;
private readonly IQueryable<ComposablepartDefinition> _partsQuery;

public FilteredCatalog(ComposablepartCatalog inner,
Expression<Func<ComposablepartDefinition, bool>> expression)
  {
      _inner = inner;
    _partsQuery = inner.parts.Where(expression);
  }

public override IQueryable<ComposablepartDefinition> parts
  {
get
      {
return _partsQuery;
      }
  }
}
```

MEF in Silverlight 4—and Beyond

MEF also ships as part of Silverlight 4. Everything I discussed here is relevant to developing extensible Rich Internet Applications. In Silverlight, we've gone even further and introduced additional APIs to ease the experience of building apps on MEF. These enhancements will eventually be rolled into the .NET Framework.

You can find out more about MEF in Silverlight 4 in this post: codebetter.com/blogs/glenn.block/archive/2009/11/29/mef-has-landed-in-silverlight-4-we-come-in-the-name-of-extensibility.aspx.

I've just scratched the surface of what you can do with MEF. It's a powerful, robust and flexible tool that you can add to your arsenal to help open up your applications to a whole new world of possibilities. I look forward to seeing what you do with it! ■

GLENN BLOCK *is a PM for the new Managed Extensibility Framework (MEF) in the .NET Framework 4. Before MEF he was a product planner in patterns & practices responsible for Prism as well as other client guidance. Block is a geek at heart and spends a good portion of his time spreading that geekdom through conferences and groups such as ALT.NET. Read his blog at codebetter.com/blogs/glenn.block.*

THANKS to the following technical expert for reviewing this article: *Hamilton Verissimo*

Writing and Testing VPL Services for Serial Communication

Trevor Taylor

Microsoft Robotics Developer Studio (RDS) is, as you'd expect, a platform for programming robots. RDS first shipped in 2006 and the latest version, RDS 2008 R2, was released in June 2009.

RDS consists of four major components: the Concurrency and Coordination Runtime (CCR), Decentralized Software Services (DSS), Visual Programming Language (VPL) and Visual Simulation Environment (VSE). Sara Morgan wrote about the VSE in the June 2008 issue of *MSDN Magazine* (msdn.microsoft.com/magazine/cc546547).

VPL, however, is more pertinent to this article. VPL is a dataflow language, which means you create programs by drawing diagrams on the screen. At run time, messages flow from one block to another in the diagram, and this dataflow is effectively the execution path of the program. Because VPL is built on top of CCR and DSS, dataflows can occur asynchronously (as a result of notifications from services) and can also run in parallel. While VPL is intended for novice programmers, experienced coders can also find it useful for for prototyping.

This article discusses:

- Communicating with robots
- Configuring a serial port service
- Using the service
- Testing with VPL

Technologies discussed:

Microsoft Robotics Developer Studio

Code download available at:

code.msdn.microsoft.com/mag201002Testing

This article outlines a simple RDS service that allows you to send and receive data using a serial port (also known as a COM port). The example code illustrates some of the key concepts involved in writing reusable RDS services.

To learn more about RDS and download the platform, go to microsoft.com/robotics. The package includes a useful help file, which is also available at msdn.microsoft.com/library/dd936006. You can get further help by posting questions to the various Robotics Discussion Forums at social.msdn.microsoft.com/Forums/en-us/category/robotics.

RDS Services

RDS services are built using CCR and DSS. They are conceptually similar to Web services because RDS has a service-oriented architecture (SOA) based on a Representational State Transfer (REST) model using the Decentralized Software Services Protocol (DSSP) for communication between services. Wading through all this alphabet soup, what this means is that you don't talk directly to RDS services. Instead, you send messages to a proxy, which acts as the external interface to the service (an approach Web developers will be familiar with). It also means that services can be distributed anywhere on the network.

Using a proxy has two effects. First, messages sent between services are serialized before transmission and deserialized at the other end. XML is the usual format for the data being transmitted. Second, the proxy defines a contract—effectively the set of APIs that are exposed to other services.

Every service has an internal state, which you can retrieve or modify by operations on the service. These involve sending a request message and waiting for a response message in a process similar to that used by most Web services.

When a service's state changes, it can send out notifications to subscribers. This publish-and-subscribe approach makes RDS services different from traditional Web services because notification messages are sent asynchronously to subscribers.

When you build a new service, it automatically becomes visible in VPL and you can start using it immediately. This is one of the key RDS features, and it makes testing and prototyping very easy—you don't have to write test harnesses in C#, because you can use VPL instead.

Controlling a Robot Remotely

Many simple educational robots have 8- or 16-bit microcontrollers for their on-board brains. But because RDS runs under the .NET Framework on Windows, it doesn't generate code that can run directly on these robots. They must be controlled remotely via a communications link, instead. (The alternative is to have an on-board PC, such as the MobileRobots Pioneer 3DX).

Since the majority of microcontrollers support serial ports, a serial connection is the obvious solution. However, supplying the link using a cable isn't ideal—it limits the robot's mobility.

As an alternative, you can use Bluetooth to create a wireless connection by installing a serial-to-Bluetooth device on the robot. Some robots, such as the LEGO NXT, come with Bluetooth built in. Others, such as the RoboticsConnection Stinger, have optional Bluetooth modules. Bluetooth is a good choice, given that it's standard on most laptops these days. Even if your PC doesn't have Bluetooth, you'll find inexpensive, readily available USB Bluetooth devices.

The good news is that you don't have to know anything about programming Bluetooth because the connection to the robot will appear as a virtual serial port. You can use the same code as you would if a physical cable were providing the serial connection. The only difference is that you have to establish a pairing between your PC and the Bluetooth device on the robot so a virtual COM port will be created.

Some robot controller boards have firmware that can accept commands using a simple command language. For example, with the Serializer from RoboticsConnection (which gets its name from its use of a serial protocol), you can type commands to the controller via a terminal program like HyperTerminal. Commands are all human-readable, and you terminate each by pressing *Enter*.

If you're designing your own protocol for use with a robot, you need to make a few choices. First, you must decide whether you'll send binary data. Converting binary values to hexadecimal or decimal for transmission requires more bandwidth and increases the processing overhead for the on-board CPU. On the other hand, it makes reading the messages much easier, and you won't experience any strange behavior due to misinterpreted control characters.

Figure 1 **SerialPortState**

```
[DataContract]
public class SerialPortState {
  private bool _openOnStart;
  private bool _asynchronous;
  private SerialPortConfig _config;
  private byte _lastByteReceived;
  private bool _isOpen;

  // Open the COM port when the service starts
  // Must be set in config file
  [DataMember]
  public bool OpenOnStart {
    get { return _openOnStart; }
    set { _openOnStart = value; }
  }

  // Operate in Asynchronous mode
  [DataMember]
  public bool Asynchronous {
    get { return _asynchronous; }
    set { _asynchronous = value; }
  }

  // Configuration parameters for the serial port
  [DataMember]
  public SerialPortConfig Config {
    get { return _config; }
    set { _config = value; }
  }

  // Last byte received from the serial port
  [DataMember]
  public byte LastByteReceived {
    get { return _lastByteReceived; }
     set { _lastByteReceived = value; }
  }

  // Indicates if the port is currently open
  [DataMember]
  public bool IsOpen {
    get { return _isOpen; }
    set { _isOpen = value; }
  }
}
```

The next question is whether you want to use fixed-length packets or a more flexible variable-length format. Fixed-length is easier to parse and works best with hexadecimal.

You should also consider whether you'll use a checksum. For computer-to-computer communication calculating check digits is easy. If, however, you want to test your robot by typing in commands manually, figuring out the check digits gets very tricky. When using checksums, typically the receiver sends back an acknowledgement (ACK) or negative acknowledgement (NAK) depending on whether the command came across successfully or not. Your decision about using a checksum really comes down to the reliability of the connection.

The SerialPort Service

It should be apparent by now why a serial-port service would be useful. The RDS package, however, doesn't include such a service, although many of the robot samples use serial communication links. If you explore the RDS sample code, you'll find that each example handles the serial port differently. This article outlines one approach to using a serial port. It's not the only way to do it and not necessarily the best way.

Before going any further, make sure you've downloaded and installed RDS. The download for this article contains the source code of the SerialPort service. Unzip it into a folder under your RDS installation directory (also known as the mount point). Note that you should keep your code separate from the samples folder that comes with RDS so that you don't mix up your code with the Microsoft

Figure 2 **Event Handler**

```
void DataReceivedHandler(object sender,
  SerialDataReceivedEventArgs e) {

  int data;
  while (sp.BytesToRead > 0) {
    // Read a byte - this will return immediately because
    // we know that there is data available
    data = sp.ReadByte();
    // Post the byte to our own main port so that
    // notifications will be sent out
    ReceiveByte rb = new ReceiveByte();
    rb.Body.Data = (byte)data;
    _mainPort.Post(rb);
    Activate(Arbiter.Choice(rb.ResponsePort,
      success => { },
      fault => { LogError("ReceiveByte failed"); }
      ));
  }
}
```

Figure 3 **WriteStringHandler**

```
[ServiceHandler]
public virtual IEnumerator<ITask> WriteStringHandler(
  WriteString write) {

  if (!_state.IsOpen) {
    throw (new Exception("Port not open"));
  }

  // Check the parameters - An empty string is valid, but not null
  if (write.Body.DataString == null)
    throw (new Exception("Invalid Parameters"));

  // NOTE: This might hang forever if the comms link is broken
  // and you have not set a timeout. On the other hand, if there
  // is a timeout then an exception will be raised which is
  // handled automatically by DSS and returned as a Fault.
  if (_state.Config.InterCharacterDelay > 0) {
    byte[] data = new byte[1];
    for (int i = 0; i < write.Body.DataString.Length; i++) {
      data[0] = (byte)write.Body.DataString[i];
      sp.Write(data, 0, 1);
      yield return Timeout(_state.Config.InterCharacterDelay);
    }
    sp.WriteLine("");
  }
  else
    sp.WriteLine(write.Body.DataString);

  // Send back an acknowledgement now that the data is sent
  write.ResponsePort.Post(DefaultSubmitResponseType.Instance);
  yield break;
}
```

code. Also, I recommend placing your code under the RDS mount point rather than elsewhere on your hard drive, because it simplifies development. I have a Projects folder where I keep all my own code in subfolders, making it easy to back up.

The main files in the SerialPort source code are SerialPort.cs, SerialPortTypes.cs and SerialPort.manifest.xml.

SerialPort.cs holds the implementation of the service, or the behavior. It consists of the service class itself, which contains the operation handlers, plus any required supporting methods and variables.

SerialPortTypes.cs contains the definitions of the service state, the operation types, and message types. In effect, it describes the interface to the service, and it should not contain any executable code. This is a key point about a service contract—it's a data definition only.

SerialPort.manifest.xml is called the manifest and describes how services are combined, or orchestrated, to run an application. This file is used as input to the DssHost utility, which creates a DSS node that services run inside. In this case the manifest only runs the SerialPort service, which is fairly useless. A useful manifest would also specify other services that rely on the SerialPort service to make connections.

Before using the SerialPort service, you must run the DssProjectMigration tool and then recompile the service. Open a DSS Command Prompt (look under RDS in the Start menu) and make sure your paths are set up so you can execute files in the \bin folder. Then change to the directory where you unzipped the code and enter the following command:

```
Dssprojectmigration /b- .
```

The /b- option means don't create a backup copy and the period character (.) refers to the current directory.

Figure 4 **Config File for SerialPortState**

```
<?xml version="1.0" encoding="utf-8"?>
<SerialPortState
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns=http://www.promrds.com/contracts/2009/12/serialport.html
  >
  <OpenOnStart>false</OpenOnStart>
  <Asynchronous>false</Asynchronous>
  <Config>
    <PortNumber>1</PortNumber>
    <BaudRate>57600</BaudRate>
    <Parity>None</Parity>
    <DataBits>8</DataBits>
    <StopBits>One</StopBits>
    <ReadTimeout>0</ReadTimeout>
    <WriteTimeout>0</WriteTimeout>
    <InterCharacterDelay>0</InterCharacterDelay>
  </Config>
  <LastByteReceived>0</LastByteReceived>
  <IsOpen>false</IsOpen>
</SerialPortState>
```

You can compile the service in Visual Studio or do so from the command line by typing:

```
Msbuild serialport.sln
```

Compiling the service generates the service DLL, a proxy DLL, and a transform DLL (which translates data types between the service DLL and the proxy). These will all be placed into the \bin folder under the RDS mountpoint. You should also keep the manifest and config files together by copying them into the \samples\config folder (though this isn't essential).

The Service Contract

Every service has a unique contract identifier, which is declared in the Types.cs file and looks like a URL. Note, however, that it has no significance on the Web, and using a URL-style identifier is just a convenient way to guarantee uniqueness by allowing organizations to create their own namespaces. (For your projects you'll need to use a namespace other than microsoft.com/robotics.) SerialPortTypes.cs contains the following definition:

```
public sealed class Contract {
  [DataMember]
  public const string Identifier = "http://www.promrds.com/
contracts/2009/12/serialport.html";
}
```

A service contract also includes the state and operations that define what properties other services can manipulate and how. SerialPortState (see **Figure 1**) includes the serial port configuration, some parameters for timeouts, and the last byte received when operating asynchronously.

You can define your own data types for use in the state and message types. In this service, there's a SerialPortConfig class. To make it visible in the proxy, it must be public and marked with the [DataContract] attribute. Each property in the class must be declared public and also be tagged using the [DataMember] attribute. If this isn't done, the property will not be accessible.

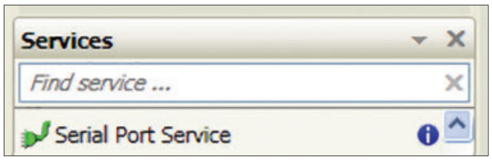


Figure 5 **VPL Service List**

You can also expose public enums—that way other programmers don't have to use magic numbers in code that uses your service. It's good programming practice, too, because it allows datatype checking.

When you design a service, you also have to decide how other services will interact with it. This SerialPort service supports the following operations, listed below in logical groups:

- Get, Subscribe
- ReceiveByte
- SetConfig, Open, Close, ClearBuffers
- ReadByte, ReadByteArray, ReadString
- WriteByte, WriteByteArray, WriteString

SerialPortService
Open
Fault

Figure 6 **Service Block**

Note that the ReceiveByte operation is for internal use by the service itself and shouldn't be used by other services. More on this later.

The Read and Write operations are all synchronous—the service does not respond until the operation has completed. If you open the serial port in asynchronous mode (explained below), the service will send you a ReceiveByte notification for every byte received and you should not use the Read operations. The Write operations, however, are always synchronous.

Each operation has a request message type and a response message type. Some of these might have properties and others will be empty—the datatype itself is sufficient to convey the appropriate message.

The RDS package doesn't include a serial port service.

Service Behavior

As noted earlier, the executable code of the service is in SerialPort.cs. All services are derived from DsspServiceBase, which provides a number of helper methods and properties.

When a service is started, its Start method is called:

```
protected override void Start() {
  InitializeState();
  base.Start();
  if (_state.OpenOnStart)
    OpenPort();
}
```

Start takes care of initializing the state (if necessary) and then opening the serial port automatically if OpenOnStart was specified in the config file (explained below).

Each operation supported by a service must have a service handler. But some operations, such as Drop and Get, are handled by the infrastructure (unless you wish to override the default behavior).

The OpenHandler shows a very simple handler:

```
[ServiceHandler]
public void OpenHandler(Open open) {
  // Remember the Asynchronous flag
  _state.Asynchronous = open.Body.Asynchronous;
  if (OpenPort()) {
    open.ResponsePort.Post(
      DefaultSubmitResponseType.Instance);
  }
  else {
    throw (new Exception("Open Failed"));
  }
}
```

This handler calls OpenPort, an internal method. If this is successful, a response is posted back. Because no information has to be returned, this is just a default response provided by DSS.

If the open fails, then an exception is thrown. DSS catches the exception and converts it to a fault, which is sent back as the response. Though not immediately obvious, if an exception occurs in OpenPort, it will also bubble up and return a fault.

There's no need to explain all of the OpenPort method, but one point is important—you can open the serial port for either synchronous or asynchronous operation. In synchronous mode, all read and write requests complete before returning a response. However, if you open in asynchronous mode, notifications are sent for every character received. To make this work, the code sets up a conventional event handler:

```
if (_state.Asynchronous) {
  // Set up an Event Handler for received characters
  sp.ReceivedBytesThreshold = 1;
  sp.DataReceived += new SerialDataReceivedEventHandler(
    DataReceivedHandler);
}
```

The event handler is shown in **Figure 2**. This handler runs on a .NET thread. But to interoperate with DSS, it must be switched to a CCR thread, so the code posts a ReceiveByte request to the service's own main operations port. After posting a request, the code should receive the response, otherwise there will be a memory leak. This is the purpose of Arbiter.Choice, which uses the C# shorthand notation for anonymous delegates to handle the two possible responses. An Activate is necessary to place the Choice receiver into the active queue. Only a fault is relevant in this case, and a successful result does nothing.

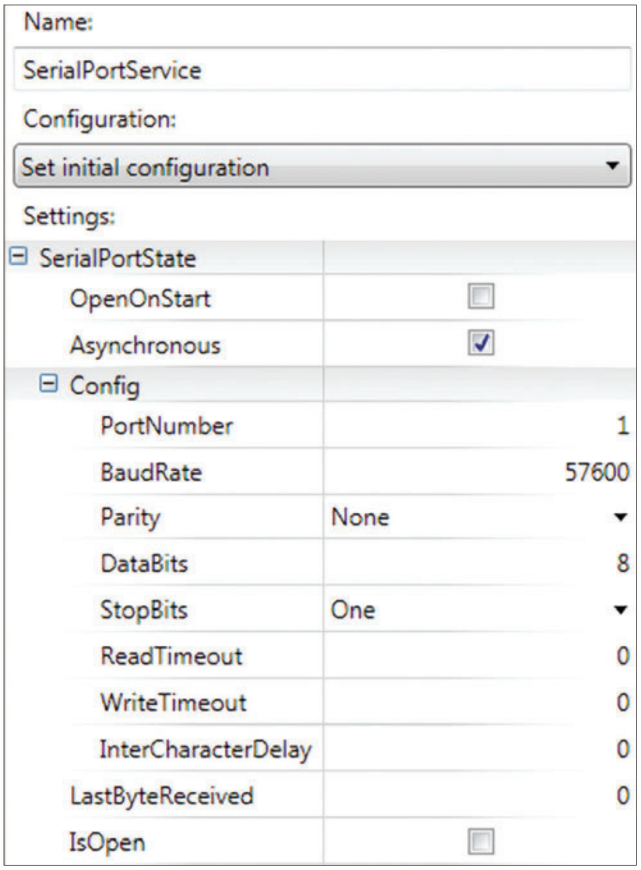
The ReceiveByte handler will be executed next to process the new character:

```
[ServiceHandler]
public void ReceiveByteHandler(ReceiveByte recv) {
  _state.LastByteReceived = recv.Body.Data;
  // Send a notification, but only if in asynch mode
  if (_state.Asynchronous)
    SendNotification(_submgrPort, recv);
  recv.ResponsePort.Post(DefaultUpdateResponseType.Instance);
}
```

The [ServiceHandler] attribute lets DSS hook the handler to the operations port during service initialization. The handler sends a notification to subscribers, then posts back a response saying the operation is complete. (Services that wish to receive notifications must send a Subscribe operation to the SerialPort service).

Service handlers for the other read and write operations are fairly self-explanatory. The WriteStringHandler (see **Figure 3**) contains one small wrinkle, though—it can insert a small delay between sending characters. This is designed to help slower microcontrollers that might not be able to keep up if the data is sent at full speed, especially devices like the BasicStamp that do bit banging and don't have a hardware Universal Asynchronous Receiver/Transmitter (UART), so they perform serial I/O in software .

Another point about this handler is that it is an Iterator. Notice the declaration of the method as IEnumerator<ITask> and the fact that it uses yield return and yield break. This feature of the C# language allows CCR to suspend tasks without blocking a thread. When


Name:
SerialPortService
Configuration:
Set initial configuration
Settings:

SerialPortState	
OpenOnStart	☐
Asynchronous	✓
Config	
PortNumber	1
BaudRate	57600
Parity	None
DataBits	8
StopBits	One
ReadTimeout	0
WriteTimeout	0
InterCharacterDelay	0
LastByteReceived	0
IsOpen	☐

Figure 7 **Serial Port Service Configuration**

the yield return is executed, the thread executing the method is returned to the pool. Once the timeout has completed, it posts back a message that causes execution to resume, though possibly on a different thread.

Configuring the Service

A configuration file is an XML serialized version of the service state that's loaded during service initialization. It's a good idea to support a config file because it allows you to change the behavior of the service without recompiling. This is especially important for setting the COM port number.

You can specify the name of the file in the service when you declare the state instance (in SerialPort.cs):

```
[ServiceState]
// Add an initial state partner to read the config file
[InitialStatePartner(Optional = true,
  ServiceUri = "SerialPort.Config.xml")]
SerialPortState _state = new SerialPortState();
```

In this case we've declared the config as optional; otherwise, the service won't start if the config file is missing. On the other hand, this means you have to check the state in the service's Start method and initialize it to some sensible defaults if necessary.

Because no path is specified for the ServiceUri, DSS will assume that the file is in the same folder as the manifest used to start the service. **Figure 4** shows the contents of a typical config file.

Note that this config file does not request the service to open the COM port automatically, so you'll have to send an Open request.

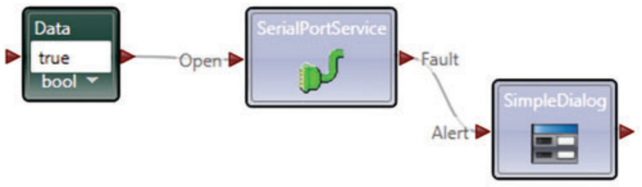


Figure 8 **Opening the Serial Port in Asynchronous Mode**

If you want your service to have a professional look, you need to pay attention to details such as the service description and icons. Even if you don't plan to use VPL yourself, it's a good idea to test your service in VPL and make it VPL-friendly so other people can use it easily.

You can decorate your service class with two attributes that describe the service:

```
[DisplayName("Serial Port Service")]
[Description("SerialPort service provides access to a Serial (COM)
Port")]
```

The first attribute displays as the name of the service in the VPL service list (see **Figure 5**), and the second attribute appears as a tooltip if you mouse over the name of the service in the list.

If you have a Web site and you want to document your services online, you can attach another attribute to the service class to provide the hyperlink.

```
[DssServiceDescription("http://www.promrds.com/SerialPort.htm")]
```

You can define your own data types for use in the state and message types.

When VPL sees this attribute, it adds a small information icon (a white "i" on a blue disc) beside the service in the service list.

If you add icons to your service, which you can easily do, they'll appear in VPL diagrams and the service list. Notice the small icon, called a thumbnail, beside the name of the service in **Figure 5**. **Figure 6** illustrates the larger version of the icon, which will show up inside the block that appears in VPL diagrams.

When you add these images to a project, make sure you change the file properties so that Build Action is set to Embedded Resource. PNG is the preferred file format because it supports an alpha channel. This lets you create an icon with a transparent background by setting the alpha value on the background color to zero.

The service icon image should be 32 by 32 pixels and the thumbnail 16 by 16. The file names of the images must begin with the class name of the service, in this case SerialPortService. Thus I made the file names for my example SerialPortService.Image.png and SerialPortService.Thumbnail.png.

Using the Service

The service is quite flexible. You can specify the serial port configuration in a config file (which is the most common way) or by sending a SetConfig request. Once you've set the config, you can call the Open operation. For convenience, a flag in the config file will

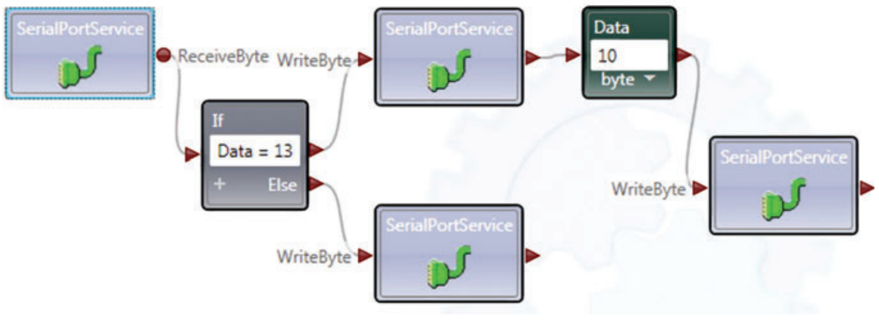


Figure 9 **EchoAsynch Program**

The Open, Close, and SetConfig operations also cause notifications to be sent.

Because the Read and Write operations do not modify the state, they are subclassed off the DSSP Submit operation. Of course, they have a side-effect, which is to receive and send data over the serial link.

Testing with VPL

The download for this article includes two sample VPL programs, EchoAsynch and EchoSynch, which show how to use the service in both asynchronous mode (via notifications) and synchronous mode. The VPL samples use config files to set the initial parameters for the COM port, including the port number (which is set to 21 in the config files and must be changed to match your computer's COM port address).

Note that to test the service you will need a null modem cable and either two computers with serial ports or two ports on the same computer. USB-to-serial devices are readily available, so it's quite feasible to have multiple serial ports on a single PC. When you have your COM ports connected together, run a terminal emulator such as HyperTerminal and connect to one of the COM ports. Start by running the EchoAsynch VPL program on the other serial port. (You can find VPL in the Start menu under RDS). When you type into the terminal emulator window, you should see the characters echoed.

You can use VPL to create config files. Click on a SerialPort service block in the diagram and look in the Properties panel. You should see something like **Figure 7**. Make sure the PortNumber is set correctly for your PC. (This must be the other serial port, not the one you opened in the

cause the service to open the port automatically on startup. If the port is already open, the Open call will first close, then re-open it.

You need to decide how you want to use the service: synchronously or asynchronously. When operating synchronously, each read or write request will wait until the operation completes before sending back a response. In terms of VPL, this is a simple approach, because the message flow will pause at the SerialPort block in the diagram. Note that asynchronous operation is not fully asynchronous—write operations still occur synchronously. But every new byte received is sent as a notification to subscribers.

In theory, every state-modifying operation on a service should cause a notification to be sent so that subscribers can keep their own cached version of the service state. This means all operations based on the DSSP Replace, Update, Insert, Delete and Upsert operations should send a corresponding notification. However, developers often play fast and loose with this requirement.

For simplicity, the SerialPort service only sends notifications using the ReceiveByte operation type when in asynchronous mode.

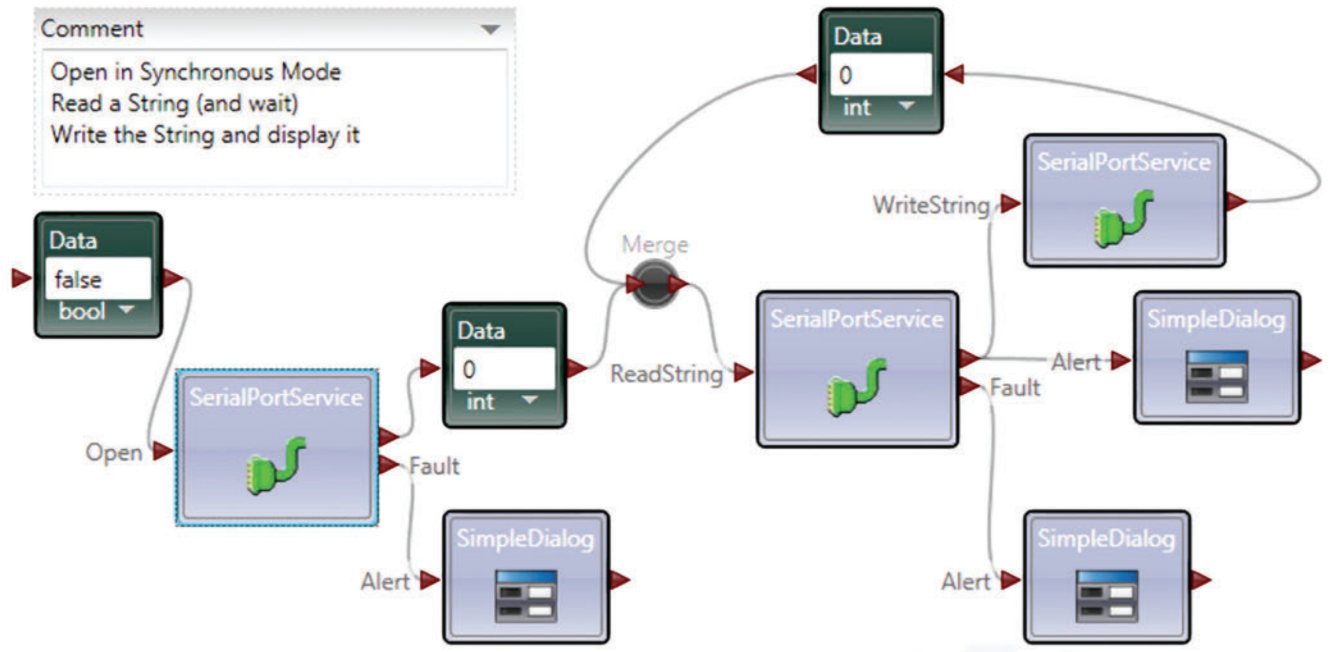


Figure 10 **The EchoSynch Program**

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

terminal emulator). You'll notice that Parity and StopBits are drop-down lists. The entries in these lists come directly from the enums defined in SerialPortTypes.cs.

This is one place where XML doc comments come in handy. When you hover over a config parameter with the mouse cursor, a tooltip will pop up showing the corresponding comment for each state member.

When you run the EchoAsynch VPL program, the first part of the program in **Figure 8** opens the serial port in asynchronous mode.

If you've entered invalid values in the config, such as a bad baud rate, the Open will fail. (It might also fail if the COM port is in use, the port doesn't exist, or you don't have appropriate permissions). This is why the program checks for a fault and displays it.

You need to decide how you want to use the service: synchronously or asynchronously.

The rest of the program (see **Figure 9**) just echoes every character received. It does this by taking the characters from ReceiveByte notifications and sending them back using WriteByte.

To make the output easier to read, a carriage return character (ASCII 13, decimal) has a linefeed (10) appended so the cursor will move to the next line in your terminal emulator window. Note that all of the SerialPortService blocks in the **Figure 9** diagram refer to the same instance of the service.

The EchoSynch VPL program (see **Figure 10**) uses the synchronous mode of operation—it doesn't use notifications. (In fact, notifications are never sent in synchronous mode).

Unlike the previous program, this one uses ReadString and WriteString to echo data. These operations perform functions similar to those of ReadByteArray and WriteByteArray, However, strings are easier to handle in VPL than byte arrays.

The string operations use a linefeed (or newline) character as an end-of-line marker. So ReadString completes when you press linefeed (Ctrl-J) not when you press Enter. This can be confusing the first time you test the program using a terminal emulator, because you might wonder why nothing is echoing. WriteString adds a carriage return and linefeed to the output so each string appears on a separate line.

Note that the config file for EchoSynch has an InterCharacter-Delay of 500ms. This causes the strings to be sent very slowly. You can try changing this value. ■

TREVOR TAYLOR, *Ph.D., is the program manager for Robotics Developer Studio at Microsoft. He's the coauthor, with Kyle Johns, of "Professional Robotics Developer Studio" (Wrox, 2008). Sample code from the book is available from promrds.com.*

THANKS to the following technical experts for reviewing this article: *Kyle Johns*

Creating Interactive Bing Maps with Silverlight and IronRuby

Ashish Ghoda

One of the notable features of Silverlight is its support for dynamic languages such as IronRuby and IronPython. This integration capability enables the development of Rich Internet Applications (RIAs) using the Silverlight platform—XAML for the presentation layer and dynamic languages for the code-behind. This article demonstrates the integration capability of Silverlight with dynamic languages and the Microsoft Bing Map controls. I'll start with a high-level overview of dynamic languages, then dive into Silverlight's support for these languages. I'll wrap up by showing you how to build an interactive 3-D animated location-finding Silverlight application using the Microsoft Bing Map Silverlight control and IronRuby.

Dynamic Language Basics

Read-Eval-Print Loop (REPL) environments provide lightweight "play as you go" programming capability for developers through the use of what are known as dynamic programming languages—languages that are dynamically typed and compiled at runtime. You do not need to declare variables of particular data types. Everything is handled by the runtime through the context of expressions.

The more familiar languages such as C# and Visual Basic are statically typed languages that are more rigid in nature. Development and deployment using dynamic languages is simpler compared to those static languages, which require compilation and distribution of output. However, you still need to do proper validation and testing for type safety when using dynamically typed languages.

With dynamic languages, you can create a function and assign it to a variable or pass it as a parameter to another function. This makes things like closures and passing functions as parameters much easier. In general, two defining characteristics of closures are the ability to assign a block of code (a function) to a variable, and this block of code's ability to retain access to variables that were accessible where it was created.

The following traditional ShortWords method in C# returns a subset of a list of words that matches the criteria of a maximum word length of 3 or fewer letters:

```
public static List<string> ShortWords(List<string> wordList) {
  List<string> shortWordList = new List<string>();
  int maximumWordLength = 3;
  foreach(string word in wordList) {
    if(word.Length <= maximumWordLength) {
      shortWordList.Add(word);
    }
  }
  return(shortWordList);
}
```

With LINQ, you can achieve similar functionality in a much more effective way, as you can see in the following code snippet:

```
public static List<string> ShortWords(List<string> wordList) {
  var maximumWordLength = 3;
  return wordList.Where(w => w.Length <=
     maximumWordLength).ToList<string>();
end
```

Implementing the same method in a dynamic language such as IronRuby—an implementation of the Ruby programming language

This article discusses:

- Dynamic languages and Silverlight
- Silverlight, IronRuby and the just-text approach
- Bing Maps integration
- Custom views and 3-D animation

Technologies discussed:

Silverlight, IronRuby, IronPython, Bing Maps

Code download available at:

code.msdn.microsoft.com/mag201002Bing

for the Microsoft .NET Framework—is similar to the C# using LINQ approach and is significantly shorter than the traditional approach:

```
def ShortWords(wordList)
  maximumWordLength = 3
  return wordList.select {|w| w.Length <= maximumWordLength}
end
```

Just comparing these two implementations of the same algorithm reveals much about IronRuby (and dynamic languages in general). The IronRuby code is concise, and nowhere do you see a data type keyword such as string or int.

IronRuby code is concise, and nowhere do you see a data type keyword such as string or int.

The most interesting aspect of this block of IronRuby code is the closure, located between the curly braces. Here the closure, essentially a function, is being passed to the select method. The select method uses a closure to extract a subset of a collection. The code that forms the closure actually executes within the select method (here, the closure extracts strings within the collection wordList that meet the criterion), but it retains access to the variables in its original scope (in this case, the maximumWordLength variable).

Closures are much more powerful than this simple example illustrates. They are similar to using LINQ or passing a delegate to a method such as Exists or Find in C#, with the additional benefit of retaining access to their original scope. You can get more details on closures from the book I wrote with Jeff Scanlon, "Accelerated Silverlight 3" (Apress, July 2009).

Dynamic Languages for Silverlight

Silverlight currently supports the IronRuby and IronPython dynamic languages via the Microsoft Dynamic Language Runtime (DLR) engine—a generic platform and hosting model for dynamic languages to run on top of the Microsoft .NET Framework Common Language Runtime (CLR).

The DLR is a set of .NET Framework libraries and services that dynamically discover types at runtime using reflection, so that code written in dynamic languages can be executed on the .NET platform.

There are five DLR scripting assemblies that provide the runtime scripting environment—bridging the dynamic languages with Silverlight:

- Microsoft.Scripting.dll
- Microsoft.Scripting.Core.dll
- Microsoft.Scripting.Silverlight.dll
- Microsoft.Scripting.ExtensionAttribute.dll
- Microsoft.Scripting.Debugging.dll

Microsoft.Scripting.Silverlight.dll contains classes that let developers write Silverlight applications using dynamic languages. One of the key classes is DynamicApplication, which inherits directly from System.Windows.Application. This class represents the Silverlight-based dynamic application object by providing access to visual elements from the dynamic language code, as well as an entry point for dynamic language applications to host on Silverlight hosts. It provides additional properties that extend the Host, Resources, and RootVisual properties already supplied by the Application class.

IronRuby (ironruby.net) is an open source implementation of the Ruby programming language that provides integration between Ruby and the .NET Framework.

The current version of IronRuby (1.0-rc1) supports the .NET Framework 3.5 and the .NET Framework 4 beta. Note that IronRuby 1.0-rc1 provides both .zip and .msi downloads. For Silverlight applications, it's better to use the .zip version.

IronPython (ironpython.codeplex.com) is an open source implementation of the Python programming language that, like IronRuby, allows integration of the Python language with the .NET Framework. Currently you can download IronPython 2.6 for the .NET Framework 3.5 and the .NET Framework 4 beta.

Figure 1 **Core Files for Dynamic Language-Based Silverlight Applications**

IronRuby	IronPython Installed with IronRuby	IronPython Installed Independently	Description
index.html	index.html	index.html	Hosts the dynamic language-based Silverlight application.
app\app.rb	app\app.py	python\app.py	Main startup file for the Silverlight application.
app\app.xaml	app\app.xaml	python\app.xaml	Main XAML user interface file.
css\screen.css	css\screen.css	stylesheets\screen.cs	Defines application styles.
Not provided	Not provided	stylesheets\error.css	Defines application error styles and format.
js\error.js	js\error.js	javascripts\error.js	Manages unhandled application errors.

IronRuby and IronPython each have two assemblies that support the specific language, providing capabilities such as parsing the language and communicating with the host environment. They are IronPython.dll and IronPython.Modules.dll for IronPython, and IronRuby.dll and IronRuby.Libraries.dll for IronRuby.

Note that both IronRuby and IronPython are in continuous development. Visit their homepages to access the latest releases and documentation. You can also get the related source code, along with the source code of the DLR, by visiting dlr.codeplex.com.

Installing Development Components

There are two approaches to developing dynamic language-based Silverlight applications.

- The traditional approach using the Chrion.exe development utility
- The just-text approach using inline browser scripting

In this article, I will provide an overview of both approaches and then will develop a sample Microsoft Bing Maps application using the newer just-text approach.

Since the introduction of Silverlight 2, along with the DLR scripting libraries, Microsoft has provided a dynamic language-based Silverlight application development environment via the Chiron.exe development utility and IronRuby and IronPython Silverlight application project templates.

To get started, download and install the DLR and either IronRuby or IronPython from the sites mentioned earlier. Samples, documentation, utilities and some additional important components are installed along with IronRuby and IronPython.

The Silverlight templates for dynamic languages provide core application files, which are available under the Silverlight\script\templates\ruby and Silverlight\script\templates\python folders. **Figure 1** shows some details about these application template files.

Figure 2 **Core Library Files for the Just-Text Approach**

IronRuby File	Description
dlr\dlr.js	Enhanced Silverlight.js file to host the dynamic language-based Silverlight application and enable inline scripting on HTML pages.
dlr\ gestaltmedia.js	Enables HTML5 video and audio playback.
dlr\dlr.xap	Includes the AppManifest.xaml file that references Microsoft.Scripting.slvx and points to Microsoft.Scripting.Silverlight.dll as an entry point assembly. Also includes languages.config to provide configuration. information for DLR languages.
dlr\IronRuby.slvx	Includes the IronRuby.dll and IronRuby.Libraries.dll files to enable development of IronRuby-based Silverlight applications.
dlr\IronPython.slvx	Includes the IronPython.dll and IronPython.Modules.dll files to enable development of IronPython-based Silverlight applications.
dlr\ Microsoft.Scripting.slvx	Includes five DLR scripting assemblies (Microsoft.Scripting.dll, Microsoft.Scripting.Core.dll, Microsoft.Scripting.Silverlight.dll, Microsoft.Scripting.ExtensionAttribute.dll and Microsoft.Scripting.Debugging.dll) that provide the runtime scripting environment, bridging the dynamic languages with Silverlight.
samples/getting.started/*.html	Sample Web pages demonstrating inline scripting, IronRuby, IronPython and XAML capabilities.

The Script folder includes the sl.bat file, which will help you to create a preliminary dynamic language-based Silverlight application. The following shows the command-line format:

```
sl [ruby|python] <ApplicationPath>
```

Chiron.exe, the Silverlight development utility, dynamically packages a set of files into an .xap file for deployment. (For further discussion of Chiron.exe and compilation, see blog.jimmy.schementi.com/2009/03/state-of-dlr-for-silverlight.html.)

You can launch an application using Chiron.exe with the /b (browser) option:

```
Chiron /b
```

One of the interesting features of Chiron.exe is that any time you modify a file within the application directory, Chiron.exe will repackage the application into an .xap and reload it. You must still refresh any active browser sessions, though.

The Just-Text Approach

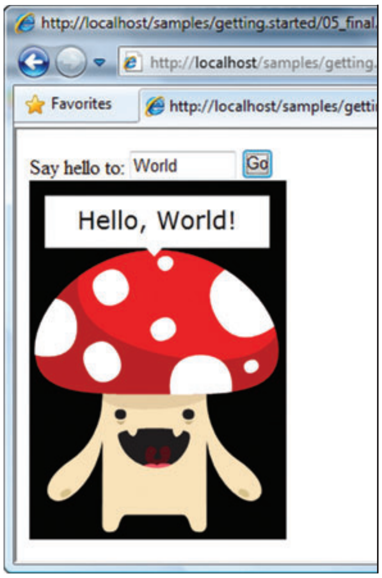
The traditional DLR-based development approach makes using the Chrion.exe utility mandatory and follows the old edit-compile-refresh development model.

It is now possible to write IronRuby, IronPython and XAML code within (X)HTML markup directly in the browser (see ironruby.com/browser for details). This is called the just-text approach. No need to install any components to create and run the DLR-based application. The just-text approach follows the write-save-refresh development model and removes the need for Chrion.exe.

With the just-text approach, you don't even need local copies of the DLR scripting assemblies, along with the IronRuby and IronPython language-specific assemblies mentioned earlier. Though the Gestalt sample package available from ironruby.com/browser contains the binaries, you can also reference the dlr.js from a well-known server, and that requires that you have nothing installed. However, you do need a way to host Silverlight controls and enable DLR integration within the HTML page.

The Gestalt project capitalizes on the existing Silverlight.js approach, which uses the JavaScript API to create the Object tag that hosts the Silverlight control. It also enables error management and detecting the browser and Silverlight plug-in requirements on the client machine. The Mix Online Lab team enhanced the Silvelright.js file to include inline scripting and DLR integration capabilities and renamed the file as dlr.js.

To get started, the Gestalt project provides the cross-browser, cross-platform library built on the DLR. You can get the compressed library file, gestalt.zip, from visitmix.com/labs/gestalt/downloads. **Figure 2** provides details on the core files included in the .zip file.



http://localhost/samples/getting.started/05_final
http://localhost/samples/getting
Favorites
http://localhost/samples/getti
Say hello to: World
Go
Hello, World!

Figure 3 **Running the Gestalt Project's Sample Application**

Note that from Silverlight 3, the Transparent Silverlight Extensions capability enables developers to package the commonly used assembly files as a separate reusable library with an .slvx filename extension. The .slvx files can be deployed on a common Internet location or client-specific location. The required .slvx files must be referenced in the AppManifest.xaml file within the ExternalParts section as an ExtensionPart with the correct path.

Jimmy Schementi's excellent paper on the just-text approach (ironruby.com/browser/sl-back-to-just-text.pdf) provides some very helpful guidance. This paper also details how to change the default DLR settings of the dlr.js file.

You need a Web server instance such as IIS or Apache to host and run the DLR-based

Figure 4 **Modified AppManifest.xaml File**

```
<Deployment
  xmlns="http://schemas.microsoft.com/client/2007/deployment"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  RuntimeVersion="2.0.31005.0"
  EntryPointAssembly="Microsoft.Scripting.Silverlight"
  EntryPointType="Microsoft.Scripting.Silverlight.DynamicApplication"
  ExternalCallersFromCrossDomain="ScriptableOnly">
  <Deployment.Parts>
    <AssemblyPart Source="Microsoft.Maps.MapControl.dll" />
    <AssemblyPart Source="Microsoft.Maps.MapControl.Common.dll" />
  </Deployment.Parts>
  <Deployment.ExternalParts>
    <ExtensionPart Source="Microsoft.Scripting.slvx" />
  </Deployment.ExternalParts>
</Deployment>
```

Figure 5 **Referencing Map Controls in the HTML File**

```
<script type="application/xml+xaml" id="sl_map"
  Width="1350" Height="575">
  <UserControl x:Name="silverlight_map"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Width="200"
    Height="280" Background="Black"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
    mc:Ignorable="d"
    xmlns:m="clr-namespace:Microsoft.Maps.MapControl;assembly=Microsoft.
Maps.MapControl">
  </UserControl>
</script>
```

inline scripted Web applications. From gestalt.zip, place the Dlr and Samplesfolders in the root of the Web server. If you do not install these folders at the root of the Web server, you need to modify the dlr.js file appropriately.

Next add MIME types for .rb, .py and .slvx files, like the following:

- For .rb and .py files set the MIME-type to: text/plain
- For .slvx files set the MIME-type to: application/octet-stream

To validate the environment, visit the samples/get.started folder and browse the 05_final.html file. The Web page demonstrates the IronPython, HTML and XAML-based graphics with animation-integration capabilities, as shown in **Figure 3**.

Silverlight, IronRuby and the Just-Text Approach

Let's start by defining the skeleton of a DLR-based Silverlight application using IronRuby, following the just-text approach. Once you copy the Gestalt files to the Web server root, simply open a text editor and start writing your HTML file. It's that simple!

The good thing is, dlr.js adds a Silverlight control to the page and provides all the necessary core requirements to enable dynamic language integration capabilities. For this, simply include the dlr.js file on the HTML page:

```
<head>
  <script src="/dlr/dlr.js" type="text/javascript"></script>
</head>
```

Note that including dlr.js will configure default settings for your DLR-based Silverlight application. If you would like to customize the settings, you need to override the defaults by writing custom script code within the HTML file. For details, see Jimmy Schementi's paper mentioned earlier.

Now you are all set to write XAML and IronRuby or IronPython code in the HTML file, within a script tag. To write inline IronRuby code, you need to add the script tag with the appropriate type and class information and place the IronRuby code within the tag, like so:

```
<script type="application/ruby"
  class="Class Name Goes Here">
  IronRuby Code Goes Here
</script>
```

To write inline IronPython code, simply do the same with the appropriate substitutions:

```
<script type="application/python"
  class="Class Name Goes Here">
  IronPython Code Goes Here
</script>
```

To write inline XAML code, add the script tag with the appropriate type, ID, width, and height information and place the XAML code within the tag:

```
<script type="application/xml+xaml" id="Place ID here"
  Width="400" Height="400">
  <UserControl ...>
    XAML Code Goes Here
  </UserControl>
</script>
```

You can access the XAML controls and implement the event integration using code, which I will demonstrate while developing the application in the next section. Once you finish with the code, just browse the page and you should see the application outcome right away.

Bing Map Integration

Now that you've seen the basic skeleton of the dynamic language Silverlight application using the inline scripting just-text approach, let's take it a step further with the integration of Microsoft Bing maps (formerly known as Virtual Earth).

The Microsoft Bing Maps Silverlight Control SDK version 1 was released in November 2009 (msdn.microsoft.com/library/ee681884). The installer is called BingMapsSilverlightControlv1.0.1Installer.msi. Note that you need at least Silverlight 3 to work with this control. The installation includes Microsoft.Maps.MapControl.dll and Microsoft.Maps.MapControl.xml, Microsoft.Maps.MapControl.Common.dll and Microsoft.Maps.MapControl.Common.xml and offline documentation.

Before you start building applications using the Silverlight Bing Maps control, you must create a Bing Maps Developer account to receive the application authentication key. To do so, visit bingmapsportal.com.

The Microsoft Bing Maps Silverlight control CTP release was available before the release of Version 1 of the SDK. There are considerable changes and enhancements in version 1 compared to the CTP release. See msdn.microsoft.com/library/ee681889 to understand the key differences between the CTP and version 1 releases.

Now create a SilverlightMap.html file and include the dlr.js file as described in the previous section.

You need to modify the AppManifest.xaml file (available in the dlr.xap file) and include Microsoft.Maps.MapControl.dll and Microsoft.Maps.MapControl.Common.dll files to load as part of the application start up. To do this, rename dlr.xap to dlr.xap.zip and extract the AppManifest.xaml and languages.config files from the file. Then add Microsoft.Maps.MapControl.dll and Microsoft.Maps.MapControl.Common.dll files as AssemblyPart, as shown in **Figure 4**.

Now zip up the modified AppManifest.xaml, existing langugages.config, Microsoft.Maps.MapControl.dll and Microsoft.Maps.MapControl.Common.dll files and rename the .zip file to dlr.xap. Overwrite the existing dlr.xap file (under the Dlr folder) on the Web server with the new one.

Next, open the SilverlightMap.html file, add the script tag for the XAML code, then add the UserControl with the name Silverlight_map and a reference to the map control to create the necessary namespace (see **Figure 5**).

Finally, add the Canvas control as the main container and the Map element under the Grid control. Notice that I kept the same width and height of Canvas and Grid controls and set the Width and Height properties of the Map control to 800 and 400, respectively:

```
<Canvas x:Name="container" Width="1350" Height="575">
<Grid x:Name="layout_root" Width="1350" Height="575">
<m:Map CredentialsProvider="AuthenticationKey"
  Width="800" Height="400" Grid.Column="1"
  HorizontalAlignment="Center"/>
</Grid>
</Canvas>
</UserControl>
```

In the code snippets shown here, you need to replace "AuthenticationKey" with your authentication key for the Map control.

Copy the file to the existing Sample/Getting.started folder on the Web server and browse the page. You should see the Map with the default Road mode (see **Figure 6**).

Map Modes and 3-D Animation

Let's change the map mode to aerial with labels as the default view. Let's also introduce 3-D animation to the map.

To change the default map mode, first give the Map element a name (in this example, I used map_in_ironruby) so it can be referenced in the IronRuby-based inline code. I'll also apply 3-D projection to the Map object. This was a new feature in Silverlight 3. To do that I set the Projection property of the Map object to PlanProjection and I set the RotationX property to -20. This transforms the Map object, giving a slightly skewed viewing angle:

```
<Grid x:Name="layout_root" Width="1350"
Height="575" Background="Black">
  <m:Map x:Name="map_in_ironruby"
  CredentialsProvider="AuthenticationKey"
  Width="800" Height="400">
    <m:Map.Projection>
      <PlaneProjection RotationX="-20"/>
    </m:Map.Projection>
  </m:Map>
</Grid>
```

Notice that I also changed the background of the Grid to black.

Now add the script tag for IronRuby and write the following lines of code to include the required assemblies (including MapControl.dlls) and turn on aerial view with labels for the Map control.

Notice that with the new just-text approach, I can reference the Map object by name. In the current version of Gestalt libraries, you need to reference objects with me or xaml shorthand (here I used the me shorthand). In the future, XAML elements with the x:Name set can be accessed through root_visual shorthand:

```
<script type="application/ruby" class="sl_map">
  require "Microsoft.Maps.MapControl.dll"
  require "Microsoft.Maps.MapControl.Common.dll"
  include System::Windows
  include System::Windows::Controls
  include Microsoft::Maps::MapControl
  sm = me.silverlight_map
  sm.map_in_ironruby.mode = AerialMode.new(true)
</script>
```

Note that I kept the class name of the IronRuby-related script tag as sl_map, which is similar to the ID of the XAML-related script tag.

Now if you run the application, you'll see the black background, skewed 3-D angle and labeled aerial view, as shown in **Figure 7**.

One popular demonstration at the MIX09 conference was Silverlight and Microsoft Bing Map integration with spinning capabilities for the Map object. Let's implement something similar in IronRuby.

To implement this feature, you first need to define the Grid with two columns using ColumnDefinitions:

```
<Grid.ColumnDefinitions>
  <ColumnDefinition Width="200"/>
  <ColumnDefinition Width="1100"/>
</Grid.ColumnDefinitions>
```

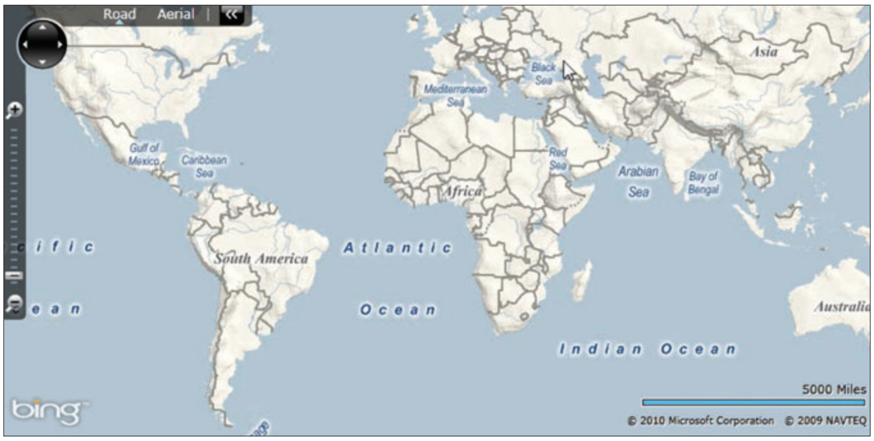


Figure 6 **DLR-Based Bing Map Silverlight Application in Default Road Map Mode**

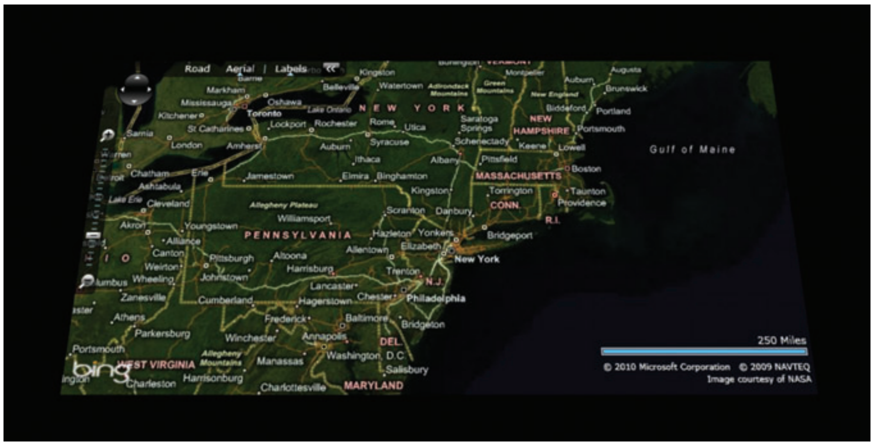


Figure 7 **Map Mode Set to Aerial with Labels Mode and with 3-D Projection**

Figure 8 **Adding Controls to the Grid Object**

```
<StackPanel Grid.Column="0" Orientation="Vertical">
  <Border CornerRadius="20" Margin="0,50,0,5" Width="150"
    Background="DarkBlue" HorizontalAlignment="Center">
    <StackPanel Orientation="Vertical">
      <TextBlock Text="3D Rotation"
        HorizontalAlignment="Center"
        FontSize="12" Foreground="White" Margin="0,5,0,10"/>
      <Button x:Name="RotateMap" Height="25"
        Content="rotate_map" Width="100" Margin="0,0,0,10"
        Foreground="Black" VerticalAlignment="Center"
        HorizontalAlignment="Center" />
      <Button x:Name="pause_resume" Height="25"
        Content="Pause" Background="DarkGoldenrod"
        Foreground="Black" Width="100" Margin="0,0,0,10"
        VerticalAlignment="Center"
        HorizontalAlignment="Center" />
      <Button x:Name="stop_reset" Height="25"
        Content="Stop and Reset" Background="DarkGoldenrod"
        Foreground="Black" Width="100" Margin="0,0,0,10"
        VerticalAlignment="Center"
        HorizontalAlignment="Center" />
    </StackPanel>
  </Border>
</StackPanel>
```

Next, add three buttons named Rotate Map, Pause, and Stop and Reset, along with title text in the XAML file within the Border. All of this is in the first column on the Grid (see **Figure 8**).

Now add the Map object to the second column of the Grid:

```
<m:Map x:Name="map_in_ironruby"
  CredentialsProvider="AuthenticationKey"
  Width="800" Height="400" Grid.Column="1"
  HorizontalAlignment="Center">
  <m:Map.Projection>
    <PlaneProjection RotationX="-20" RotationY="0"
      RotationZ="0"/>
  </m:Map.Projection>
</m:Map>
```

In the next step, you'll need to create some rather complex XAML update code. I recommend creating the XAML file using a development environment such as Visual Studio or Expression Blend to take advantage of their editing and IntelliSense features. Then you can copy the completed XAML to the app.xaml file of your project.

Create a Storyboard with the name map_animation targeted for the Map object named map_in_ironruby. (An excerpt is shown in **Figure 9**. See the code download for this article for the entire Storyboard code block.). The Storyboard defines the keyframes for the PlaneProjection animation properties RotationZ, RotationY, GlobalOffsetX and GlobalOffsetZ. Add the StoryBoard as a UserControl resource.

The next step is to add the Click events to the inline IronRuby code to start, pause, resume and stop the animation. First you need to add a reference to System.Windows.Media and System.Windows.Media.Animation:

```
include System::Windows::Media
include System::Windows::Media::Animation
```

Disable the pause_resume button during the initialization process of the application:

```
sm.pause_resume.is_enabled = false
```

Now implement the Click event of each button. Start with the rotate_map button. First set the Map object to left horizontal alignment in order to best use the available space for the animation. Then set the RepeatBehavior property of the StoryBoard animation to

Figure 9 **Creating the Storyboard for Animation**

```
<UserControl.Resources>
  <Storyboard x:Name="map_animation">
    <DoubleAnimationUsingKeyFrames BeginTime="00:00:00"
      Storyboard.TargetName="map_in_ironruby"
      Storyboard.TargetProperty=
        "(UIElement.Projection).(PlaneProjection.RotationZ)">
      <EasingDoubleKeyFrame KeyTime="00:00:00" Value="15"/>
      <EasingDoubleKeyFrame KeyTime="00:00:01" Value="0"/>
      <EasingDoubleKeyFrame KeyTime="00:00:02" Value="-15"/>
      <EasingDoubleKeyFrame KeyTime="00:00:03" Value="0"/>
      <EasingDoubleKeyFrame KeyTime="00:00:04" Value="15"/>
    </DoubleAnimationUsingKeyFrames>
    ...
  </Storyboard>
</UserControl.Resources>
```

Forever and begin the animation. Finally, enable the pause_resume button and set the button content to Pause:

```
sm.rotate_map.click do |s,e|
  sm.map_in_ironruby.horizontal_alignment = HorizontalAlignment.Left
  sm.map_animation.repeat_behavior = RepeatBehavior.Forever
  sm.map_animation.begin
  sm.pause_resume.is_enabled = true
  sm.pause_resume.content = "Pause"
end
```

Next, implement the pause_resume button Click event. Here, depending on whether you're in a paused state or a running state, you want to either resume or pause the Storyboard animation and change the button content:

```
sm.pause_resume.click do |s,e|
  strbtnContent = sm.pause_resume.content.ToString
  if strbtnContent == "Pause"
    sm.pause_resume.content = "Resume"
    sm.map_animation.pause
  else
    sm.pause_resume.content = "Pause"
    sm.map_animation.resume
  end
end
```

Finally, implement the stop_reset button Click event. Here you stop the Storyboard animation, disable the pause_resume button and reset the button content to Pause. You also reset the alignment of the Map object:

```
sm.stop_reset.click do |s,e|
  sm.map_animation.stop
  sm.pause_resume.is_enabled = false
  sm.pause_resume.content = "Pause"
  sm.map_in_ironruby.horizontal_alignment = HorizontalAlignment.Center
end
```

Compile and run the project using Chiron /b command to see the map with animation. **Figure 10** shows the rotating map.

Targeting Predefined Locations

Now let's highlight three predefined locations on the map: New York, San Francisco and Vancouver. This is demonstrated in C# as part of the documentation of the Microsoft Bing Map Control for Silverlight CTP. I will show you how to implement this feature using IronRuby.

First you need to update the inline XAML code to add three additional buttons in a new section on the left side of the screen, one for each location—New York, San Francisco and Vancouver. These are implemented much like the previous buttons. One notable change is the addition of the Tag attribute to each Button element. The Tag attribute defines specific location coordinates and the zoom level of the map.

The following code snippet shows the XAML code for adding a Button for the New York location:

```
<Button x:Name="newyork" Height="25"
Width="100"
  Content="New York" Margin="0,0,0,10"
Foreground="Black"
  VerticalAlignment="Center"
HorizontalAlignment="Center"
  Tag="40.7199,-74.0030,0.0000 12.0000"/>
```

This attribute provides the coordinate information for each location. When the user clicks the button, this information is used to retarget the map. See the code download for the entire location-finder buttons code.

Most applications have a title, and this should be no different. I added the title "Microsoft Bing Maps Silverlight Control and IronRuby Integration" in the second column of the Grid by replacing the existing Map element to place it under the StackPanel along with the title TextBlock control:

```
<StackPanel Grid.Column="1" Orientation="Vertical">
  <TextBlock VerticalAlignment="Top"
    HorizontalAlignment="Center" FontSize="20"
    Foreground="Red" Margin="0,5,0,0"
    Text="Microsoft Bing Maps Silverlight Control and IronRuby
Integration" />
  <m:Map x:Name="map_in_ironruby" Width="800" Height="400"
    HorizontalAlignment="Center" Margin="0,50,0,20">
...
```

Now the presentation layer is complete. If you execute the application at this point, you should see the additional three buttons under the new Locate Location section. However, the map will not be moved to the corresponding location if you click on any of the newly added buttons. For that you need to implement code behind for each button Click event.

The Click events are the same for all three buttons. Based on the value of the Tag property of the corresponding clicked button, pass those coordinates and zoom level as the view specification to create the new map view. Here I used the Split method to split the coordinates and the zoom level and set the map view using the SetView method of the Maps control. The new map view will show the defined location:

```
sm.newyork.click   do |s,e|
  tag_information = s.Tag.split
  location_Converter = LocationConverter.new
  location_info = location_Converter.ConvertFrom(tag_information[0].
ToString)
  sm.map_in_ironruby.SetView(location_info, tag_information[1]);
end
```

You also need to add the reference Microsoft.Maps.MapControl.Design to the program to create a new map view.

```
include Microsoft::Maps::MapControl::Design
```

And that's the finished application. As you can see, it would be easy to customize the views, add other location targets and implement additional features.

Moving Forward

Before finishing the article, I would like to quickly introduce the externalizing inline script code (XAML and IronRuby/IronPython) approach.

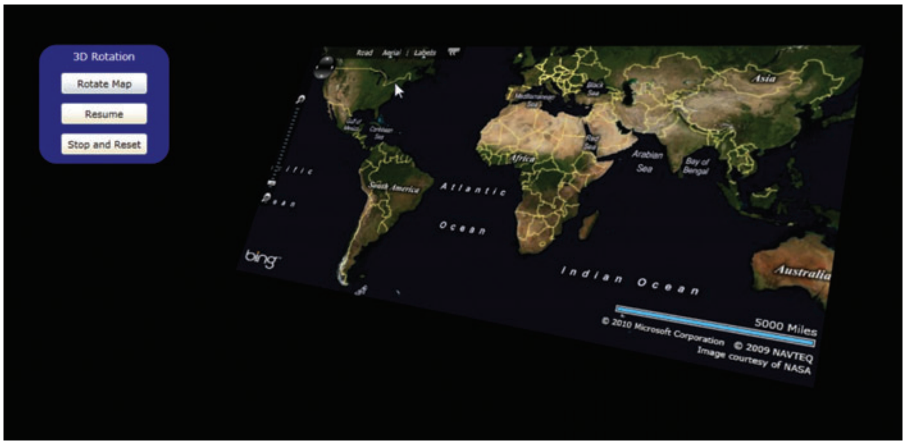


Figure 10 **3-D Map Animation**

To modularize your programming model, first copy the final SilverlightMap.html file and rename it to SilverlightMap-ExternalScript.html. Then cut and paste the inline XAML code from the SilverlightMap-ExternalScript.html file to a new blank text file and save as SilverlightMap.xaml file. Next, cut and paste the IronRuby code from the SilverlightMap-ExternalScript.html file to the new blank text file and save as SilverlightMap.rb file.

The Silverlight templates for dynamic languages provide core application files.

Now, update the XAML and IronRuby script tags of the SilverlightMap-ExternalScript.html file with the src attribute defining path of the external XAML and IronRuby files:

```
<html><head>
  <script src="/dlr/dlr.js" type="text/javascript"></script>
</head>
<body>
  <script type="application/xml+xaml"
    src="/samples/getting.started/SilverlightMap.xaml"
    id="sl_map" Width="1350" Height="575">
  </script>
  <script type="application/ruby"
    src="/samples/getting.started/SilverlightMap.rb"
    class="sl_map">
  </script>
</body> </html>
```

Finally, copy three new files—SilverlightMap-ExternalScript.html, SilverlightMap.xaml and SilverlightMap.rb—to the Sample/Getting.started Web server folder. Now if you browse the SilverlightMap-ExternalScript.html file, you will get the same rotating map with location-finding capabilities.

Please visit SilverlightStuff.net to read my article on the same application created using the traditional approach (using Chiron.exe). ■

ASHISH GHODA *is founder and president of Technology Opinion LLC, and associate director at a Big Four accounting firm. Visit his sites technologyopinion.com and SilverlightStuff.net, or contact Ghoda directly at askashish@technologyopinion.com.*

THANKS to the following technical experts for reviewing this article: *Laurence Maroney and Jimmy Schementi*

VSLive!

Empowering Developers Since 1993

THE EVENT THAT'S BEEN EMPOWERING DEVELOPERS SINCE 1993...

JOIN YOUR FELLOW DEVELOPERS FOR TOPICS ON:

- **Silverlight/WPF**
  Silverlight, WPF, XAML, Visual Studio 2010's XAML Designer, Expression Blend, WCF RIA Services
- **Web**
  Web Forms, ASP.NET MVC, ASP.NET AJAX and jQuery
- **Visual Studio 2010/.NET 4**
  Visual Studio features, TFS, new language features, parallel programming extensions
- **SharePoint**
  SharePoint 2010, Office 2010, Visual Studio Tools for Office
- **Cloud Computing**
  Includes cloud, server and messaging technologies,Windows Azure, AppFabric (for Windows Server and Windows Azure), REST services programming, Project "Dallas", WCF, Windows Workflow
- **Data Management**
  SQL Server, Microsoft BI, Entity Framework, ADO.NET Data Services, oData, Sync Services

1105 ENTERPRISE COMPUTINGGROUP Visual Studio MAGAZINE ENTERPRISE SOLUTIONS FOR .NET DEVELOPMENT

AUGUST 2-6, 2010

REDMOND, WA
MICROSOFT CAMPUS

...ARRIVES AT THE MICROSOFT CAMPUS IN REDMOND, WA!

DETAILS AND REGISTRATION AT

VSLIVE.COM

USE PRIORITY CODE VSLMS1

Take your code to the next level at VSLive! at the Microsoft Campus in Redmond, WA.

Our goal at VSLive! is to first help you learn what you need to know about the Microsoft Development Platform, and then burrow deep into the subjects you need to master.

Windows Azure Platform for Enterprises

Hanu Kommalapati

Cloud computing has already proven worthy of attention from established enterprises and start-ups alike. Most businesses are looking at cloud computing with more than just idle curiosity. As of this writing, IT market research suggests that most enterprise IT managers have enough resources to adopt cloud computing in combination with on-premises IT capabilities.

Of course, there are people who are skeptical of cloud computing's ability to deliver on the promises. This emerging solution is almost analogous to the creation of ARPANET (precursor to Internet); many skeptical research institutions didn't want to join the initial network for fear of losing their private data. Once scientists saw the benefits of data networking and the collaboration it enabled, there was no stopping them, and the rest is history. Today's large enterprises, like the ARPANET skeptics, are in the process of getting acquainted with the paradigm shift that is occurring in how computing capabilities are acquired and operated.

Sensing industry trends and customer demand, Microsoft made a huge bet on cloud computing by releasing the Windows Azure platform and the necessary supporting services for building and running industrial-strength services in the cloud. In this article, I will discuss the Windows Azure platform at the architectural level and intersect it with the needs of enterprise-class solutions.

This article discusses:

- Windows Azure platform integration in the enterprise
- Software as a Service
- Platform as a Service
- Infrastructure as a Service
- Pricing of Windows Azure platform services
- Cost calculator for Windows Azure platform services
- Security in the Windows Azure platform
- Storage in the Windows Azure platform

Technologies discussed:

Windows Azure platform, ASP.NET, Software as a Service, Platform as a Service, Infrastructure as a Service

Code download available at:

code.msdn.microsoft.com/mag201002Azure

Cloud Computing

I am sure there are several definitions of cloud computing, but the one I like the most is: computing capability delivered as a utility through Internet standards and protocols. This definition opens up the possibilities for "public cloud" and "private cloud" concepts. Public clouds, as the name indicates, are available for anyone who wields a credit card. Private clouds are meant for the exclusive use of a business or a consortium of businesses as identified by the private cloud's mission statement.

The Windows Azure platform, Amazon Web Services, Google App Engine and Force.com are a few examples of public clouds. Any private datacenter run by a large enterprise can be called a private cloud if it takes advantage of the unified resource model enabled by broader virtualization that treats compute, storage and networking as a homogenous resource pool and takes advantage of highly automated processes for operating the system.

Utility computing has been a dream of visionaries in the computer automation space for as long as I can remember. Microsoft's Dynamic Systems Initiative (DSI) and similar initiatives from other vendors made bold attempts to help datacenter operators provide utility-like characteristics: highly automated, self-managed, self-optimizing and metered storage, networking and compute cycles. While the vision was laudable, it saw mixed success. The advent of virtualization made utility computing a reality. Virtualization helped decouple the operating system and applications from physical hardware. It treats them as data, so automated processes can be developed for on-demand streaming of operating system and other dependent resources to the target hardware.

To set the stage for a Windows Azure platform discussion, I will briefly look at the industry terminology in the cloud computing space and map the Windows Azure platform to those terms so we can readily comprehend it. **Figure 1** shows the sandwich diagram of the industry terminology and the mapping of the Windows Azure platform. I will look at various cloud service types and their relative differences in detail in the following sections.

Software as a Service

Software as a Service (SaaS) is a software delivery business model in which a provider or third party hosts an application and makes it available to customers on a subscription basis. SaaS customers use the software running on the provider's infrastructure on a pay-as-you-go basis. There are no upfront commitments, so the customer is spared any long-term contracts.

Based on the contractual terms, customers may elect to quit using the software at any time. The underlying infrastructure and the software configuration are invisible to the users, and, hence, customers have to settle for the functionality that is provided out of the box. SaaS uses a highly multi-tenant architecture, and user contexts are separated from one another logically at both runtime and rest.

This multi-tenancy may be objectionable to some companies due to the nature of their business, so providers may offer a physically isolated infrastructure for such customers and charge them for the extra costs associated with maintenance of the software and the hardware. Microsoft Business Productivity Online Suite (BPOS) and CRM Online are good examples of SaaS. Microsoft also offers dedicated hosting for these services for extra charge.

Collaboration applications that solve the same problem across many enterprises have been very successful in the SaaS space. Because the hardware and software configuration is transparent to end users, there is minimal if any need for IT pro involvement. Some SaaS applications can be customized by end users through configuration; however, most do not allow customization. As a result, the footprint of the development staff in the context of the SaaS application is also minimized.

SaaS can improve the time-to-market aspect of applications, and in the process fix the often-complained-about business-IT alignment problems. During early stages of the SaaS adoption in the enterprise, to the nightmare of enterprise architects, "shadow IT" (a small team of spreadsheet-savvy programmers attached to business units, for example) may distract from the enterprise-wide initiatives. This is because SaaS empowers business units to bypass IT procurement processes. Enterprise architecture teams need to

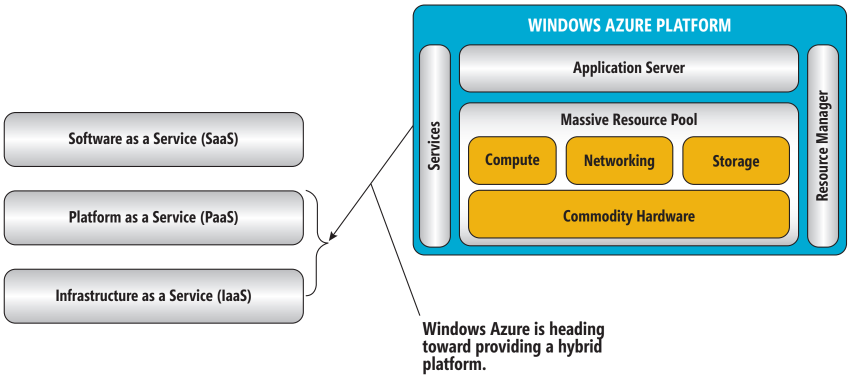

WINDOWS AZURE PLATFORM
Application Server
Massive Resource Pool
Compute
Networking
Storage
Commodity Hardware
Services
Resource Manager
Software as a Service (SaaS)
Platform as a Service (PaaS)
Infrastructure as a Service (IaaS)
Windows Azure is heading toward providing a hybrid platform.

Figure 1 **Windows Azure Platform Is a PaaS Offering**

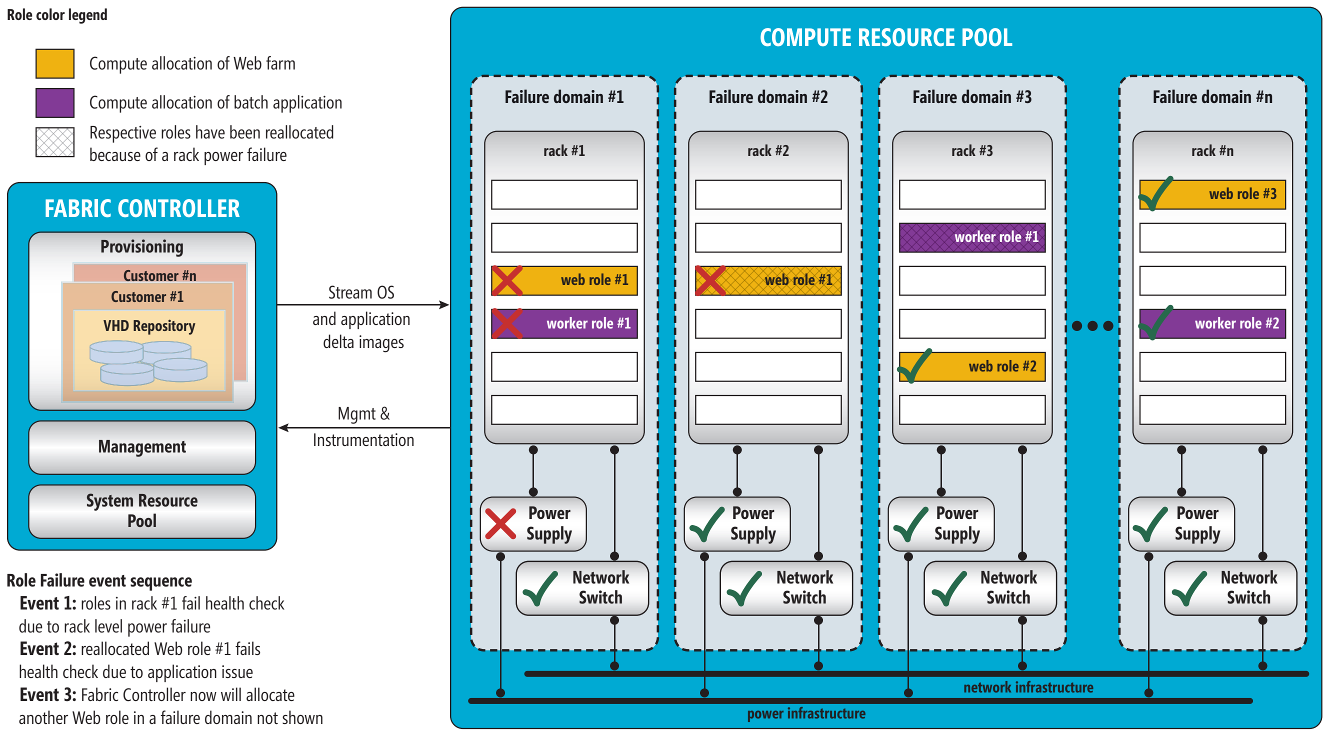

Role color legend
Compute allocation of Web farm
Compute allocation of batch application
Respective roles have been reallocated because of a rack power failure
FABRIC CONTROLLER
Provisioning
Customer #n
Customer #1
VHD Repository
Management
System Resource Pool
Stream OS and application delta images
Mgmt & Instrumentation
COMPUTE RESOURCE POOL
Failure domain #1
Failure domain #2
Failure domain #3
Failure domain #n
rack #1
rack #2
rack #3
rack #n
web role #1
worker role #1
web role #1
worker role #1
web role #2
web role #3
worker role #2
Power Supply
Network Switch
network infrastructure
power infrastructure
Role Failure event sequence
Event 1: roles in rack #1 fail health check due to rack level power failure
Event 2: reallocated Web role #1 fails health check due to application issue
Event 3: Fabric Controller now will allocate another Web role in a failure domain not shown

Figure 2 **Conceptual View of Compute Infrastructure (Windows Azure Setup May Be Different)**

realize this and educate business units about the importance of governance. They also should design new governance processes or modify the existing ones to accommodate SaaS.

Current IT environments may preclude small and midsize enterprises from having the necessary capabilities to optimally run their businesses because of the burden of huge IT investments. SaaS may provide to every company the same kind of IT capabilities that are now affordable only by large enterprises. Since SaaS doesn't require heavy IT investments, it can level the playing field for small companies, putting enterprise-class IT capabilities within their grasp.

Figure 3 **Service Model for Web and Worker Roles**

ShoppingListService Definition

```
<?xml version="1.0" encoding="utf-8"?>
<ServiceDefinition name="ShoppingList">
<WebRole name="ShoppingList_WebRole">
<LocalResources>
<LocalStorage name="ShoppoingList_ImageCache" sizeInMB="100"
cleanOnRoleRecycle="false"/>
</LocalResources>
<InputEndpoints>
<InputEndpoint name="HttpIn" protocol="http" port="80" />
<InputEndpoint name="HttpsIn" protocol="https" port="443" />
</InputEndpoints>
<ConfigurationSettings>
<Setting name="DiagnosticsConnectionString" />
<Setting name="DataConnectionString" />
<Setting name="ShoppinglistOut"/>
</ConfigurationSettings>
</ WebRole>
< WorkerRole name="ShoppingList_WorkerRole">
<Instances count="2" />
<ConfigurationSettings>
<Setting name="DiagnosticsConnectionString" />
<Setting name="DataConnectionString" />
<Setting name="ShoppinglistIn"/>
</ConfigurationSettings>
< WorkerRole />
</ServiceDefinition>
```

ShoppingListService Configuration

```
<?xml version="1.0"?>
<ServiceConfiguration serviceName="ShoppingList">
</Role>
<Role name="ShoppingList_WebRole">
<Instances count="3" />
<ConfigurationSettings>
<Setting name="DiagnosticsConnectionString" value=
              "UseDevelopmentStorage=true" />
<!-- flip the commenting of the following two lines for application
storage needs on  local dev fabric -->
<!--<Setting name="DataConnectionString" value=
                  "UseDevelopmentStorage=true" />-->
<Setting name="DataConnectionString"
value="DefaultEndpointsProtocol=http;
AccountName=<<account name>>;AccountKey=<<account key>>" />
<Setting name="ShoppinglistOut" value="shoppinglistq"/>
</ConfigurationSettings>
</Role>
<Role name="ShoppingList_WorkerRole">
<Instances count="2" />
<ConfigurationSettings>
<Setting name="DiagnosticsConnectionString" value=
              "UseDevelopmentStorage=true" />
<!-- flip the commenting of the followign two lines for local dev fabric
-->
<!--<Setting name="DataConnectionString" value=
                  "UseDevelopmentStorage=true" />-->
<Setting name="DataConnectionString"
value="DefaultEndpointsProtocol=http;
AccountName=<<account name>>;AccountKey=<<account key>>" />
<Setting name="ShoppinglistIn" value="shoppinglistq"/>
</ConfigurationSettings>
</Role></ServiceConfiguration>
```

From the service provider perspective, any small company can become a SaaS provider and compete with large software houses. Such companies now can focus on their core domain strengths instead of outlaying scarce capital for acquiring and managing hardware and software infrastructure.

Platform as a Service

SaaS seems to be the right thing to do for all software needs of a company. However, every company is unique in its IT personality, resulting from legacy technology as well as from its particular business domain. Finding a SaaS service for every line-of-business need is often impossible, so companies need to continue building applications. Platform as a Service (PaaS) fills the needs of those who want to build and run custom applications as services. These could be ISVs, value-added service providers, enterprise IT shops and anyone who needs custom applications. PaaS offers hosted application servers that have near-infinite scalability resulting from the large resource pools they rely on. PaaS also offers necessary supporting services like storage, security, integration infrastructure and development tools for a complete platform.

A service provider offers a pre-configured, virtualized application server environment to which applications can be deployed by the development staff. Since the service providers manage the hardware (patching, upgrades and so forth), as well as application server uptime, the involvement of IT pros is minimized. Developers build applications and annotate the applications with resource descriptors. Upon deployment, the provisioning engine binds the necessary infrastructure capabilities declared in the descriptors to the application. The resources may include network endpoints, load balancers, CPU cores, memory and software dependencies. On-demand scalability combined with hardware and application server management relieves developers from infrastructure concerns and allows them to focus on building applications. PaaS is generally suitable for brand-new applications, as legacy applications often require extensive refactoring to comply with sandbox rules.

Infrastructure as a Service

Infrastructure as a Service (IaaS) is similar to traditional hosting, where a business will use the hosted environment as a logical extension of the on-premises datacenter. The servers (physical and virtual) are rented on an as-needed basis, and the IT professionals who manage the infrastructure have full control of the software configuration. Some providers may even allow flexibility in hardware configuration, which makes the service more expensive when compared to an equivalent PaaS offering

The software composition may include operating systems, application platforms, middleware, database servers, enterprise service busses, third-party components and frameworks, and management and monitoring software. With the freedom to choose the application server comes flexibility in choosing the dev tools as well. This kind of flexibility increases the complexity of the IT environment, as customer IT professionals need to maintain the servers as though they are on-premises. The maintenance activities may include patching and upgrades of the OS and the application server, load balancing, failover clustering of database servers, backup and restoration, and any other activities that mitigate the risks of hardware and software failures.

The development staff will build, test and deploy applications with full awareness of the hardware and software configuration of the servers. Often disaster recovery and business continuity are the responsibilities of the customer. One important benefit of IaaS is that it can allow the migration of legacy applications to the cloud. Since the flexibility of IaaS allows the construction of any configuration, the portability of an application among cloud providers is difficult. Legacy application migration is the sweet spot for IaaS, as it allows mimicking the corporate infrastructure in the cloud. The flexibility of IaaS also enables new applications that require significant control of software configuration. For example, some applications may require the installation of third-party libraries and services, and IaaS allows such installation with no constraints.

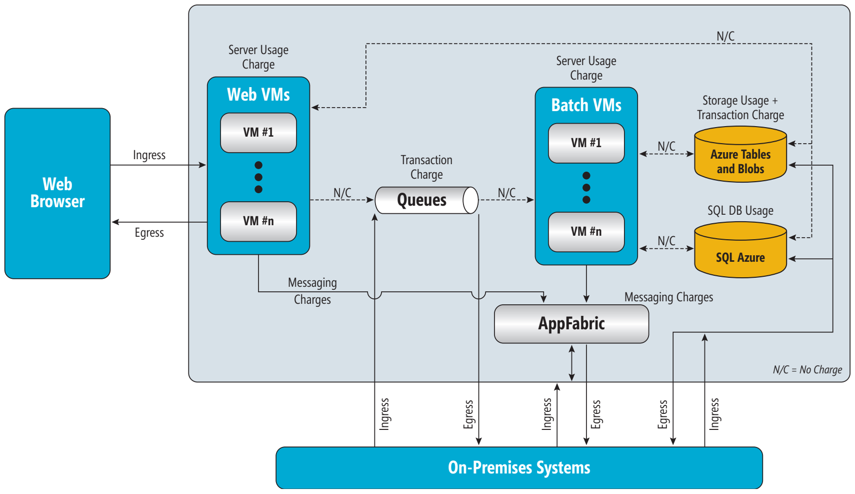


Figure 4 **Windows Azure Platform Application Charge Model**

The Windows Azure platform has all the benefits of PaaS, while at the same time promising to be as flexible as IaaS, as illustrated in **Figure 1**. The Windows Azure platform combines large pools of compute (commodity servers), networking and storage resources into a utility computing environment from which customers can draw resources on-demand and pay only for the usage. Typical of cloud environments, the Windows Azure platform helps customers avoid upfront capital outlays and allows the growth of IT capabilities on an as-needed basis.

Windows Azure Platform

The Windows Azure platform provides a hosted application server and the necessary storage, networking and integration infrastructure for building and running Windows applications. The Windows Azure platform relies on large pools of commodity hardware in creating the utility computing environment. **Figure 2** shows the Windows Azure platform resource model where virtualized storage, network and compute resources are deployed on demand by the provisioning policies set at deployment time. The Fabric Controller is the brain of the entire ecosystem, with a set of dedicated resources that aren't part of the application resource pool. Since the Fabric Controller can't ever fail, it provides a highly redundant hardware and software environment.

The compute resource pool comprises commodity resources that are made fault-tolerant by the Fabric Controller. The Fabric Controller is architected for early detection of application failures, and spawns additional instances to meet contractual service-level agreements. Since the Windows Azure environment is a complete platform for application hosting, it ensures systemic qualities of the application by offering virtually unlimited resources through on-demand provisioning. Unused resources are returned to the pool, thereby increasing utilization. The resources include compute cycles, virtualized storage for persistence and virtualized networking resources for dynamic reconfiguration of private and public network routes. The physical configurations of these resources are by design invisible to application architects and developers.

So, how do the application owners provision these resources? Picking up the phone and calling an IT pro like we do in traditional on-premises environments is out of question, as the massive Windows Azure platform datacenters are managed by a handful of professionals who rely heavily on automation. Normal day-to-day operations of the datacenter require no human intervention. The Windows Azure platform enables application owners to provision necessary resources through machine-readable models comprising resource descriptors. In the Windows Azure platform, these resource descriptors are called service models. These service models specify the application resources and their dependencies sufficient for provisioning the complete runtime infrastructure with no human involvement. Because of this automation, the provisioning time of the application infrastructure is often less than five minutes. When you compare this with the procure-and-provision approach of typical on-premises environments, you grasp the power of cloud computing.

Compute

The compute part of the Windows Azure platform is responsible for providing CPU cycles for executing applications. Applications are hosted inside virtualized environments to prevent any physical dependencies on the underlying operating system and hardware. Loose coupling of applications is accomplished through virtualized resources, which include local files, persistent storage (structured and unstructured), and diagnostic and instrumentation resources. The hosting environment is implemented as a virtual machine, thus any application failures won't impact other applications running on the same physical hardware.

Applications are deployed into the Windows Azure platform as packages of roles and associated executable code and resources. An Azure role describes the characteristics of the hosting environment

Figure 5 **Windows Azure Pricing**

Windows Azure Capability	Charge	Remarks
Server Usage	Small: $0.12 /service-hour Medium: $0.24/service-hour Large: $0.48/service-hour XLarge: $0.96/service-hour	The roles with active applications determine the charges. Small : (1.6Ghz), 1.75GB memory (moderate IO capacity) Medium: (1.6Ghz), 3.5GB memory Large: (1.6Ghz), 7.0GB memory XLarge: (1.6Ghz), 14.0GB memory
Windows Azure Blobs and Tables	$0.15/GB	Daily average measured during each billing cycle. See details on how the charges are computed, as it requires more elaboration.
Transactions	$0.01/10K transactions	Create, Read, Update and Delete into Windows Azure Queues, Blobs and Tables is considered a transaction.
SQL Azure: Web Edition	$9.99/month (1GB RDBMS)	Metadata of a large application or product catalog of a small e-commerce Web site that sells a few hundred items.
SQL Azure: Business Edition	$99.99/month (10GB RDBMS)	Useful for medium businesses. Or, by data sharing, it is possible to build applications with large data storage needs.
AppFabric	$0.15/100K message operations	A message operation may be a service bus message, an access control token request or a service management API call.
Ingress GB	$0.10/GB ($0.30 in Asia)	Only the data transferred in and out of the data center will be billed.
Egress GB	$0.15/GB ($0.45 in Asia)	Only the data transferred in and out of the data center will be billed.

declaratively. When a deployed application is activated, the Azure provisioning environment parses the service model, selects a preconfigured virtual machine (VM) image based on the role type, copies the application bits to the VM, boots the machine and starts the necessary application services. The service definition shown in **Figure 3** represents a Shopping List application, used as a reference throughout this article.

The Shopping List application described in **Figure 3** requests three instances of the Web role and two instances of the worker role. The Web traffic to the multiple instances of a Web role is automatically load-balanced, and all three instances will be provisioned with SSL as well as HTTP endpoints per the service model description. To avoid the total failure of the application, the Fabric Controller spreads the allocations across many failure domains. The failure domain organization is lot more complex than the simplified view shown in **Figure 2**. For simplicity's sake, you can consider each server rack with the associated network switch and power supply as one failure domain.

Per **Figure 2**, the Fabric Controller initially allocated one Web role from each failure domain–racks #1, #3 and #n. I will look at the reliability of the entire Windows Azure compute layer from the perspective of a couple of hypothetical events. During event #1, Web role #1 and worker role #1 stop responding as a result of the rack's power failure. The Fabric Controller starts the provisioning process of Web role #1 and worker role #1 in the available racks–rack #2 and rack #3. Sometime later, event #2 happens, during which the reallocated Web role #1 fails the health check due to an application failure. Now the Fabric Controller starts allocating Web role #1 to one of the other available racks.

During the course of these events, application availability isn't impacted. The Web page requests continue to be served by at least two Web roles, while at least one worker role continues to pull transactional items from the queues and to write to Windows Azure Storage. The Fabric Controller strives to attain the equilibrium of three healthy Web role instances and two worker roles instances at any given instance of time. In reality, the racks may be equipped with redundant power supplies and network switches, hence the role recycling and reallocation may often occur due to application issues or scaling up to meet scalability goals.

The Shopping List service model requested two worker roles to avoid a single point of failure. Even though the Web tier and the batch tier are decoupled through Windows Azure Queues, it's still a good practice to request at least two worker roles for hosting time-sensitive, mission-critical batch services. You may just get away with one worker role if the hosted service isn't time-sensitive.

Figure 1 indicates that the Windows Azure platform is striving to give the benefits of PaaS but at the same time is capable of attaining the flexibility of IaaS. This is enabled by the policy-based deployment model manifested by Web roles, worker roles, CGI roles and a multitude of other roles to come in the future. The list of supported roles will continue to grow to satisfy the diverse application and deployment needs of customers.

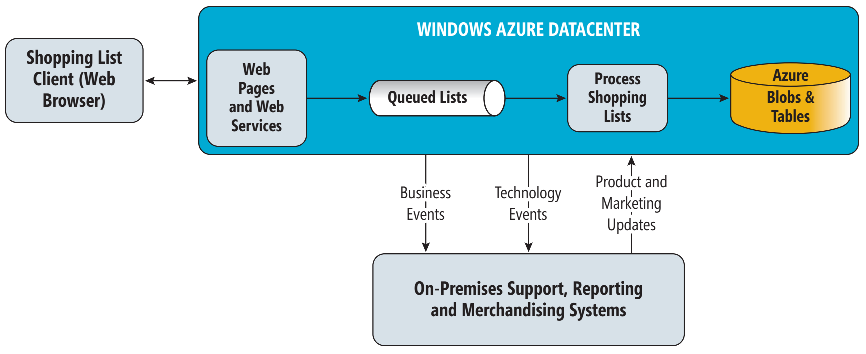

Shopping List Client (Web Browser)
WINDOWS AZURE DATACENTER
Web Pages and Web Services
Queued Lists
Process Shopping Lists
Azure Blobs & Tables
Business Events
Technology Events
Product and Marketing Updates
On-Premises Support, Reporting and Merchandising Systems

Figure 6 **Shopping List Application on the Windows Azure Platform**

Cost-Oriented Architecture for the Cloud

Architecture decisions can have profound impacts on the economics of operations for small and large enterprises. Even though cloud computing is about IT agility, operating-expense considerations of the enterprises need to be taken into consideration while architecting the solution. The architecture of a cloud application needs to deliver functionality and systemic qualities (scalability, availability, reliability and performance) while at the same time optimizing operational expenses. In on-premises situations, application architects rarely pay attention to the cost of storage, the network bandwidth or the cost of compute cycles, as these are capital expenses incurred at the organization level.

As an example, optimizing application storage often isn't on the top of an architect's tasks, as storage carrying costs aren't part of the operational expenditure. The top priority for on-premises systems is to deliver important systemic qualities within the allocated budget. Architecting systems for optimal operating expense becomes an important element of the software development process in the context of cloud computing.

I will look at the cost model of the Windows Azure platform from the perspective of the Shopping List application as shown in **Figure 4**. The diagram shows the logical architecture view of the Shopping List with the arrows indicating data movement. The dashed arrows indicate intra-datacenter bandwidth consumption, while the solid lines show the data movement in and out of the cloud datacenter.

The Windows Azure platform only considers ingress and egress charges for data transfer, ignoring the local data transfers inside a datacenter. Any data written to Windows Azure Queues won't incur any bandwidth or storage charges, as the storage consumption by queues is highly transient. However, the queues will incur per-transaction fees. Peeking, reading or writing a queue item is considered a transaction.

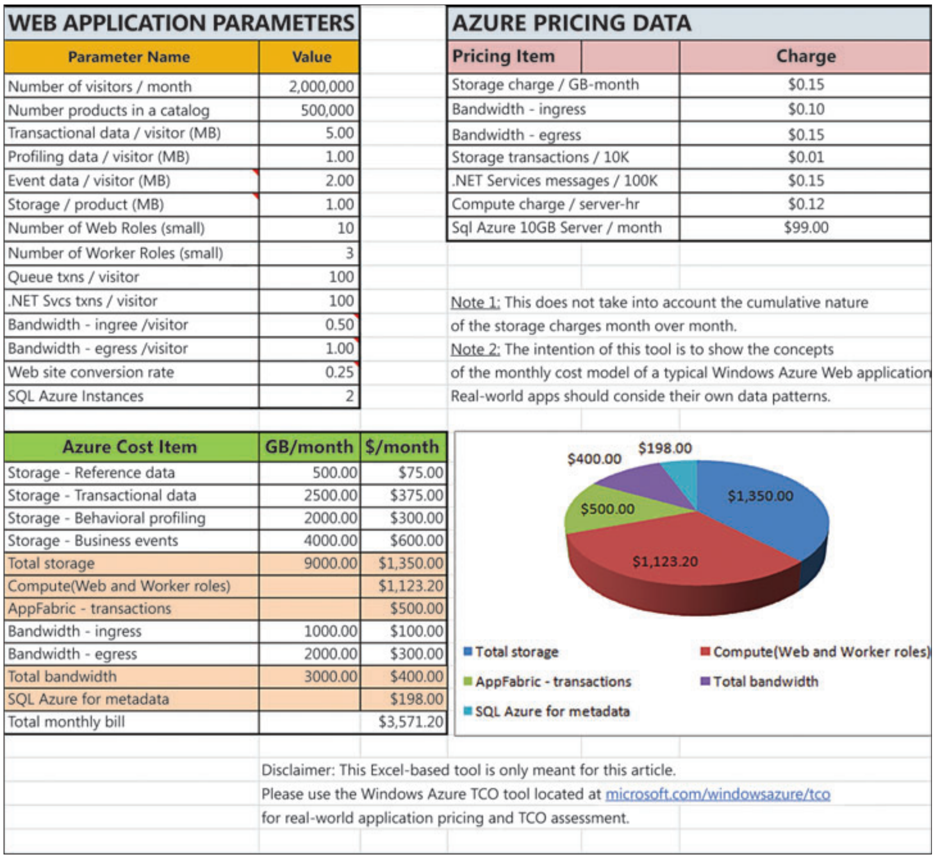
Windows Azure platform server usage charges are based on the number of roles and the amount of time an application is deployed. That means that, even if the application has no requests from end users, the Windows Azure platform billing system will charge per instance-hour for both the role states "suspended" and "started." So it's advised to proactively trim down the number of active roles based on application demands. At the moment, this is a manual process; however, you might be able to auto-scale the application based on application scalability patterns by leveraging diagnostic data and the service management API. Use of Windows Azure Tables, Queues and Blobs will incur storage carrying costs, charged on a monthly basis, per each GB on the record. Please refer to **Figure 5** for Windows Azure pricing that was publicly announced at the time of this writing.

Windows Azure platform pricing is straightforward with the one exception of storage used by Blobs and Tables. An account's Windows Azure Storage usage is measured each day during a billing cycle and a daily average is computed. The charge will be computed by multiplying this daily average by $0.15/GB. For example if you store 20GB on day one, add 10GB on day two, add 5GB on day three, and delete 5GB on day four, with no activity during the rest of the billing cycle, the price will be computed as shown below:

```
((20 +10 + 5 - 5)/30) * 0.15 = $0.15
```

This assumes a 30-day billing cycle. Daily sampling of storage will make sure that applications with highly transient storage needs will still pay for their storage usage, unlike a system that measures only at the end of the billing cycle.

As mentioned earlier, architecture is an important factor in the monthly operating cost of an application. For instance, if an application generates lots of data and only the latest data—say the last two weeks—is needed for the functionality of the application, the architecture can be tweaked to delete the unneeded data or to periodically transfer it to on-premises systems. You may be better off by paying a onetime bandwidth cost than incurring perpetual storage costs. The same can be true with the reference data that is



WEB APPLICATION PARAMETERS

Parameter Name	Value
Number of visitors / month	2,000,000
Number products in a catalog	500,000
Transactional data / visitor (MB)	5.00
Profiling data / visitor (MB)	1.00
Event data / visitor (MB)	2.00
Storage / product (MB)	1.00
Number of Web Roles (small)	10
Number of Worker Roles (small)	3
Queue txns / visitor	100
.NET Svcs txns / visitor	100
Bandwidth - ingree /visitor	0.50
Bandwidth - egress /visitor	1.00
Web site conversion rate	0.25
SQL Azure Instances	2

AZURE PRICING DATA

Pricing Item	Charge
Storage charge / GB-month	$0.15
Bandwidth - ingress	$0.10
Bandwidth - egress	$0.15
Storage transactions / 10K	$0.01
.NET Services messages / 100K	$0.15
Compute charge / server-hr	$0.12
Sql Azure 10GB Server / month	$99.00

Note 1: This does not take into account the cumulative nature of the storage charges month over month.
Note 2: The intention of this tool is to show the concepts of the monthly cost model of a typical Windows Azure Web application Real-world apps should conside their own data patterns.

Azure Cost Item	GB/month	$/month
Storage - Reference data	500.00	$75.00
Storage - Transactional data	2500.00	$375.00
Storage - Behavioral profiling	2000.00	$300.00
Storage - Business events	4000.00	$600.00
Total storage	9000.00	$1,350.00
Compute(Web and Worker roles)		$1,123.20
AppFabric - transactions		$500.00
Bandwidth - ingress	1000.00	$100.00
Bandwidth - egress	2000.00	$300.00
Total bandwidth	3000.00	$400.00
SQL Azure for metadata		$198.00
Total monthly bill		$3,571.20

Disclaimer: This Excel-based tool is only meant for this article.
Please use the Windows Azure TCO tool located at microsoft.com/windowsazure/tco for real-world application pricing and TCO assessment.

Figure 7 **Windows Azure Operating Expenses Calculator for an E-Commerce Application**

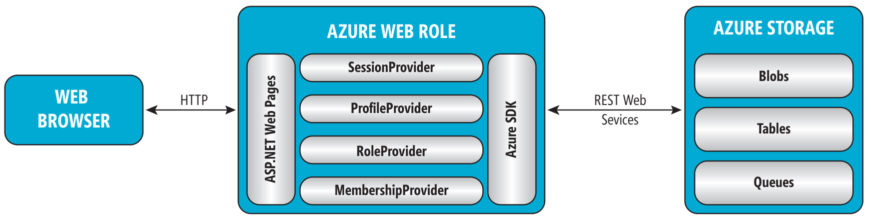


Figure 8 **Windows Azure ASP.NET Providers**

no longer part of the active data set. This approach may work well for companies that have already invested in data archival capacity.

The Application Scenario

I will look at various aspects of the Windows Azure platform in the context of an industrial-strength e-commerce scenario: the Shopping List application. I will focus on creating a grocery list and saving it for later use while shopping at the store. The Web UI composes the shopping list and uses Web services to save it to Windows Azure Storage. For scalability, the Web tier writes shopping lists to a Windows Azure Queue; periodically, a batch process polls the lists from queues and saves them to Windows Azure Tables. I will use Windows Azure-based authentication and role-based security to demonstrate real-world solution aspects.

In the context of the cost-oriented architecture discussed previously, various decisions will impact monthly operational expenses. Here are a few aspects of a system that must be considered before proceeding with the architecture:

1. Growth rate of reference data
2. Growth rate of transactional data
3. Capture rate of behavioral profiling data
4. Growth rate of business event data
5. Capture rate of system event data
6. Media content related to products
7. Using queues vs. direct interaction with persistent storage

My Shopping List scenario didn't include much media content, so that wasn't a big factor in the cost equation, but it may be very important to consider for content sites that deliver videos, imagery and audio streams. **Figure 7** shows for a typical application the monthly operational cost on the Windows Azure platform. The spreadsheet doesn't include the personnel costs for development, operational support and end user support.

A cloud computing environment will reduce the number of operational support staff, so this should be factored in when comparing the ROI between on-premises and the cloud. Also, it's important to include power and depreciated capital expense per application in the equation for ROI. Current on-premises application cost models often don't include these expenses, as it's very difficult to break down the power consumed on a per-application basis. The same is true for cooling and floor space. ROI calculators can use educated guesses in the absence of objective cost breakdowns.

The simple cost calculator shown in **Figure 7** estimates the operating expense of applications hosted on the Windows Azure platform. This Microsoft Excel-based tool allows various input parameters of a typical e-commerce application, and it computes the monthly operational cost using the Windows Azure platform pricing table shown in **Figure 5**. Please keep in mind that the default parameters used in the tool are fictitious; you need to take your own system into consideration before making decisions based on the tool output. The cost calculator is driven by the number of visitors per month and assumes a certain number of page views and transactional and event data creation. The Windows Azure platform team created a more comprehensive tool that calculates the monthly cost of an application and also compares the TCO of on-premises applications with that of Windows Azure. The Windows Azure TCO tool can be accessed at microsoft.com/windowsazure/tco.

As shown in **Figure 7**, our fictitious application generates 9000GB of data in a given month, which costs about $1,350 per month if we were to store this inside Windows Azure Tables. Please keep in mind that **Figure 7** only shows point-in-time storage, and event-data charges can accumulate as the application continues to operate. Such costs can be optimized by tuning the amount of event data captured as an application matures operationally. The cost calculator is driven by the number of visitors per month and uses a hypothetical number of 10 Web roles and 3 worker roles. The total monthly bill is $3,571.

Figure 9 **Web.config Changes for Windows Azure ASP.NET Providers**

```
<system.web>
  ... ... ... ...
<authentication mode="Forms" />
<!-- Membership Provider Configuration -->
<membership defaultProvider="TableStorageMembershipProvider"
userIsOnlineTimeWindow="20">
<providers>
<clear/>
<add name="TableStorageMembershipProvider"
type="Microsoft...AspProviders.TableStorageMembershipProvider"
description="Membership provider using Azure storage"
applicationName="ShoppingList"
... ... ... ... ...
minRequiredNonalphanumericCharacters="0"
requiresUniqueEmail="true"
passwordFormat="Hashed"/>
</providers>
</membership>
<sessionState mode="Custom"
customProvider="TableStorageSessionStateProvider">
<providers>
<clear />
<add name="TableStorageSessionStateProvider"
type="Microsoft...AspProviders.TableStorageSessionStateProvider"
applicationName="ShoppingList"/>
</providers>
</sessionState>
<roleManager enabled="true"
defaultProvider="TableStorageRoleProvider"
cacheRolesInCookie="true"
cookieName=".ASPXROLES"
cookieTimeout="30"
... ... ... ... ...
cookieProtection="All">
<providers>
<clear/>
<add name="TableStorageRoleProvider"
type="Microsoft....AspProviders.TableStorageRoleProvider"
description="Role provider using table storage"
applicationName="ShoppingList" />
</providers>
</roleManager>
... ... ... ...
</system.web>
```

Alternatively, the application can be architected to channel the event data by paying onetime bandwidth costs ($0.10/GB transferred out) to an already-depreciated on-premises storage system, if it exists. Similar strategies can be applied to transactional and behavioral profiling data to avoid cumulative storage charges.

Compute charges aren't cumulative in nature and thus have less impact on the overall operational expenditure of the application. However, there is opportunity to tune the number of active Web and batch role instances based on the observed scalability profile of the application, to get marginal relief on the operating expenses. Between compute and storage charges, compute usage can be controlled at any given time, whereas storage cost depends on architectural decisions that can't be undone easily once the application is built. So my suggestion is to get your persistence architecture right the first time.

In addition to the cost model of the application, large enterprises will pay close attention to application security, which I will explore now.

Compute Security

Enterprises are finicky about application and data security in the cloud. While security of the datacenter, infrastructure and the operating system are taken care of by Microsoft, application security is still the responsibility of the application owners. I will look at application security from the perspective of my Shopping List Web application. Securing a Windows Azure platform application is similar to its on-premises counterpart. The Windows Azure platform provides various system components to help developers integrate security into applications. These system components allow basic, self-contained authentication and authorization to federated scenarios suitable for large enterprises.

Basic Identity

A basic identity model, as the name suggests, implements a self-contained identity architecture to meet the needs of one application or a co-located group of applications that share the same set of users and are tightly coupled to the same identity system at the implementation level. The Windows Azure platform samples contain a set of ASP.NET providers (membership, role, profile and session) that can be used for implementing a basic identity solution. Windows Azure ASP.NET providers are implemented on Windows Azure Storage, which includes Windows Azure Tables and Blobs. These providers implement the ASP.NET provider contracts and leverage StorageClient APIs that are part of the Windows Azure platform SDK. The schematic of the providers is shown in **Figure 8**.

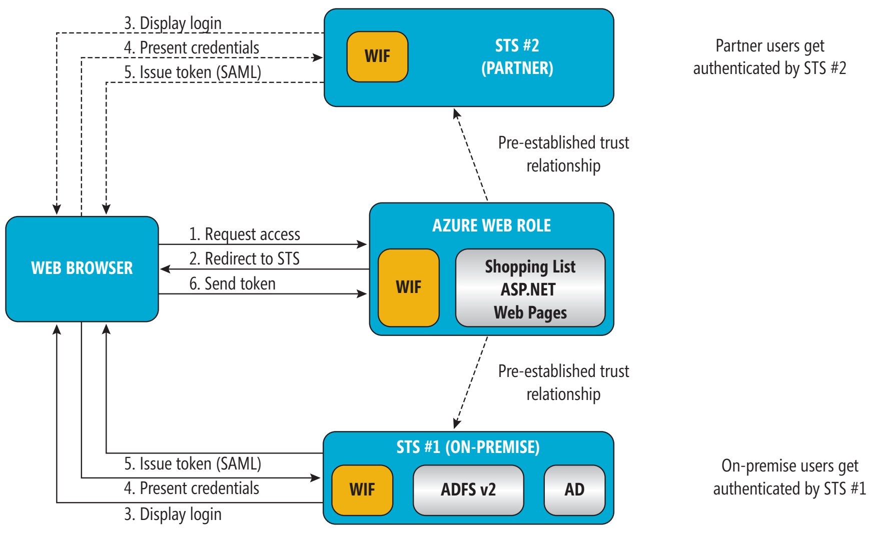


Figure 10 **Multiple Token Services Can Be Registered with Windows Azure Applications**

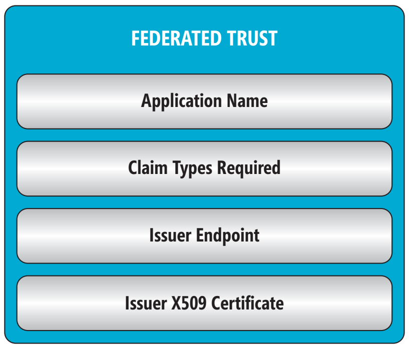


Figure 11 **Federated Trust Descriptor**

In order for the applications to use Windows Azure ASP.NET providers, the Web.config file needs to be modified to remove the default providers and include new ones. The configuration changes shown in **Figure 9** are similar to the changes that must be made for custom ASP.NET providers in on-premises situations.

Once the ASP.NET providers are configured, authentication, authorization and user profiles can be implemented similarly to traditional ASP.NET applications. Note that the configuration in **Figure 9** contains a Windows Azure storage-based session provider, which allows the storage of session states on a durable medium. Since Windows Azure load balancers don't support sticky sessions, storing session data on Windows Azure storage offers a better user experience through session-based personalization. The basic identity model is suitable for applications that have user identity lifecycles (user account creation, usage and closure) that begin and end in the same application. Basic identity implementation may be elected for a variety of reasons, including the following:

- An application wants to retain the complete ownership of the user identity records
- A lack of infrastructure for implementing federated identity store and supporting services
- Applications with a short lifespan (for example, marketing contests and promotions) that require user registration

Windows Azure ASP.NET providers can also be used to authenticate users from AJAX as well as Silverlight applications. The AJAX callable AuthenticationService, ProfileService and RoleService classes, located inside System.Web.Extensions.dll, can be published as .svc endpoints through the Windows Azure Web role. Keep in mind that these services require ASP.NET compatibility for accessing HTTP context-specific data. The article titled "Build Line-Of-Business Enterprise Apps with

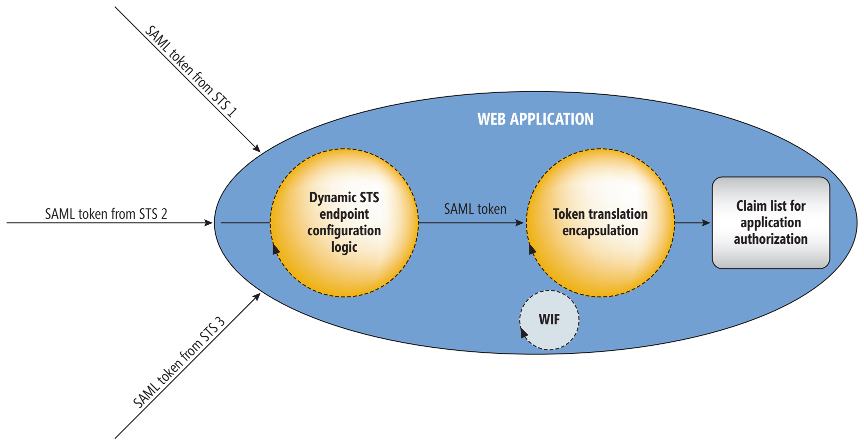

SAML token from STS 1
SAML token from STS 2
SAML token from STS 3
WEB APPLICATION
Dynamic STS endpoint configuration logic
SAML token
Token translation encapsulation
Claim list for application authorization
WIF

Figure 12 **Federated Identity System with Multiple Token Providers**

Silverlight, Part 2" (msdn.microsoft.com/magazine/dd434653), gives detailed information on setting up the above services to be called from Silverlight or AJAX.

Federated Identity Model

Federated identity is necessary for applications that include supply chain, value chain, collaboration and social networking, as well as applications that integrate popular identity stores on the Internet. The Windows Azure ASP.NET stack can be combined with Windows Identity Foundation (WIF) to integrate with one or more security token service providers. WIF works in conjunction with the pre-established trust relationships enabled by WS-Trust and WS-Federation. **Figure 10** shows a conceptual view of the Shopping List application working with two token providers—one on-premises and the other a fulfillment partner.

The trust describes the Secure Token Service (STS) endpoints and the necessary X509 certificates for signing token requests and responses. **Figure 11** shows the trust schematic, the XML representation of which will be included in the Shopping List application configuration at the time of deployment. Users get authenticated in their respective systems and the resulting Security Assertion Markup Language (SAML) token gets forwarded to the requesting application.

As shown in **Figure 10**, when a user accesses secure Web content on the Windows Azure platform-hosted Shopping List application, WIF forwards the request to a Shopping List STS URL present in the trust configuration. The Shopping List STS gathers credentials, authenticates users against Active Directory, constructs a SAML token with the help of Active Directory Federation Services (ADFS, formerly "Geneva Server") and forwards it to the Shopping List application via the Web browser. WIF running inside the Shopping List site on Windows Azure will extract SAML claims and perform authorization checks.

When multiple STSes are involved, a Web site will have to implement token translation logic for converting diverse tokens into a canonical format. To minimize the impact of introducing a new STS into the system, the token translation logic can be externalized or encapsulated into a component that can be modified without impacting the applications that consume them. **Figure 12** shows the token translation schematic that works in conjunction with WIF.

Scenarios such as the following will be enabled by the federated identity model:

- Storage of identity records on-premises for regulatory compliance
- Leveraging the existing on-premises application security infrastructure
- Integrating with partners in the value chain and supply chains
- Single sign-on between the on-premises and the Windows Azure platform application

Often, large enterprises have already implemented authentication services and directory servers that need to be leveraged for securing applications. The Windows Azure platform allows leveraging of the cloud for expedited application deployment while at the same time taking advantage of the existing infrastructure for security. Also, the Windows Azure platform by design allows the use of federated identity that enables various integration scenarios across business partners and value chains.

Windows Azure Storage

Applications and services deployed on the Windows Azure platform may use Windows Azure Storage for persistence of

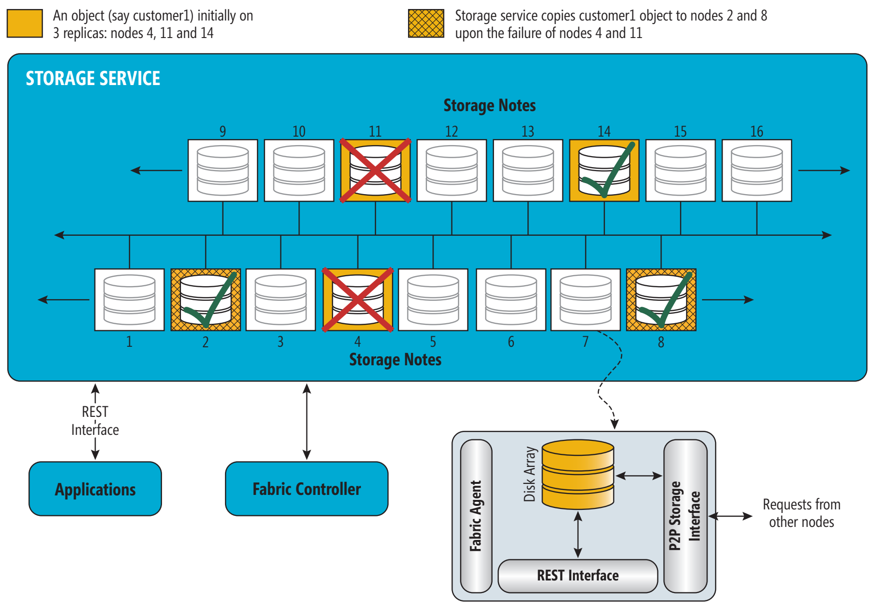
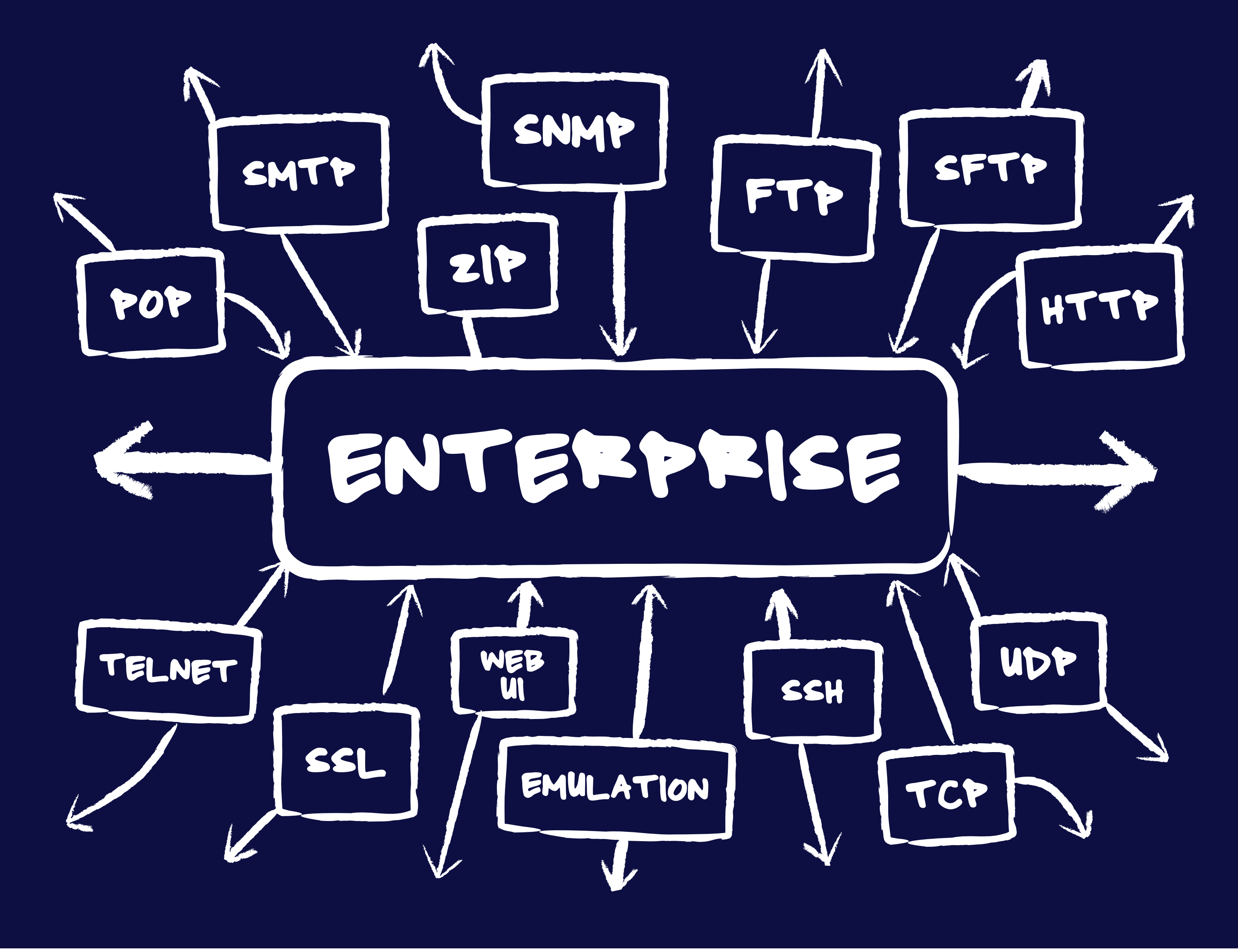

An object (say customer1) initially on 3 replicas: nodes 4, 11 and 14
Storage service copies customer1 object to nodes 2 and 8 upon the failure of nodes 4 and 11
STORAGE SERVICE
Storage Notes
9
10
11
12
13
14
15
16
1
2
3
4
5
6
7
8
Storage Notes
REST Interface
Applications
Fabric Controller
Fabric Agent
Disk Array
P2P Storage Interface
REST Interface
Requests from other nodes

Figure 13 **Storage Service**



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	**FTP**	**SMTP**	**DNS**	**Telnet**
UDP	**SFTP**	**IMAP**	**Rlogin**	**VT Emulation**
TCP	**HTTP**	**S/MIME**	**Rsh**	**ZIP Compression**
SSL	**POP**	**Ping**	**Rexec**	*more...*

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

Download a fully functional product trial today!

DART.com

unstructured and semi-structured content. Windows Azure Storage comprises three fundamental capabilities necessary for building industrial-strength applications and services: Tables, Blobs and Queues. Windows Azure Storage is a massively scalable and highly reliable persistence mechanism that is also accessible to the applications hosted on-premises through industry-standard Web services interface like REST. For on-the-wire privacy, Windows Azure Storage supports SSL (HTTPS)-based access in addition to the standard HTTP protocol. Scalability and other systemic qualities are achieved through a large storage farm comprising commodity server hardware and disk arrays, which is managed by Windows Azure Storage software. The storage access is load-balanced automatically across a set of nodes, for scalability and availability. Each node is responsible for a finite amount of physical storage. Access to storage outside a node's scope is accomplished through a peer-to-peer interface. The reliability is achieved through the redundancy of the stored entities (such as ShoppingList) on multiple nodes. The storage software makes multiple replicas (three at the time of this writing) of the data automatically once a write occurs. Storage supports atomic transactional writes, and the transaction will complete only after all the replicas are written to the drives. **Figure 13** shows a collection of commodity storage nodes forming the Windows Azure Storage Service.

While being used, any storage drive anywhere may fail, the possibility of which is shown by the red "X" on node numbers 4 and 11. Once the storage service identifies a failed drive, it replicates the data from a functioning drive to a new node. The storage service is always compliant with the replica policies at any given point in time. As mentioned earlier, the request traffic from applications will be load-balanced across multiple nodes.

This kind of architecture will help the massive scales required by public cloud PaaS offerings such as the Windows Azure platform. As shown in **Figure 13**, let us assume that nodes 4, 11 and 14 own the initial three replicas of a piece of data. In the event of the failure of nodes 4 and 11, node 14 will continue servicing the requests directly, as well as quickly re-replicating the data to at least two additional nodes (node 2 and 8) to keep the data at a healthy number of replicas.

Storage Security

Windows Azure Storage relies on Hash-based Message Authentication Code (HMAC) for authenticating the REST Web requests. The shared, secret key associated with the Windows Azure Storage project is combined with the HTTP request in computing a 256-byte hash that gets embedded as an "authorization" header into the Web request. The same process is repeated on the server to verify

Figure 14 **Pseudo-Code that Shows the Authenticated Creation of Windows Azure Tables and Data**

```
[DataServiceKey("TableName")]
Public class StorageTable
{
  Private string _tableName;
  Public string TableName
{
  get { return this._tableName; }
  set { this._tableName = value; }
}
}

Public class Customer: TableServiceEntity
{
  Public string Name { get; set; }
  Public string CustomerID { get; set; }
  public Customer()
{
  PartitionKey = "enterprise";
  RowKey = string.Format("{0:10}_{1}", DateTime.MaxValue.Ticks -
  DateTime.Now.Ticks, Guid.NewGuid());
}
}
CloudStorageAccount _storageAccount = CloudStorageAccount.FromConfigurati
onSetting("DataConnectionString");

Public void CreateMultipleCustomers(List<Customer> customers)
{
  TableServiceContext tsc = new
  TableServiceContext(_storageAccount.TableEndpoint.AbsoluteUri,
  _storageAccount.Credentials);
   foreach (Customer cust in customers)
{
  tsc.AddObject("customers", cust);
}
try
{
  DataServiceResponse resp =  tsc.SaveChanges(SaveChangesOptions.Batch);
  foreach (ChangeOperationResponse cor in resp)
{
  if (cor.Error != null)
{
//cor.Headers["Location"] can be parsed to find out the failed
//requests which can be retried after correcting the error condition
}
}
}
catch (Exception ex){ //do something with the exception }
}

protectedvoid linkCreateTables_Click(object sender, EventArgs e)
{
  labelStatus.Text = string.Empty;
try
{
  CreateTable("customers");
  CreateTable("products");
}
catch (DataServiceRequestException ex)
{
  labelStatus.ForeColor = System.Drawing.Color.Red;
  labelStatus.Text = "Error: Table creation error : " + ex.Message;
}
}
//Use ADO.NET services directly to create an Windows Azure Table
Public void CreateTableUsingContext(AzureStorageTable storageTable)
{
  TableServiceContext tsc = new
  TableServiceContext(_storageAccount.TableEndpoint.AbsoluteUri,
  _storageAccount.Credentials); tsc.AddObject("Tables", storageTable);
try
{
  DataServiceResponse resp = tsc.SaveChanges(SaveChangesOptions.None);
//handle errors
}
catch (Exception ex){//do something here}
}
//much simpler way of creating an Windows Azure Table
  publicvoid CreateTable(string tableName)
{
CloudTableClient ctc = _storageAccount.CreateCloudTableClient();
try
{
  ctc.CreateTable(tableName);
}
catch(Exception e) { //handle exception }
}
```

the authenticity of the request. Windows Azure Table, Queue and Blobs all follow the same authentication process, while the payload and the target URLs are different for each of the storage types. The following are the URLs for accessing the above three storage capabilities under the project, say "hkshoppinglist":

- http(s)://hkshoppinglist.blob.core.windows.net/
- http(s)://hkshoppinglist.queue.core.windows.net/
- http(s)://hkshoppinglist.table.core.windows.net/

The code sample in **Figure 14** shows the creation of multiple Windows Azure Tables as a part of the storage preparation for application deployment.

Using Windows Azure Tables as an example, I will show some simple ways of preparing Windows Azure Storage for transactional population of data. The code samples show the creation of "customers" and "products" tables using TableServiceContext as well as CloudTableClient to illustrate the flexibility of the REST-based interaction. In fact, you can also craft a raw payload, attach HMAC to the Web request and do an HTTP POST to the table URL, but it requires lot of code and should only be done as an academic exercise. The recommended approach is to use StorageClient, which is part of the Windows Azure SDK.

The CreateTableUsingContext function uses the AzureStorageTable class to generate the table creation payload with the help of ADO.NET Data Services. TableServiceContext automatically generates HMAC and attaches to the request using the key contained in the CloudStorageAccount.Credentials property.

Windows Azure Table storage allows batch transactions, as shown in the function CreateMultipleCustomers in **Figure 14**. The batch should not exceed 100 operations in a given change set, and a single batch should not exceed 4MB in size. For more details, please refer to the Windows Azure Storage documentation. Batch transactions are only allowed with the entities belonging to the same partition.

Credentials necessary for the generation of HMAC are specified in the service configuration of the respective Windows Azure role. The following is the format of the connection string for local storage and the cloud:

Local storage:

```
<Setting name="DataConnectionString"
value="UseDevelopmentStorage=true"/>
```

Cloud storage:

```
<Setting name="DataConnectionString"
value="DefaultEndpointsProtocol=
http;AccountName=
<your account>;AccountKey=<your account key"/>
```

There is no notion of role-based security in Windows Azure Storage; so an authenticated request will have complete access to the storage in the context of the storage project. An exception to this is the blob container, which can be public (anonymous) or private. Authorization is the responsibility of the application that consumes the storage services.

Wrapping Up

In this article I only scratched the surface of the Windows Azure platform. I am sure there will be plenty of coverage in the future about Microsoft SQL Azure, AppFabric, various server roles and other security scenarios not covered here. The Windows Azure platform is a cloud computing platform that is architected to enable on-demand utility computing for developing and hosting applications and services.

Large pools of commodity hardware are made highly reliable by software through a high degree of automation. The economic advantages of massive scales are passed back to consumers through a low subscription fee. Subscribers will be charged based on the usage of bandwidth, storage and compute cycles over a monthly billing cycle. The Windows Azure platform comes with the platform components necessary for building enterprise-class applications and services with no upfront commitment of capital or long-term contracts. ■

HANU KOMMALAPATI *is a platform strategy advisor at Microsoft, and in this role he advises enterprise customers in building scalable line-of-business applications on the Silverlight and Windows Azure platforms.*

THANKS to the following technical experts for reviewing this article: *Vittorio Bertocci, Brad Calder, Ryan Dunn and Tim O'Brien*

Extending the MVP Pattern for Enterprise-Class Application UI Architecture

Zhe Ma

Model-View-Presenter (MVP) represents a breakthrough in thinking about UI patterns, and makes it clear that UI designers should maintain separation of concern in their applications.

However, there are many different interpretations of the MVP pattern. For example, some people take for granted that the MVP pattern explicitly represents the UI architecture pattern. This is not exactly true for enterprise-class applications. Compared to other types of UI applications, enterprise-class applications need to deal with many different requirements, more parties involved, more complexity, and many cross-dependencies on other systems such as services, other applications and so on. These particular characteristics have required the UI architecture of enterprise-class applications to have more emphasis on flexibility, maintainability, reusability, implementation consistency, and decoupling business functionality from the underlying technology to avoid dependencies on specific products and vendors.

This article discusses:

- Patterns in UI architecture
- Extending the MVP pattern
- UI pattern elements
- A UI call sequence

Technologies discussed:

Design Patterns, ASP.NET, Windows Communication Foundation

Code download available at:

code.msdn.microsoft.com/mag201002MVP

If only the MVP pattern itself is applied as the UI architecture pattern for enterprise-class applications, some questions will be raised. Here are just a few:

A typical enterprise-class application contains many views, and events occurring in one view could impact other views. For example, clicking a button in one screen could cause a pop-up window to show up, and another screen's data may be updated at the same time. Who is responsible for controlling such screen-flow logic? Should this be controlled by each view's pairing presenter?

In a Service-Oriented Architecture (SOA), the application UI generally gets information through services. For example, the UI would need to call a generated WCF service client proxy in order to call the WCF service to get data. Is it a good design for presenter to call this service client proxy directly? If these services are implemented with different technologies or if service models are changed, how do you design the UI architecture so that the impact of these changes on the UI implementation can be minimized?

Following this train of thought, some implementations might use generated service client proxy models across the application. Are there any risks of doing that? If a dedicated UI model is needed, which part will be responsible for providing the mapping between the service client proxy model and the UI model?

These are not new questions, and many other patterns have been introduced to fill in the gap. For example, the Application Con-

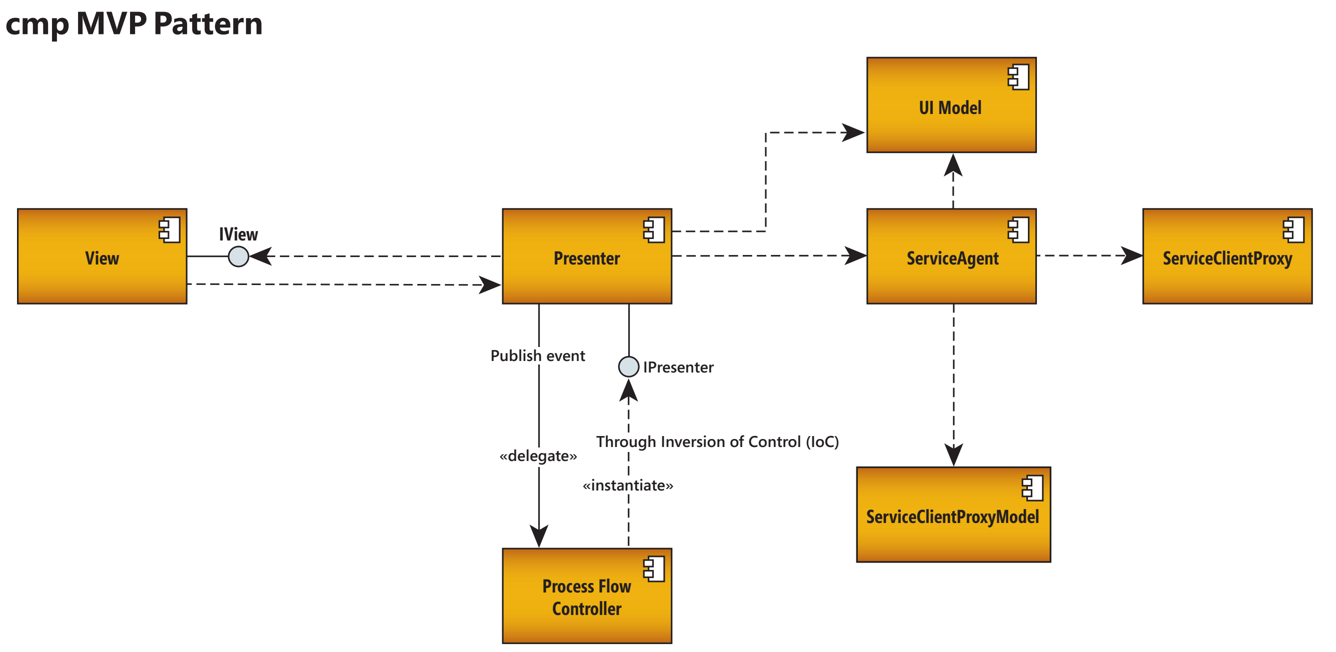


Figure 1 **UI Architecture Based on the Passive View Pattern**

troller pattern (msdn.microsoft.com/library/cc304764) was introduced to assume the responsibility for controlling navigation flow.

I thought it would be helpful to pull some of these disparate MVP-extending discussions together and draw a holistic view of UI architecture design. Looking at the problem from an enterprise-class application perspective, this will help UI architects recognize what key parts are needed for UI design and define a consistent pattern to guide UI application implementation.

The term "MVP pattern" will be used throughout the article, but actually the original MVP pattern has been retired and two variations of the original MVP are currently in place. One is the Passive View pattern and the other is the Supervising Controller pattern. Although either one fits certain scenarios and both have pros and cons, the UI architecture described in **Figure 1** is primarily based on and extended from the Passive View pattern. This certainly doesn't mean UI architecture couldn't be constituted based on the Supervising Controller pattern, but this is my personal preference.

Let's begin the discussion with a clear understanding of what constitutes the UI architecture by extending the MVP pattern. **Figure 1** shows what key parts are needed from a high-level view of UI architecture. In this diagram, seven major parts are introduced: View, Presenter, UI Model, Process Flow Controller, Service Agent, Service Client Proxy Model, and Service Client Proxy.

View

View basically follows the View role in the Passive View pattern. View assumes a simple list of responsibilities, starting with handling UI display layout and presentation-specific logic.

View's second responsibility is to raise events to the Presenter, and it requires several implementations to handle this responsibility. First, View needs to implement the IView interface. To ensure there are few if any implementation impacts on Presenter logic, and to provide unit testing capability, the Presenter should interact with View through an IView interface.

Second, View needs to define public properties that Presenter can interact with. In the Passive View pattern, View does not pass data to Presenter. Instead, it is up to Presenter to choose data it is interested in from View. This type of implementation further reduces the contract binding between View and Presenter and separates their responsibilities more clearly.

One question, though, is about what data type the View properties should use. The ideal way is to have View define these properties using only simple types, such as string, integer and so on. However, in a real-world implementation, it can become tedious if the View defines data this way. It is acceptable to expose a data property by referencing complex types, such as UI Model definitions. This balances architectural purity with implementation considerations.

There are many different interpretations of the MVP pattern.

Third, View also needs to call Presenter operations when events happen in the View. The View can call the Presenter directly without passing any data. There is no loosely coupled design between View and Presenter because View and Presenter are always paired. It is good design to have one Presenter operation dedicated to one View event.

View's last responsibility is to respond to property value updates. Presenter updates View properties to indicate a change. View has the knowledge to decide how to respond to such changes. It could

sd MVP Pattern

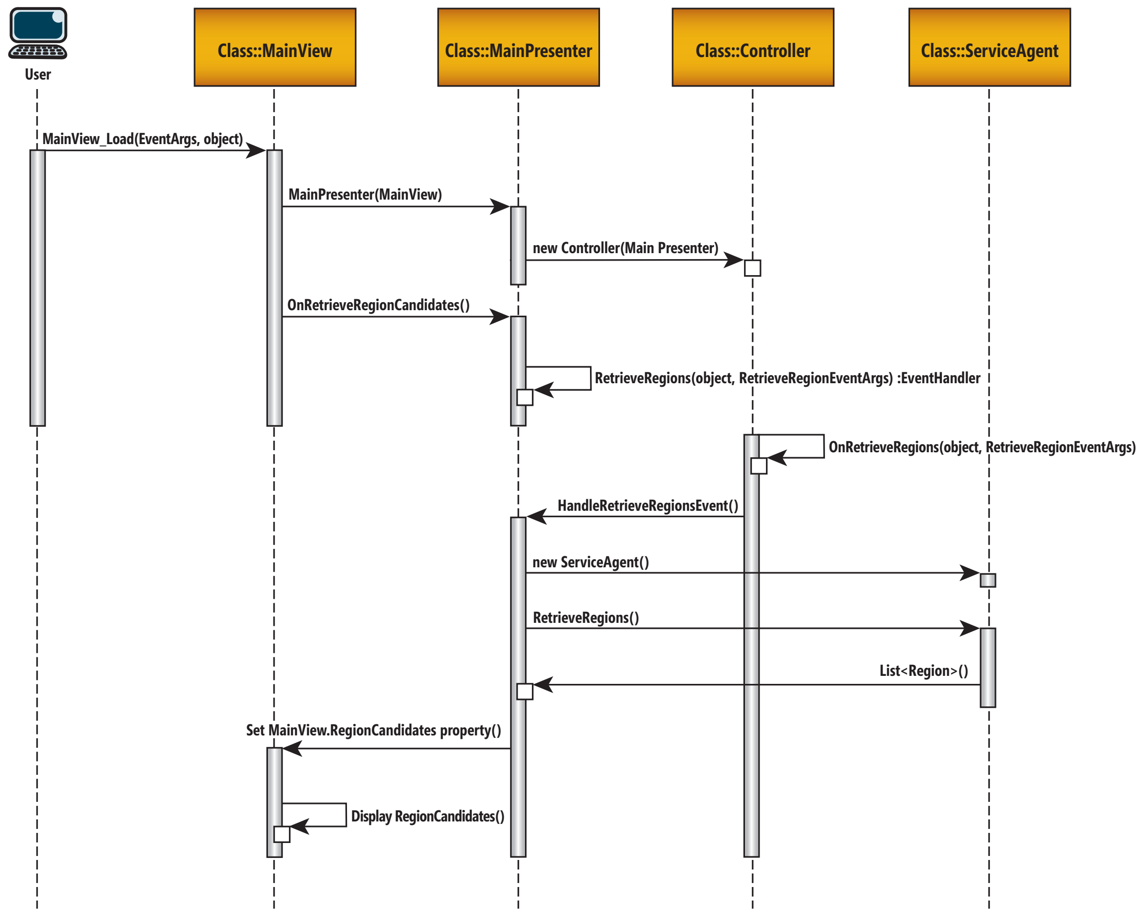

User
Class::MainView
Class::MainPresenter
Class::Controller
Class::ServiceAgent
MainView_Load(EventArgs, object)
MainPresenter(MainView)
new Controller(Main Presenter)
OnRetrieveRegionCandidates()
RetrieveRegions(object, RetrieveRegionEventArgs) :EventHandler
OnRetrieveRegions(object, RetrieveRegionEventArgs)
HandleRetrieveRegionsEvent()
new ServiceAgent()
RetrieveRegions()
List<Region>()
Set MainView.RegionCandidates property()
Display RegionCandidates()

Figure 2 UI Call Sequence

go ahead to refresh the view to reflect the data change, or it could decide not to take any action.

Presenter

Presenter essentially follows the Presenter role in the Passive View pattern. However, here, the Presenter does not determine process flow. The Presenter takes event requests from the View and publishes event requests to Controller to let Controller decide the next step. Because Presenter will not handle process flow logic, it can't know whether the event requests from View will have any impact on other Views. So when Presenter receives event requests from View, it will immediately publish corresponding events so that Controller can respond to these event requests and decide on the next process step. Presenter never assumes it can go ahead and perform some actions until it is instructed by Controller.

Controller decides whether Presenter operations need to take place. When a Presenter operation is called by Controller, Presenter then performs the actions, such as to retrieve data through a Service Agent. If Presenter needs to take actions against services, it does that through Service Agent. It will pass required parameters to the Service Agent and get results back from the Service Agent.

When Presenter is ready to notify View about data changes, it will do so by updating View's property values. Then it is up to View to decide how to display them. As mentioned previously, Presenter interacts with View through the IView interface rather than accessing the View object directly. Since the View instance has already been passed to Presenter when initiating Presenter, Presenter already has a View instance to deal with.

Finally, Presenter can access the UI Model and can put it in a cache if UI Model data needs to be accessed later.

In a SOA, the application UI generally gets information through services.

Process Flow Controller

The Process Flow Controller is close to the Application Controller pattern. The difference is that the sole responsibility of the Process

Flow Controller discussed here is to control process flow based on typed events raised by Presenter. Process flow is not limited to screen navigation flow. It also includes controlling the order of Presenter actions related to event requests.

Process Flow Controller subscribes to events published by Presenter, and responds only to events published by Presenter. These events are typed events. In other words, Process Flow Controller doesn't respond to a general event.

Because Process Flow Controller only responds to a typed event, a process flow actually has already been predetermined when an event occurs. This simplifies Process Flow Controller's logic. Each event published by Presenter contains the data needed for Process Flow Controller to carry on when initiating other Presenter operations.

Process Flow Controller will initiate Presenter and related View instances if they haven't been initiated yet. Inversion of Control (IoC) will be needed due to cross-reference concerns. This also provides a loosely coupled design between Presenter and Process Flow Controller.

UI Model

UI Model basically follows the Model role in the Passive View pattern. In Passive View, Model is really not doing much work and it simply provides the model structure definition. In addition, as described in the Presenter section, Presenter is responsible for maintaining Model state.

The reason I call this UI Model rather than simply Model is to differentiate it from Service Client Proxy Model, which will be described later the article.

UI Model defines the model structure that is suitable for UI application logic handling. The UI model definition may look exactly the same as Service Client Proxy Model. However, in some situations—especially if the UI needs to display data from multiple service sources—a restructured UI Model is needed that will be different from the Service Client Proxy Model.

Service Agent

Service Agent plays an intermediary role between Presenter and Service Client Proxy. The service in the name is not necessarily a Web service. It represents any resources that will provide data or perform business logic. This could be a Web service, but could also be simply file I/O.

Service Client Proxy has specific meaning in Web service technology. Here, I use Service Client Proxy to represent the gateway for a service.

The Service Client Proxy implementation is technically specific. From Presenter's perspective, it would rather not know how data is transmitted or provided. Such technically specific details could be hidden inside of Service Agent. So the layer of Service Agent protects Presenter from being affected by service implementation technology changes, service versioning, service model changes, and so on.

Service Agent provides operations for Presenter to interact with. If a complex type needs to be passed to these operations, you need to define a complex type under UI Model. This is also true for the operation return type. You then pass these operation calls to corresponding Service Client Proxy operations. In some cases, one Service Agent operation may initiate several Service Client Proxy operation calls.

Because Service Agent operations take in complex types defined under UI Model, Service Agent operations need to map from UI Model to Service Client Proxy Model when calling Service Client Proxy operations. When Service Agent operations need to return results back to Presenter, they would map from Service Client Proxy Model to UI Model after getting results from Service Client Proxy operations.

This could be a tedious job. However, there are tools available to map from one model structure to another model structure easily, so this become more of a one-time design job.

Service Client Proxy and Service Client Proxy Model

Service Client Proxy in Web service technology provides local access for the service client even though the service is hosted remotely. In this article, I would describe Service Client Proxy as the gateway to the service. Service Client Proxy Model represents the service contract model definition.

Service Client Proxy passes calls to services and returns service responses. If the service is implemented with ASP.NET Web Services (ASMX) or Windows Communication Foundation (WCF) technologies, the Service Client Proxy can be generated automatically.

Service Client Proxy Model will reflect the service contract model structure definition.

Process Flow Controller subscribes to events published by Presenter.

Implementation Example

To illustrate the UI architecture described in **Figure 1**, let's take a look at a Windows Forms application that demonstrates the implementation. This sample app is included in the download for this article.

This sample application first needs to load a list of regions. When a region is selected, customers that belong to the region are displayed. When a customer is selected, a time range query window will pop up. After a start time and end time are entered, a list of orders that belong to the selected customer will be displayed on the data grid in the main screen.

I will use the scenario of displaying a list of regions to explain how the UI architecture in **Figure 1** is implemented in the sample app. **Figure 2** shows the call sequence for this scenario.

When the Main Screen form is loaded, it first initiates the Presenter interface and passes the current View instance to Presenter's constructor:

```
private void MainView_Load(
  object sender, EventArgs e) {

  _presenter = new MainPresenter(this);
  ...
}
```

The MainPresenter instance initiation will cause MainPresenter's constructor to first assign the passed-in MainView instance to a private variable of type IMainView. It then initiates a Controller instance and passes the current Presenter instance to the Controller constructor:

```
public MainPresenter(IMainView view) {
  _view = view;
  _controller = new Controller(this);
}
```

Controller instance initiation will cause the constructor to assign the passed-in MainPresenter instance to a private variable of type IMainPresenter. This constructor also defines the event handler to be prepared to respond to MainPresenter's published events, such as RetrieveRegions:

```
public Controller(IMainPresenter presenter) {
  _mainPresenter = presenter;
  ...
  _mainPresenter.RetrieveRegions += (OnRetrieveRegions);
}
```

Back in the main screen form load event, Presenter is called to retrieve regions after the Presenter object is initiated:

```
Private void MainView_Load(object sender, EventArgs e) {
  ...
  _presenter.OnRetrieveRegionCandidates();
}
```

When Presenter receives this call, it first publishes the event RetrieveRegions instead of going ahead to retrieve the regions. The RetrieveRegions event has been defined in the IMainPresenter interface and is implemented in MainPresenter:

```
public event EventHandler<RetrieveRegionsEventArgs>
  RetrieveRegions;
  ...

public void OnRetrieveRegionCandidates() {
  if (RetrieveRegions != null) {
    RetrieveRegions(this,
      new RetrieveRegionsEventArgs());
  }
}
```

In the Controller class, since an event handler for RetrieveEvents has been registered, it can respond to the RetrieveRegions event:

```
private void OnRetrieveRegions(
  object sender, RetrieveRegionsEventArgs e) {

  _mainPresenter.HandleRetrieveRegionsEvent();
}
```

Controller decides that the process flow should return control to MainPresenter and asks it to continue to retrieve regions. If Controller needs to initiate presenters other than MainPresenter, it can employ Unity Framework to perform that task.

If Controller needs to initiate presenters other than MainPresenter, it can employ Unity Framework.

In MainPresenter's HandleRetrieveRegionsEvent operation, it calls Service Agent to retrieve regions. For simplicity, my example doesn't actually implement the service. It just writes some dummy data to make the application function. After a result is returned from Service Agent, note that MainPresenter doesn't pass data to MainView. Instead, it updates the MainView's RegionCandidates property:

```
public void HandleRetrieveRegionsEvent() {
  RegionAdminServiceAgent agent =
    new RegionAdminServiceAgent();
  List<Region> regionCandidates = agent.RetriveRegions();
  _view.RegionCandidates = regionCandidates;
}
```

In MainView's RegionCandidates property, it handles the display of regions:

```
public List<UIModel.Region> RegionCandidates {
  set {
    _regionCandidates = value;
    PopulateRegionCandidates();
  }
}
```

This is the whole sequence of retrieving regions and displaying them in the MainView. It definitely involves more steps than simply calling Service Agent to get regions. However, when thinking from the perspective of an enterprise-class application, it not only introduces a loosely coupled design, it also promotes a consistent implementation pattern. This can greatly simplify maintenance and knowledge-transfer for a development team.

Just one more comment about this code example: the whole sequence starts with the first Windows Forms load event. A more advanced implementation could start with Controller and let Controller decide what the first form is to be loaded.

Wrapping Up

In this article, I introduced one approach to UI architecture design based on extending the MVP pattern. UI applications can be complicated and there are many different flavors of UI application design. The technique I present in this article represents one of these many solutions. This is a useful technique in many situations, but be sure it suits your requirements before implementing it.

There are already many UI frameworks on the market and many are based on MVP, Model-View-Controller or patterns developed as extensions of these two. A good first step is to see what major parts are implemented by these frameworks—for example, like the way I abstracted the UI architecture in this article. Falling into implementation details without first considering the big picture is not good architectural thinking. Starting with a broad architectural understanding of the problem at hand ensures not only that the foundational problems of system architecture are resolved, but also that a repeatable and well-thought-out design can be followed.

Finally, in the example of this article, the Controller implementation was created with C#. A better approach might be to use a process flow technology such as Windows Workflow Foundation, which may allow a more flexible design and implementation. However, this technical implementation detail will not affect the underlying principles behind the UI architecture described in this article.

For further discussion of the MVP pattern, see the August 2006 issue of *MSDN Magazine* (msdn.microsoft.com/magazine/cc188690). ■

ZHE MA *is a technical architect of Enterprise Architecture at Unum Group, based in Portland, Maine. He can be reached at zma@unum.com.*

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

LEADTOOLS®

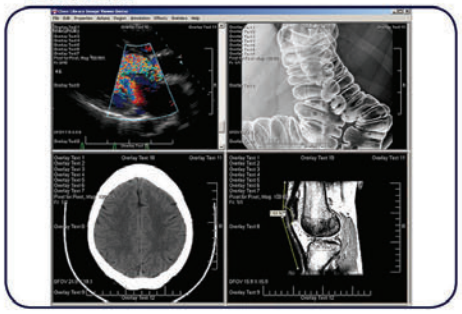
The World Leader in Imaging Development SDKs

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.

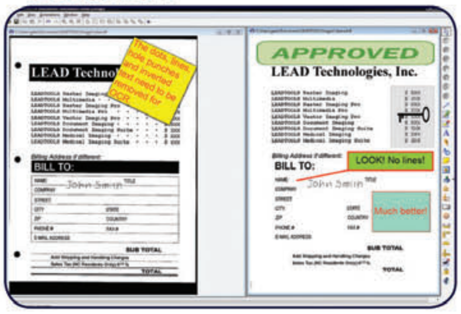
.NET, WPF, WCF, WF, C API, C++ Class Lib, COM & more!

Medical

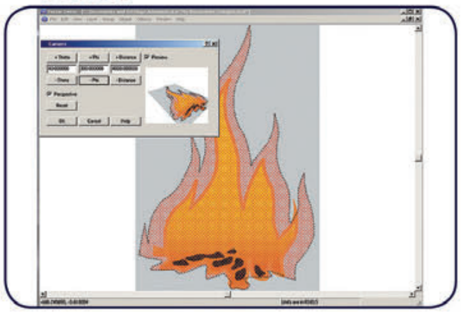


Multimedia

Document



Vector



•**Image Formats & Compression:** Supports 150+ image formats and compressions including TIFF, EXIF, PDF, JPEG2000, JBIG and CCITT.
•**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.
•**Forms Recognition and Processing:** Automatically identify forms and extract user filled data.
•**Barcode:** Detect, read and write 1D and 2D barcodes for multithreaded 32 and 64 bit development.
•**Document Cleanup/Preprocessing:** Deskew, despeckle, hole punch, line and border removal, inverted text correction and more.
•**PDF and PDF/A:** Read and write searchable PDF with text, images 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.
•**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 2008 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.
•**JPIP:** Client and Server components for interactive streaming of large images and associated image data using the minimum possible bandwidth.
•**Scanning:** TWAIN 2.0 and WIA (32 and 64-bit), autodetect optimum driver settings for high speed scanning.
•**DVD:** Play, create, convert and burn DVD images.
•**DVR:** Pause, rewind and fast-forward live capture and UDP or TCP/IP streams.
•**Multimedia:** Capture, play, stream and convert MPEG, AVI, WMV, MP4, MP3, OGG, ISO, DVD and more.
•**Enterprise Development:** Includes WCF services and WF activities to create scalable, robust enterprise applications.

Version 16.5 is a free upgrade for qualifying customers. To see if you qualify, or for more information on what is new in version 16.5, contact: **sales@LEADTOOLS.com**

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

LEAD TECHNOLOGIES INCORPORATED

High Level Design • Low Level Control

The LEADer in Intelligent Imaging

CHARLES PETZOLD

Sound Generation in WPF Applications

A few weeks ago I sat in a new Toyota Prius while the agent at the rental car company explained the unfamiliar controls and indicators arrayed on the dashboard. "Wow," I thought. "Even for a technology as old as the automobile, manufacturers are continually refining the user interface."

In the broadest sense, the user interface is the place where human and machine interact. While the concept is as old as technology itself, the user interface really blossomed as an art form only with the personal computer revolution.

Just a tiny fraction of today's personal computer users can remember the days before the advent of the graphical user interfaces of the Apple Macintosh and Microsoft Windows. At the time (the mid- to late 1980s), some pundits feared that standardization of the user interface would impose an oppressive uniformity over applications. That was not the case. Instead, as the availability of standard controls freed designers and programmers from the need to reinvent the scrollbar, user interfaces actually began to evolve and become much more interesting.

I am convinced that the user interface has become an even more crucial part of application programming.

In this respect, the new paradigms introduced by Windows Presentation Foundation (WPF) have allowed user interfaces to get even fancier. WPF lays down a strong foundation of retained-mode graphics, animation and 3-D. It adds to that a tree-based hierarchical structure of parent and child elements and a powerful markup language known as XAML. The result is unprecedented flexibility in customizing existing controls through templating, and building new controls by assembling existing components.

But these new concepts aren't just for client programming. A healthy subset of the Microsoft .NET Framework, XAML and WPF classes have become available in Web-based programming through Silverlight. The day has already arrived when you can actually share custom controls between client applications and Web applications. I'm sure this trend will continue into mobile applications and eventually encompass many different types of information and entertainment systems, while taking advantage of new technologies such as multi-touch.

For these reasons I am convinced that the user interface has become an even more crucial part of application programming. This column will explore the potential of user-interface design in WPF and Silverlight, including the use of cross-platform code when possible.

Sounding Off

It's not always possible to differentiate between good and bad user-interface choices right away. Clippy—the anthropomorphized paperclip that debuted in Microsoft Office 97—probably seemed like a good idea at the time. For that reason, I'll focus more on the technological potential rather than design. I'll tend to avoid the term "best practices." That's a matter for history and the market.

For example, a good case could be made that computers should not make noise except when they're playing a video or a sound file in response to a specific command from the user. I'm going to ignore that stricture and show you how to play custom sounds in a WPF application by generating waveform data at runtime.

This sound-making capability isn't an official part of the .NET Framework yet, but it's made possible by the NAudio library

Figure 1 **A Class to Generate Sine Wave Samples for NAudio**

```
class SineWaveOscillator : WaveProvider16 {
  double phaseAngle;

  public SineWaveOscillator(int sampleRate):
    base(sampleRate, 1) {
  }

  public double Frequency { set; get; }
  public short Amplitude { set; get; }

  public override int Read(short[] buffer, int offset,
    int sampleCount) {

    for (int index = 0; index < sampleCount; index++) {
      buffer[offset + index] =
        (short)(Amplitude * Math.Sin(phaseAngle));
      phaseAngle +=
        2 * Math.PI * Frequency / WaveFormat.SampleRate;

      if (phaseAngle > 2 * Math.PI)
        phaseAngle -= 2 * Math.PI;
    }
    return sampleCount;
  }
}
```

Code download available at code.msdn.microsoft.com/mag201002WPF.

available on Codeplex (naudio.codeplex.com). Following links from that site, you can check out Mark Heath's blog for some sample code, and Sebastian Gray's site tutorials.

You can use the NAudio library in Windows Forms or WPF applications. Because it accesses Win32 API functions through PInvoke, it can't be used with Silverlight.

For this article, I used NAudio version 1.3.8. When you create a project that uses NAudio, you'll want to compile for 32-bit processing. Go to the Build tab of the Properties page and select x86 from the Platform Target dropdown.

Deliver a constant stream of integers describing a waveform to the two DACs, and stereo sound comes out.

Although the library provides many features for specialized applications that need to use sound, I'm going to show you a technique that might find its way into a more general-purpose application.

Suppose, for example, your application allows the user to drag objects around the window, and you want this dragging to be accompanied by a simple sound (a sine wave, say) that increases in frequency the further the object gets from the center of the window.

This is a job for waveform audio.

Almost all PCs these days include sound-generation hardware, often implemented with a chip or two right on the motherboard. This hardware is usually not much more than a pair of digital-to-analog converters (DACs). Deliver a constant stream of integers describing a waveform to the two DACs, and stereo sound comes out.

How much data is involved? Applications these days commonly generate "CD quality" sound. The sampling rate is a constant 44,100 samples per second. (The Nyquist Theorem states that the sampling rate needs to be at least twice the highest frequency to be reproduced. Humans are commonly said to hear sounds with frequencies between 20Hz and 20,000Hz, so 44,100 is comfortably adequate.) Each sample is a signed 16-bit integer, a size that implies a signal-to-noise ratio of 96 decibels.

Making Waves

The Win32 API provides access to the sound-generation hardware through a collection of functions beginning with the words waveOut. The NAudio library encapsulates those functions in a WaveOut class that takes care of the Win32 interoperability and hides much of the messiness as well.

WaveOut requires a class that you provide that implements the IWaveProvider interface, which means the class defines a gettable property of type WaveFormat that (at the very least) indicates the sample rate and the number of channels. The class also defines a method named Read. The arguments to the Read method include a byte-array buffer that the class is required to fill with waveform data. With default settings, this Read method will be called 10 times a second. Fall behind a little in getting this buffer filled and you'll hear unaesthetic gaps in the sound and ugly static.

NAudio provides a couple of abstract classes that implement IWaveProvider and make things a little easier for common audio jobs. The WaveProvider16 class implements an abstract Read method that lets you fill the buffer with shorts rather than bytes, so you don't have to break the samples in half.

Figure 1 shows a simple SineWaveOscillator class that derives from WaveProvider16. The constructor allows specifying a sampling rate, but calls the base class constructor with a second argument indicating one channel for monaural sound.

SineWaveOscillator defines two properties named Frequency (of type double) and Amplitude (a short). The program maintains a field named phaseAngle that always ranges between 0 and 2π. For each sample, the phaseAngle is passed to the Math.Sin function, and then increased by a value called the phase angle increment, which is a simple calculation involving the frequency and the sampling rate.

(If you're going to be generating many waveforms simultaneously, you'll want to optimize processing speed by using integer arithmetic whenever possible, even to the extent of implementing a sine wave table as an array of shorts. But for simple uses of waveform audio, floating point calculations are fine.)

To use SineWaveOscillator in a program, you'll need a reference to the NAudio.dll library and a using directive:

```
using NAudio.Wave;
```

Here's some code that starts playing a sound.

```
WaveOut waveOut = new WaveOut();
SineWaveOscillator osc = new SineWaveOscillator(44100);
osc.Frequency = 440;
osc.Amplitude = 8192;
waveOut.Init(osc);
waveOut.Play();
```

Here the Frequency property is initialized to 440Hz. In musical circles, that's the A above middle C, and is often used as a pitch standard and for tuning purposes. Of course, as the sound is playing, the Frequency property can be changed. To turn off the sound, the Amplitude could be set to 0, but the SineWaveOscillator will continue receiving calls to the Play method. To stop those calls, call Stop on the WaveOut object. When you don't need the WaveOut object any more, you should call Dispose on it to properly release resources.

For simple uses of waveform audio, floating point calculations are fine.

Off Key

When I used SineWaveOscillator in my sample program, it didn't do what I wanted. I wanted a sound to accompany objects dragged around the window, and I wanted the frequency of that sound to be based on the distance of the object from the center. But as I moved my objects, the frequency transitions weren't smooth. I was getting a bumpy glissando (such as fingers are swept across the keys of a piano or the strings of a harp), whereas what I wanted was a smooth portamento (like a trombone or the opening clarinet of Gershwin's "Rhapsody in Blue").

The problem is that each call to the Play method from WaveOut causes an entire buffer to be filled based on the same frequency value. During the time that the Play method is filling the buffer, the frequency can't change in response to the user dragging the mouse because Play is executing on the user-interface thread.

So how bad is this problem, and how large are these buffers?

The WaveOut class in NAudio includes a DesiredLatency property that, by default, is set to 300 milliseconds. It also includes a NumberOfBuffers property set to 3. (Multiple buffers help throughput because the API can be reading a buffer while an application is filling another.) Hence, each buffer is equivalent to .1 second of samples.

The Frequency property is initialized to 440Hz. In musical circles, that's the A above middle C.

Through experimentation, I discovered that it is not possible to decrease the DesiredLatency significantly without causing audible gaps. It is possible to increase the number of buffers—be sure to select a value so that the buffer size in bytes is a multiple of 4—but this didn't seem to help significantly. It's also possible to have the Play method run on a secondary thread by passing the static method call WaveCallbackInfo.FunctionCallback to the WaveOut constructor, but that didn't help much either.

It soon became obvious that what I needed was an oscillator that itself performed the portamento while filling the buffer. Instead of SineWaveOscillator, I needed a PortamentoSineWaveOscillator.

PortamentoSineWaveOscillator

I wanted to make other changes as well. Human perception of frequency is logarithmic. The octave is defined as a doubling of frequency, and octaves are audibly similar across the spectrum. To the human nervous system, the difference between 100Hz and 200Hz is the same as the difference between 1000Hz and 2000Hz. In music, each octave comprises 12 audibly equal steps called semitones. Hence, the frequencies of these semitones increase sequentially by a multiplicative factor equal to the twelfth root of two.

I wanted my portamento to be logarithmic as well, so in PortamentoSineWaveOscillator I defined a new property named Pitch that calculates frequency like this:

```
Frequency = 440 * Math.Pow(2, (Pitch - 69) / 12)
```

This is a fairly standard formula that comes from conventions used in the Musical Instrument Digital Interface (MIDI), which I'll discuss in a future column. If you number all the notes of the piano from the bottom to the top where Middle C is assigned a Pitch value of 60, then the A above Middle C is 69, and the formula determines the frequency to be 440Hz. In MIDI these Pitch values are integers, but in the PortamentoSineWaveOscillator class, Pitch is a double, so gradations between notes are possible.

Human perception of frequency is logarithmic.

In PortamentoSineWaveOscillator, the Play method detects when Pitch has changed and then gradually changes the value used to calculate the frequency (and hence the phase angle increment) based on the remaining size of the buffer. The logic allows Pitch to change while the method is executing, but that will only happen if Play is executing on a secondary thread.

As the AudibleDragging program in the code download demonstrates, it worked! The program creates seven little blocks of different colors near the center of the window. When you grab them with the mouse, the program creates a WaveOut object using PortamentoSineWaveOscillator. As the object is dragged, the program simply determines a distance from the center of the window, and sets the pitch of the oscillator based on the following formula:

```
60 + 12 * distance / 200;
```

In other words, Middle C plus one octave for every 200 units in distance. AudibleDragging is a silly little program, of course, and it may convince you more than ever that applications should forever be silent. But the potential of generating custom sounds at runtime is simply too powerful to be rejected categorically.

You're not limited to single sine-wave oscillators.

Play On

Of course, you're not limited to single sine-wave oscillators. You can also derive a mixer from WaveProvider16, and use that to combine several oscillators. You can combine simple waveforms into more complex ones. The use of a Pitch property suggests an easy approach to specifying musical notes.

But if it's music and musical instruments you want your application to blast from the speakers, you'll be pleased to know that NAudio also includes classes that let you generate MIDI messages from your Windows Forms or WPF applications. I'll show you how to do that soon. ■

CHARLES PETZOLD *is a longtime contributing editor to* MSDN Magazine. *His most recent book is "The Annotated Turing: A Guided Tour through Alan Turing's Historic Paper on Computability and the Turing Machine" (Wiley, 2008). Petzold blogs on his Web site charlespetzold.com.*

THANKS to the following technical expert for reviewing this article: *Mark Heath*

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

FusionCharts | from $195.02

InfoSoft Global
empowering human thoughts

Interactive and animated charts for ASP and ASP.NET apps.

- Liven up your Web applications using animated Flash charts
- Create AJAX-enabled charts that can change at client-side without invoking server requests
- Export charts as images/PDF and data as CSV for use in reporting
- Also create gauges, financial charts, Gantt charts, funnel charts and over 550 maps
- Used by over 14,000 customers and 250,000 users in 110 countries

Janus WinForms Controls Suite | from $757.44

Janus systems

Add Outlook style interfaces to your WinForms applications.

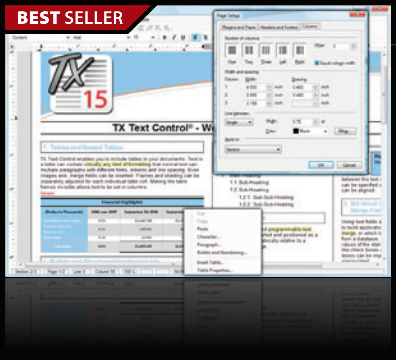
- Janus GridEX for .NET (Outlook style grid)
- Janus Schedule for .NET and Timeline for .NET (Outlook style calendar view and journal)
- Janus ButtonBar for .NET and ExplorerBar for .NET (Outlook style shortcut bars)
- Janus UI Bars and Ribbon Control (menus, toolbars, panels, tab controls and Office 2007 ribbon)
- Now includes Office 2007 visual style for all controls

ContourCube | from $900.00

Contour components

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



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

TX TEXT CONTROL word processing components

Word processing components for Visual Studio .NET.

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

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

US Headquarters	European Headquarters	Asia / Pacific Headquarters
ComponentSource	ComponentSource	ComponentSource
650 Claremore Prof Way	30 Greyfriars Road	3F Kojimachi Square Bldg
Suite 100	Reading	3-3 Kojimachi Chiyoda-ku
Woodstock	Berkshire	Tokyo
GA 30188-5188	RG1 1PE	Japan
USA	United Kingdom	102-0083

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

MasterCard VISA DISCOVER
GSA Schedule Contract GS-35F-0188R

Security Compliance as an Engineering Discipline

As a result of new initiatives and requirements like the Payment Card Industry Data Security Standard (PCI-DSS), many organizations are building comprehensive application security programs for the first time. To do so, a number of those concerns are looking to the proven success of the Microsoft Security Development Lifecycle (SDL). This can be a very smart business move, but it's important to understand how the engineering focus of the SDL makes it different from the typical security-compliance effort. This month I'm going to address some of the ways to harmonize compliance-focused programs with security engineering to improve your software development practices.

By integrating secure engineering practices into the entire software lifecycle, the SDL lets you achieve application security assurance using a holistic approach. The process requires direct engagement with software developers and testers, and unlike the typical compliance program, it's incremental, iterative and should be customized to every organization.

In this article, I'll focus on some of the strategies and best practices for deploying SDL and integrating it with security compliance regimes.

Compliance and the SDL

Building a software security program from scratch can seem daunting. Few organizations have a program or team dedicated specifically to secure application development. The portfolio of the typical security group includes networks and firewalls, policy and identity management, systems configuration, and general operations and monitoring. Securing the ever-growing set of custom-built, mission-critical applications is a new and challenging addition to these responsibilities.

While a mature software security practice will facilitate compliance efforts like PCI and the Common Criteria for Information Technology Security Evaluation, looking at the SDL as just another compliance effort will keep you from getting the best results from your program. Unlike compliance programs, which have externally defined goals and activities, the SDL recognizes that the business goals and abilities of attackers are constantly changing. Many organizations achieve compliance through large up-front efforts followed by minimal maintenance. In contrast, the SDL starts incrementally and focuses on using constant effort for continuous improvement. In addition, compliance often encourages separation of duties between implementation and enforcement. The SDL is built on close coordination with developers and testers. Given these differences, if your experience is only in managing compliance efforts, how should you approach an SDL implementation?

First, consider how your organization has managed other major software industry changes in the past decade, such as moving from procedural to object-oriented development, client-server to Web-based software, or waterfall to agile development methods. These changes weren't made across the entire organization with the stroke of a pen. They were introduced incrementally and grew organically. A similar approach, as outlined below, will work best for the SDL.

- Start small with one or two pilot projects.
- Even though you'll introduce the new way of doing things with a capable team, bring in outside help where needed.
- Study and learn from the results at the completion of a project. Adapt the standard practices to fit your particular organization. Get rid of the things that didn't work.
- Let coverage expand organically, nurturing internal experts and spreading the culture change at a pace the organization can manage without disrupting the business.

The SDL recognizes that the business goals and abilities of attackers are constantly changing.

Second, recognize that being in compliance is just one goal for your program—the first step, not the finish line. More-secure and higher-quality software provides its own business and competitive value. Measure your program by how successful you ultimately are at delivering more trustworthy products and what that means for your bottom line.

Adding Value with Secure Engineering

Often a team needs help optimizing its software security program because the project is in the early stages and the implementers require guidance and encouragement. But sometimes even established programs somehow fail to gain traction. Although the organization may have recognized the value of security and implemented a wide variety of activities and best practices—training,

Send your questions and comments for Brad to briefs@microsoft.com.

design review, static analysis, dynamic scanning, code review and manual penetration testing—the results end up reflecting poorly on the large investment made.

The compliance-oriented approaches of traditional IT security frequently bring with them biases that prevent these programs from being truly successful in changing software quality. By reorienting programs to avoid some common compliance pitfalls and by including the development organization, thus integrating engineering approaches into compliance efforts, organizations can often identify simple changes that help their security effort reach its full potential.

In the following sections I'll highlight four of the most common pitfalls and explain how to avoid or address them.

Don't Do Everything—Do a Few Things Well

Compliance efforts and their results tend to be well-defined. Individual requirements are compartmentalized and have standard measures, and overall compliance status is a matter of having enough of the right boxes checked. From the abstract goal of becoming compliant, the correct management strategy is to use the available resources to cover as many areas as possible, and to spend as little as necessary to become compliant with any individual requirement.

Quality software engineering is much less amenable to this sort of measurement, except at the very finest granularity. Spreading resources too thin is a common pitfall. What's worth doing is worth doing well. A cursory or unskilled penetration test may give a false sense of security—overconfidence in the capabilities of a code-scanning tool may lead to under-allocating resources to manual code review, and limited training in areas such as cryptography may give developers just enough knowledge to be dangerous. It's better to do a few things really well than everything poorly.

Consider one example from the code review process used in security effort my company consulted on. A security code review by a dedicated team was a mandatory part of the development process for all Internet-facing applications—a rarely seen practice and capability that's usually indicative of a very strong program.

Code review by humans can be highly effective at finding difficult-to-spot and important classes of security flaws, but what did this organization's process look like? Because it was mandatory, the reviewers had a great deal of work to do and little time to complete it. Almost no time was allocated for the development team to collaborate in the process, so the code reviewers were working with little business context. Additionally, they had no access to previous bug reports from tools or manual testing, nor did they have access to a live instance of the system against which to prove their findings.

What were the results? Findings from code review were delivered to the development team mere days before scheduled go-live dates, and false-positive rates were as high as 50 percent. It was rare that these bugs could even be triaged, let alone fixed, during the same release cycle in which they were found. Furthermore, identification of logic flaws—perhaps the most important benefit of manual analysis—was unattainable due to the lack of business context and collaboration.

A better approach—one unlikely to be heard of often in the world of compliance—is do less, and do it less often. Instead of devoting two days each to code review and black-box penetration testing on 10 releases a quarter, eliminate code review as a distinct activity. Instead, perform eight days of source-code-informed, white-box penetration testing on five releases per quarter. The reduced number of targets and broader context would allow enough time to find more flaws and to actually fix them before release.

Target Projects, Not Policies

Generally, when a compliance program is undertaken, its requirements are mandated for the entire organization and activities often are structured to have few dependencies. This usually means that compliance efforts proceed horizontally, with new policies made effective for everyone in an organization simultaneously. While some SDL activities are amenable to this approach, it makes much more sense to pick a few teams or projects to target with a vertical rollout.

To understand this, consider an analogy to another methodology change that many software organizations have undertaken recently: the use of agile development methods. No company approaches getting all its teams on agile methods by eliminating formal specifications for all its projects in Q1, increasing iteration speeds in Q3, and starting to write unit tests in Q2 of the following year. The practices of agile development enhance and support each other and should be rolled out together, one team and one project at a time.

Approach the SDL similarly: Don't exhaust your initial hard-won resources by implementing a universal mandate for threat modeling without follow-up that uses the threat models, measures their success, and iteratively improves. No SDL activity reaches its potential in isolation from the others, and there's no surer path to a security initiative's failure than a stack of expensive security documentation with no actionable next steps.

This isn't to say you should do everything at once. Pick a few activities to start with, such as threat modeling, code review, and automated scanning and roll them out together for a limited set of projects. This approach allows you to gather data on how accurate and useful your code scanner was in reducing vulnerabilities. It also lets you see what kinds of bugs the scanner missed, but that manual testing was able to find or that threat modeling was able to prevent. This is important information that helps you fine-tune each process, understand the assurance levels you're achieving, and guide the effective allocation of future resources.

Always Measure Effectiveness

Too often when it comes to compliance, the thinking is, to paraphrase Tennyson, "Ours is not to reason why. Ours is but to do and die." If a requirement is on the list, you won't be compliant without it. With such an iron-clad mandate, all incentive to measure the benefit of the activity is lost. So, for example, while most organizations have password rotation policies, few quantify how much security benefit such guidelines provide. Not many concerns attempt a cost-benefit analysis to see if changing passwords on 15- or 45-day cycles would be better than 30, because there's no reason to question the standard.

Avoid making this mistake when quality is your real goal. Typical training regimens provide a great example of this error. Often the only two measurements we see for evaluating instructional programs are: did everyone get security training and did the training mention the major vulnerability classes? If your business goal is more-secure software, though, have those two hard-earned (and probably expensive) checked boxes actually improved your software quality?

Measuring effectiveness comes back to the core idea of relating software security directly to business goals. Before you start any activity, understand what you're trying to achieve and find a way to measure whether it works. Are developers and testers retaining what they learn? Can they apply it to the real problems they have to solve? Are teams that have been trained producing software with fewer bugs or eliminating bugs earlier in the lifecycle than teams that haven't been trained?

While most organizations have password rotation policies, few quantify how much security benefit such guidelines provide.

Metrics don't have to be expensive to collect—a simple survey might be enough to tune a training course—but they have to be there. Developers are savvy enough to tell apart meaningful activities tuned to produce results and hoops to be jumped through. The difference between successful and unsuccessful security programs is often one of mistaking the hoops for the results.

Setting principled goals, measuring the program's progress against those goals, and improving the process at each step will help motivate developers and change the internal culture of quality. Showing progress against meaningful metrics also provides a good justification for the effort and will support the procurement of additional resources required for the maintenance and growth of the program.

Just Because Everyone Else Is Doing It ...

Just as there are no silver bullets for the general software-development process, there can be none for developing secure software. Be wary of anyone telling you they have the one universal solution for all your security problems. Start small, adopt practices that are appropriate to your organization, and learn from the experiences of others with similar security needs and resources.

But how do you know where to start? The list of things to do seems huge.

A recent survey of the world's best software security programs observed that, essentially, all of them performed the same set of nine activities. At last: best practices! Shouldn't you set as your target the procedures the top 10 programs in the world agree on?

If you look at the world's top 10 navies, you'll find that they all deploy aircraft carriers and submarines, and their admirals would be quick to tell you that these technologies are the most effective way to project force at sea. Of course, as you keep looking, you'll find that nobody *but* the top 10 navies has submarines and aircraft carriers. Such fleets require vast expenditures, support structures and technical capabilities that are only within the reach of a major power. Many smaller countries would find little use for equivalent resources and might very well be bankrupted trying to acquire and maintain such systems.

Similarly, a small software-development organization may well bankrupt itself following practices and trying to manage tools whose purpose is scaling secure engineering to thousands of developers managing many millions of lines of code.

The big-ticket, silver-bullet solutions for security compliance tend to be better at killing budgets and credibility than bugs—unless an organization is well-prepared for the introduction of such products. The most successful organizations aren't those that check all the big boxes, but those that start small and that grow sustainable and manageable security programs closely related to the goals of the business.

Before you buy that expensive software package, build your expertise with the growing arsenal of free and built-in security tools available on the major enterprise software platforms. Get comfortable triaging and fixing all the bugs FxCop and C++ /analyze (or FindBugs if your organization uses Java) can turn up, and get your code to compile cleanly against the SDL banned API list before you spend a penny doing anything more complex.

Once you're used to managing and running cleanly with these tools, you're much more likely to succeed with something more comprehensive. You may find that the free tools are just what you need, or they may leave you eager for bigger and better. You haven't lost anything by starting with free.

Harmonizing Quality and Compliance

While it may be starting to sound like the SDL is antithetical to compliance, that's definitely not the case. The two target the same basic needs, and harmonizing and integrating them can produce significant cost savings. When extending a compliance program to improve quality through the SDL, you can avoid common pitfalls and maximize your success with the four strategies discussed.

Don't try to do everything at once. Start with what you can do well. Increment your efforts project by project, not policy by policy. Always measure the effectiveness of your efforts for your own business bottom line. Grow into your program, don't buy into it.

Most importantly, get started now! There's no minimum increment for quality. Start by taking advantage of the free tools and guidance at the Microsoft SDL Web site (msdn.microsoft.com/security/cc448177). Download and try out the SDL Optimization Model to evaluate the current state of your organization's capabilities, and use the professional services and training available through the Microsoft SDL Pro Network (msdn.microsoft.com/security/dd219581) to start improving quality and delivering more secure software while you meet your audit and compliance goals. ■

BRAD HILL *is the Director of SDL Services at iSEC Partners, a full-service security consulting firm that provides penetration testing, secure systems development, security education and software design verification. Visit isecpartners.com for more information.*

THANKS to the following technical expert for reviewing this article: *Michael Howard*

Does your Team do more than just track bugs?

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

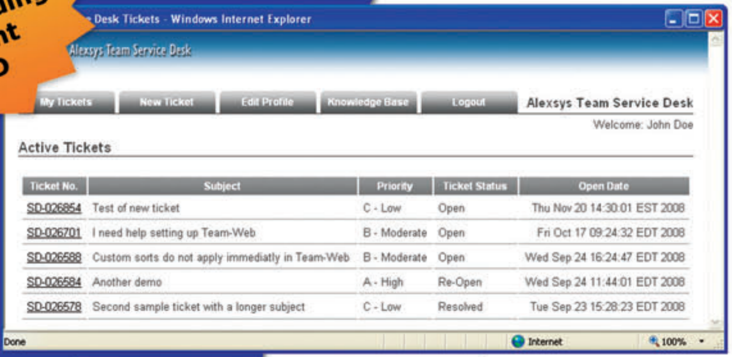
Alexsys Team® does! Alexsys Team 2 is a multi-user Team management system that provides a powerful yet easy way to manage all the members of your team and their tasks - including defect tracking. Use Team right out of the box or tailor it to your needs.

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

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

Native Smart Card Login Support including Government and DOD

Alexsys Team



Desk Tickets - Windows Internet Explorer

Alexsys Team Service Desk

My Tickets | New Ticket | Edit Profile | Knowledge Base | Logout

Alexsys Team Service Desk

Welcome: John Doe

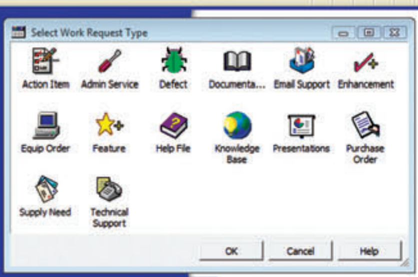
Active Tickets

Ticket No.	Subject	Priority	Ticket Status	Open Date
SD-026854	Test of new ticket	C - Low	Open	Thu Nov 20 14:30:01 EST 2008
SD-026701	I need help setting up Team-Web	B - Moderate	Open	Fri Oct 17 09:24:32 EDT 2008
SD-026588	Custom sorts do not apply immediatly in Team-Web	B - Moderate	Open	Wed Sep 24 16:24:47 EDT 2008
SD-026584	Another demo	A - High	Re-Open	Wed Sep 24 11:44:01 EDT 2008
SD-026578	Second sample ticket with a longer subject	C - Low	Resolved	Tue Sep 23 15:28:23 EDT 2008

Done | Internet | 100%

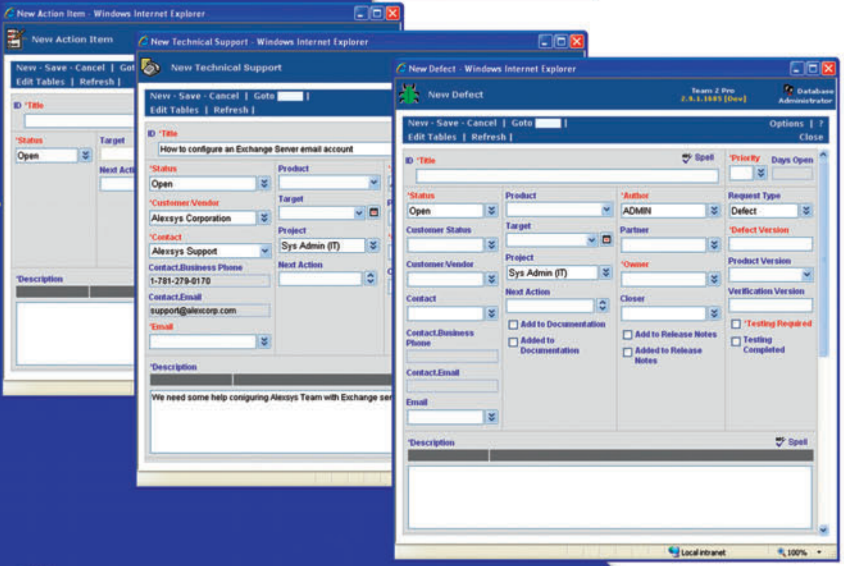
New in Team 2.11

- 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



New Action Item: Status: Open

New Technical Support: Title: How to configure an Exchange Server email account; Status: Open; Customer/Vendor: Alexsys Corporation; Contact: Alexsys Support; Contact.Business Phone: 1-781-279-0170; Contact.Email: support@alexcorp.com; Project: Sys Admin (IT); Description: We need some help coniguring Alexsys Team with Exchange ser

New Defect: Status: Open; Author: ADMIN; Request Type: Defect; Project: Sys Admin (IT)

Team 2 Features

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

ZDNet Editors' Pick

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

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

DR. JAMES MCCAFFREY

WCF Service Testing with Sockets

In this month's column I am joined by Carlos Figueira, a senior software development engineer in Test on the Windows Communication Foundation (WCF) team. With his help I intend to show you how to test WCF services using a network socket-based approach.

A good way for you to see where I'm headed is to examine the screenshots in **Figures 1, 2** and **3**. **Figure 1** shows a WinForm application that hosts a simple-but-representative WCF service named MathService. Behind the scenes, MathService contains a single operation named Sum that accepts two values of type double, then computes and returns their sum.

Figure 2 shows a typical WCF ASP.NET Web application client that accepts two values from the user, sends those two values to the MathService service, fetches the response from the service, and displays the result in a ListBox control.

WCF bindings include information about secutity and reliabilty settings, transport protocol, and encoding type.

Figure 3 shows a console application test harness that performs functional verification of the MathService service. The test harness sends SOAP messages directly to MathService using a network socket, accepts the response from the service, and compares an expected result with the actual result to determine a pass or fail. In test case #001, the test harness sends 3.4 and 5.7 to the WCF service under test and receives the expected value 9.1. Test case #002 is a deliberate, spurious failure just for demonstration purposes.

In the sections that follow, I first briefly describe the WCF service under test shown in **Figure 1** so you'll understand which factors are relevant when constructing WCF socket-based test automation. Next I briefly explain the demonstration Web client to give you some insight into when socket-based testing is more appropriate than alternative techniques. Then I explain in detail the code that created the test harness so that you will be able to adapt the technique I present here to meet your own needs. This article assumes you have intermediate-level C# coding skills.

The WCF Service Under Test

I used Visual Studio 2008 to create the WCF service under test. One of the neat features about WCF services is that they can be hosted in many different types of applications. I decided to host the WCF MathService service in a WinForm application, but the techniques I present in this article work with any type of WCF host.

After creating an empty WinForm, I added two Button controls and one ListBox control. Then, just below the Form definition, I added the simple code required to declare a WCF service:

```
[ServiceContract]
public interface IMathService {
[OperationContract]
  double Sum(double x, double y);
}
```

The ServiceContract attribute applied to an interface generates all the code needed for a WCF inteface. If you're moving to WCF from Web services, you can think of an OperationContract attribute as being analogous to the WebMethod attribute. Implementing the WCF service is simple:

```
public class MathService : IMathService  {
  public double Sum(double x, double y) {
    double answer = x + y;
    return answer;
  }
}
```

With my WCF plumbing in place, I added a reference to System.ServiceModel.dll, the .NET assembly that houses WCF functionality, to my project. Then I added using statements to the two key .NET namespaces contained within the assembly needed for my WCF service:

```
using System.ServiceModel;
using System.ServiceModel.Description;
```

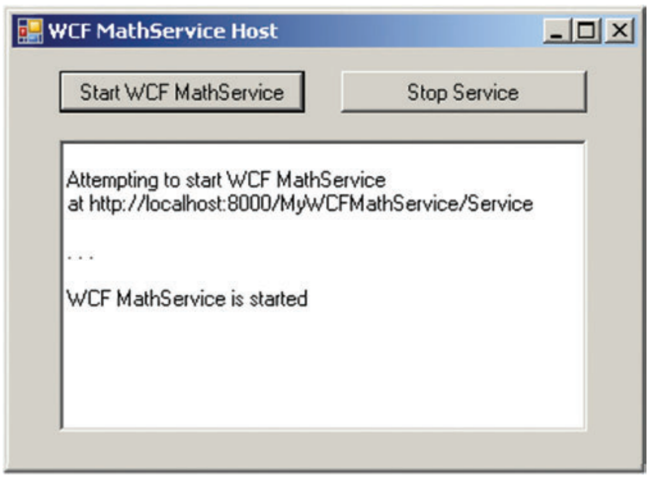
The ServiceModel namespace holds the ServiceHost class and several classes that define WCF bindings. The Description namespace holds the ServiceMetadataBehavior class I use to publish information about my WCF service. Next I added service instantiation logic to the Button1 control event handler:

```
try {
  string address =
    "http://localhost:8000/MyWCFMathService/Service";
  Uri baseAddress = new Uri(address);
  serviceHost =
    new ServiceHost(typeof(MathService), baseAddress);
  serviceHost.AddServiceEndpoint(typeof(IMathService),
    new WSHttpBinding(SecurityMode.None), "MathService");
  . . .
```

The key factor to note here is that I create a WCF endpoint using WSHttpBinding. WCF bindings are a collection that includes information about secutity and reliabilty settings, transport protocol and encoding type. The WSHttpBinding is an excellent general purpose

Send your questions for Dr. McCaffrey to testrun@microsoft.com.

Code download available at code.msdn.microsoft.com/mag201002WCF.



```
Attempting to start WCF MathService
at http://localhost:8000/MyWCFMathService/Service

. . .

WCF MathService is started
```

Figure 1 **WCF MathService Service Under Test**

binding for non-duplex communication. By default, WSHttpBinding uses encrypted transmission, but here I specify SecurityMode.None so that you can more easily see request-response data.

Next I add code to make my service visible to clients through the Add Service Reference mechanism:

```
ServiceMetadataBehavior smb =
  new ServiceMetadataBehavior();
smb.HttpGetEnabled = true;
serviceHost.Description.Behaviors.Add(smb);
. . .
```

Now I'm ready to add code that starts up the WCF service:

```
serviceHost.Open();
int count = 0;
while (serviceHost.State !=
       CommunicationState.Opened &&
       count < 50) {
  System.Threading.Thread.Sleep(100);
  ++count;
}
. . .
```

I use a simple but effective way to detect a service hang at startup by going into a delay loop of 0.1 seconds per delay, up to a maximum of 50 delays. After the delay loop terminates, either the WCF service has started or the maximum number of delays has been reached:

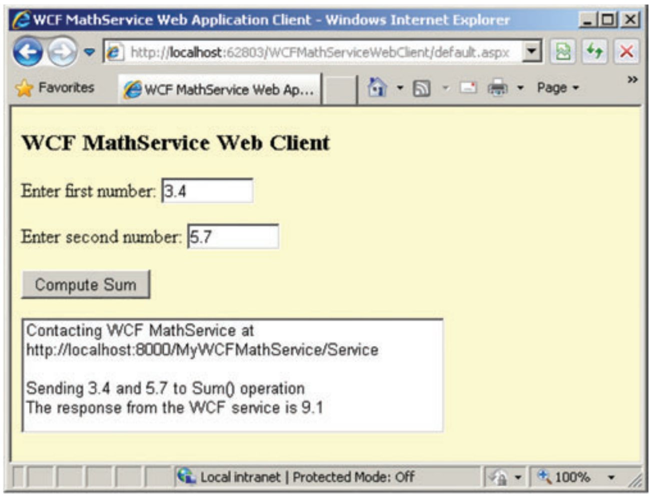
```
if (serviceHost.State == CommunicationState.Opened)
  listBox1.Items.Add("WCF MathService is started");
else
  throw new Exception(
    "Unable to start WCF MathService in a timely manner");
```

I'm taking many shortcuts you wouldn't take when writing production code, such as not checking to see if the serviceHost object is already open before attempting to open it. The logic in the Button2 control event handler closes the WCF MathService, and I use that same pattern to start the service:

```
int count = 0;
serviceHost.Close();
while (serviceHost.State !=
       CommunicationState.Closed &&
       count < 50) {
  System.Threading.Thread.Sleep(100);
  ++count;
}
```

A Typical Web Application Client

I used Visual Studio 2008 to create the WCF Web application client shown in **Figure 2**. I started by clicking File | New | Web


WCF MathService Web Application Client - Windows Internet Explorer
http://localhost:62803/WCFMathServiceWebClient/default.aspx
Favorites
WCF MathService Web Ap...
Page
Local intranet | Protected Mode: Off
100%

WCF MathService Web Client

Enter first number: 3.4

Enter second number: 5.7

Compute Sum

```
Contacting WCF MathService at
http://localhost:8000/MyWCFMathService/Service

Sending 3.4 and 5.7 to Sum() operation
The response from the WCF service is 9.1
```

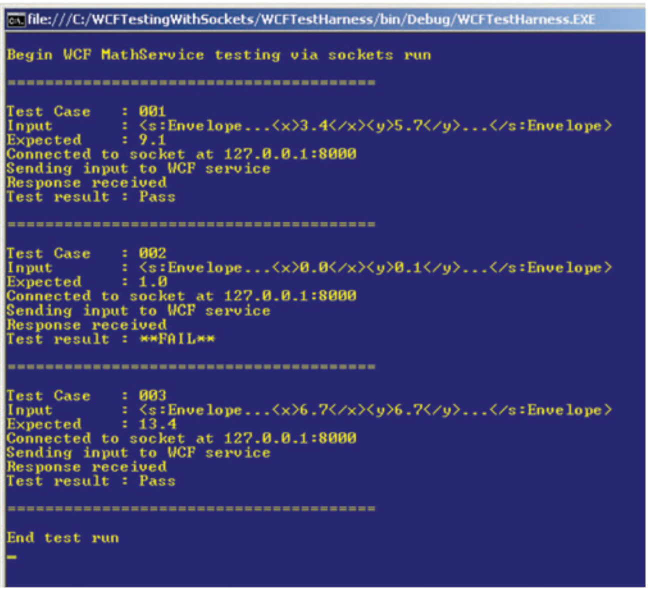
Figure 2 **Typical WCF ASP.NET Client**

Site. In the new Web site dialog box, I targeted the Microsoft .NET Framework 3.5, selected the Empty Web Site template, chose a File System location using the C# language, and named my project WCFMathServiceWebClient.

Next, in Solution Explorer, I right-clicked on my project and selected the Add New Item option from the context menu. In the resulting dialog box, I selected the Web Form item. I deselected the "Place code in separate file" option so I could place all my application code in a single file.

After adding the Web Form, I added server-side control tags to create the very simple UI of the Web application:

```
<asp:Label ID="Label1" runat="server"
  Text="Enter first number: "/>
<asp:TextBox ID="TextBox1" runat="server" /><p />
<asp:Label ID="Label2" runat="server"
  Text="Enter second number: "/>
<asp:TextBox ID="TextBox2" runat="server" /><p />
<asp:Button ID="Button1" runat="server" Text="Compute Sum"
  onclick="Button1_Click" /><p />
<asp:ListBox ID="ListBox1" runat="server" />
```


file:///C:/WCFTestingWithSockets/WCFTestHarness/bin/Debug/WCFTestHarness.EXE

```
Begin WCF MathService testing via sockets run

=====================================

Test Case   : 001
Input       : <s:Envelope...<x>3.4</x><y>5.7</y>...</s:Envelope>
Expected    : 9.1
Connected to socket at 127.0.0.1:8000
Sending input to WCF service
Response received
Test result : Pass

=====================================

Test Case   : 002
Input       : <s:Envelope...<x>0.0</x><y>0.1</y>...</s:Envelope>
Expected    : 1.0
Connected to socket at 127.0.0.1:8000
Sending input to WCF service
Response received
Test result : **FAIL**

=====================================

Test Case   : 003
Input       : <s:Envelope...<x>6.7</x><y>6.7</y>...</s:Envelope>
Expected    : 13.4
Connected to socket at 127.0.0.1:8000
Sending input to WCF service
Response received
Test result : Pass

=====================================

End test run
-
```

Figure 3 **WCF Test Harness Run**

Next, I hit the F5 key to instruct Visual Studio to build the application and prompt me to allow the automatic generation of a Web.config file for my project. After I OK'd the creation of the Web.config file, I started the WCF MathService service so my client project would be able to see the service.

I right-clicked on the client project in Solution Explorer and selected the Add Service Reference option from the context menu. In the Add Service Reference dialog box, I entered the location of my WCF service (http://localhost:8000/MyWCFMathService/ Service) and clicked the Go button. Because my WCF service is running and the service publishes metadata about itself, the Visual Studio tool will find the service. I renamed the service namespace from the default ServiceReference1 to WCFMathServiceReference.

Behind the scenes, Visual Studio adds information about the WCF service to the Web.config file by creating a <system.serviceModel> entry. That entry contains an <endpoint> element with attribute binding="wsHttpBinding" so that the Web application client knows how to communicate with the WCF service. In design view, I double-clicked on the Button1 control to register its event handler and added logic to access the service:

```
try {
  WCFMathServiceReference.MathServiceClient sc =
    new WCFMathServiceReference.MathServiceClient();
  double x = double.Parse(TextBox1.Text);
  double y = double.Parse(TextBox2.Text);
  double sum = sc.Sum(x, y);
  . . .
  ListBox1.Items.Add(
    "The response from the WCF service is " + sum);
```

The name of the class that holds the Sum operation is MathServiceClient—in other words, the name of the class (MathService) derived from the WCF contract interface (IMathService), with "Client" appended.

The Test Harness

I also used Visual Studio 2008 to create a C# console application project named WCFTestHarness. At the very top of the Visual Studio-generated Program.cs file I added these using statements:

```
using System;
using System.Text;
using System.Net;
using System.Net.Sockets;
```

I need classes in the System.Text namespace to convert text to byte arrays because TCP works at the byte level. I need classes in the System.Net namespace to create objects that are abstractions of IP addresses. And I need the System.Net.Sockets namespace to create a socket object that performs the actual send and receive operations. I added a brief logging message to the Main method and then set up my test case data as an array of strings:

```
namespace WCFTestHarness {
  class Program {
    static void Main(string[] args) {
      try {
        Console.WriteLine(
        "\nBegin WCF MathService testing via sockets run");

        string[] testCases = new string[] {
          "001,3.4,5.7,9.1",
          "002,0.0,0.1,1.0",
          "003,6.7,6.7,13.4"
        };
        . . .
```

Each test case data string contains four comma-delimited fields: a test case ID, two inputs to the Sum operation, and an expected response value. I use my test case data to create the main test harness processing loop:

```
foreach (string testCase in testCases) {
  Console.WriteLine("\n============\n");
  string[] tokens = testCase.Split(',');
  string caseID = tokens[0];
  string input1 = tokens[1];
  string input2 = tokens[2];
  string expected = tokens[3];
  . . .
```

I use the Split method to break each string into its four fields.

Next comes one of the key parts of sending input to a WCF service using sockets. I create a SOAP message as shown in **Figure 4**.

Notice that my test case input values, input1 and input2, are embedded into the SOAP message inside a <Sum> element near the end of the message. But how did I determine this not-so-simple message structure?

There are several ways you can determine the required SOAP message structure and data for a WCF service. The easiest approach, and the one I used, is to use a network traffic examination tool such as netmon or Fiddler to capture data while you exercise a client application. In other words, with my WCF service running, and a traffic capture tool also running (I used Fiddler), I launched the Web application client program shown in **Figure 2** and used the client to send a request to the WCF service. The network traffic capture tool showed me the SOAP message that was sent from the client to the WCF service. I used that information to craft the SOAP message in my test harness.

With my SOAP message constructed, next I displayed my test case input as expected data to the shell:

```
Console.WriteLine(
  "Test Case   : " + caseID);
Console.WriteLine(
  "Input       : <s:Envelope..." + "<x>" +
  input1 + "</x>" + "<y>" + input2 +
  "</y>...</s:Envelope>");
Console.WriteLine("Expected    : " + expected);
```

Then I set up the IP address of the target WCF MathService service under test:

```
string host = "localhost";
IPHostEntry iphe = Dns.Resolve(host);
IPAddress[] addList = iphe.AddressList;
EndPoint ep = new IPEndPoint(addList[0], 8000);
```

The Dns.Resolve method returns a list of IP addresses (there may be more than one) associated with a particular host machine name as an IPHostEntry object. The IPHostEntry object has an AddressList property that is an array of IPAddress objects. An EndPoint

Figure 4 **SOAP Message for testing the WCF service**

```
string soapMessage = "<s:Envelope xmlns:s='http://www.w3.org/2003/05/soap-envelope'";
soapMessage += " xmlns:a='http://www.w3.org/2005/08/addressing'><s:Header>";
soapMessage += "<a:Action s:mustUnderstand='1'>http://tempuri.org/IMathService/Sum</a:Action>";
soapMessage += "<a:MessageID>urn:uuid:510b1790-0b89-4c85-8015-d1043ffeea14</a:MessageID>";
soapMessage += "<a:ReplyTo><a:Address>http://www.w3.org/2005/08/addressing/anonymous</a:Address>";
soapMessage += "</a:ReplyTo><a:To s:mustUnderstand='1'>";
soapMessage += "http://localhost:8000/MyWCFMathService/Service/MathService</a:To></s:Header>";
soapMessage += "<s:Body><Sum xmlns='http://tempuri.org/'>";

soapMessage += "<x>" + input1 + "</x>" + "<y>" + input2 + "</y>";

soapMessage += "</Sum></s:Body></s:Envelope>";
```

object consistes of an IPAddress object plus a port number. Now I can create my socket:

```
Socket socket = new Socket(AddressFamily.InterNetwork,
  SocketType.Stream, ProtocolType.Tcp);
socket.Connect(ep);
if (socket.Connected)
  Console.WriteLine(
    "Connected to socket at " + ep.ToString());
else
  Console.WriteLine(
    "Error: Unable to connect to " + ep.ToString());
```

Socket objects implement the Berkeley sockets interface, which is an abstraction mechanism for sending and receiving network traffic. The first argument to the Socket constructor specifies the addressing scheme. Here I use InterNetwork, which is ordinary IPv4.

The second argument to the Socket constructor specifies which of six possible socket types you want to create. Stream indicates a TCP socket. The other common type is Dgram, which is used for UDP (User Datagram Protocol) sockets.

The third argument to the Socket constructor specifies which protocols are supported by the socket. The Socket.Connect method accepts an EndPoint object.

Next I construct my header information:

```
string header =
  "POST /MyWCFMathService/Service/MathService HTTP/1.1\r\n";
header += "Content-Type: application/soap+xml; charset=utf-8\r\n";
header += "SOAPAction: 'http://tempuri.org/IMathService/Sum'\r\n";
header += "Host: localhost:8000\r\n";
header += "Content-Length: " + soapMessage.Length + "\r\n";
header += "Expect: 100-continue\r\n";
//header += "Connection: Keep-Alive\r\n\r\n";
header += "Connection: Close\r\n\r\n";
```

Just as with the SOAP message, I determined the header information by using a network traffic monitoring tool. Notice that each line ends with a \r\n (carriage return, linefeed) terminator rather than a single \n (newline) token, and the last line ends with a double \r\n. As I've indicated by the commented line of code above, depending on a number of factors you may need to send either a Keep-Alive or a Close argument to the Connection header entry. Similarly, the SOAPAction header is not necessary for a WSHttpBinding.

Now I can combine header and SOAP message, convert the entire message to bytes, and send the request:

```
string sendAsString = header + soapMessage;
byte[] sendAsBytes = Encoding.UTF8.GetBytes(sendAsString);
Console.WriteLine("Sending input to WCF service");
int numBytesSent = socket.Send(sendAsBytes, sendAsBytes.Length,
  SocketFlags.None);
```

The Socket.Send method has several overloads. Here I send the entire request without any special send or receive options. I do not make use of the return value, but I could have used that value to check that my entire message was sent. Now I can fetch the response from the WCF service:

```
byte[] receivedBufferAsBytes = new byte[512];
string receiveAsString = "";
string entireReceive = "";
int numBytesReceived = 0;

while ((numBytesReceived =
  socket.Receive(receivedBufferAsBytes, 512,
  SocketFlags.None)) > 0) {
  receiveAsString =
    Encoding.UTF8.GetString(receivedBufferAsBytes, 0,
    numBytesReceived);
  entireReceive += receiveAsString;
}
```

Because in most cases you cannot predict the number of bytes that will be in the response, the idea is to create a buffer and read chunks of the response until the entire response has been consumed. Here I use a buffer of size 512. As each group of 512 bytes are received, they are converted into text and appended to an aggregate result string.

With the response received, I check the response to see if it contains the current test case expected value:

```
Console.WriteLine("Response received");
if (entireReceive.IndexOf(expected) >= 0)
  Console.WriteLine("Test result : Pass");
else
  Console.WriteLine("Test result : **FAIL**");
```

There are several ways you can determine the required SOAP message structure and data for a WCF service.

The approach I use here is effective for very simple test scenarios, but you may have to add additional logic if your testing scenario is more complicated. I finish my harness by tying up loose ends:

```
  . . .
        } // main loop
        Console.WriteLine(
          "\n=====================================");
        Console.WriteLine("\nEnd test run");
      } // try
      catch (Exception ex)
      {
        Console.WriteLine("Fatal: " + ex.Message);
      }
    } // Main()
  } // Program
} // ns
```

Wrapping Up

There are many alternatives to WCF testing, but the socket-based approach to testing WCF services is extremely flexible. Because TCP and sockets are low-level constructions, they work in a variety of scenarios, in particular when you are testing in a technologically heterogeneous environment. For example, you could test a WCF service hosted on a Windows platform from a non-Windows client. Although you would have to modify the specific C# code I've presented here to a language supported by the client (typically C++ or Perl), the overall technique would be the same.

Additionally, a socket-based approach is quite useful for security testing (you must assume any WCF service will be subject to non-friendly socket probes) and performance testing (a socket approach is direct and can provide baseline round-trip timing data). ■

DR. JAMES MCCAFFREY *works for Volt Information Sciences Inc., where he manages technical training for software engineers working at the Microsoft Redmond, Wash., campus. He has worked on several Microsoft products including Internet Explorer and Search. Dr. McCaffrey is the author of ".NET Test Automation Recipes" (Apress, 2006) and can be reached at jammc@microsoft.com.*

THANKS to the following technical expert for reviewing this article:
Carlos Figueira

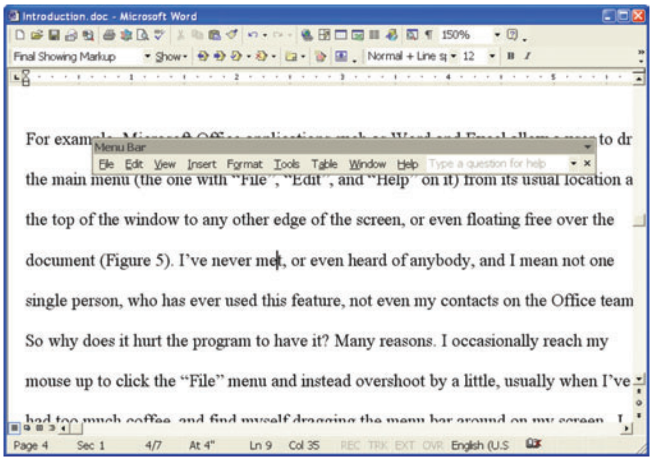
The Human Touch

It understands; it respects; it even enhances the humanity of the user. This is what computer software really can be and should be.

The screenshot below shows the movable, dockable main menu bar, a so-called "feature" that appeared in Microsoft Office Word 97 and departed with the menu in Office Word 2007. Have you ever moved the main menu because you *wanted* to? Have you ever seen, or even heard of, anyone doing that? You haven't, and neither have I. We've only dislodged it by accident, reaching for the File menu and overshooting by a pixel or two, usually in the morning when we've had too much coffee. We had to break our train of thought, re-dock the bar, then spend 30 seconds cursing the programmers who wrote that behavior. It might not sound like much, but figure 30 seconds, twice a day, times a billion users, and Word wastes 27 human lifespans every single day on this foolishness. Don't get me started ...

This feature is bad because it requires the user to become more precise in his mouse clicking, and punishes him if he doesn't. It disrespects and denies the fundamental humanity of the user. Computers are diligent; they are thorough; they are precise. Humans are none of these. That's why we invented computers: to be these things that we are not. This feature demands that users become less human and more like a computer. It's impolite, counterproductive, and philosophically wrong. It's not a feature: Because our goal is to make users happy so they pay us money, it's a bug.

One might reasonably argue that, given the adolescent age of the PC software industry at the time, nobody knew how wrong this feature would be until they had tried it; thus, it represents a necessary step in the evolution of software. And nobody died from it, (leastways not that I know of). But it's time we took what we learned from treating users wrong, and use that knowledge to treat them right.


Introduction.doc - Microsoft Word
150%
Final Showing Markup
Show
Normal + Line sp
12
Menu Bar
File Edit View Insert Format Tools Table Window Help
Type a question for help
Page 4 Sec 1 4/7 At 4" Ln 9 Col 35 REC TRK EXT OVR English (U.S

For exam[illegible] to dr
the main menu (the one with "File", "Edit", and "Help" on it) from its usual location a
the top of the window to any other edge of the screen, or even floating free over the
document (Figure 5). I've never met, or even heard of anybody, and I mean not one
single person, who has ever used this feature, not even my contacts on the Office team
So why does it hurt the program to have it? Many reasons. I occasionally reach my
mouse up to click the "File" menu and instead overshoot by a little, usually when I've
had too much coffee, and find myself dragging the menu bar around on my screen. I

The Moveable Menu Bar: Bad Idea

We can see this evolution in Word's auto-correct feature, an absolute gem. I think much faster than I type. (I don't think all that fast, I just type really slowly, for a geek.) In my haste to get my words into Word before they flee my brain, my one hand sometimes moves faster than the other and I type "hte" when I really mean "the."

Word does not scream an alarm to demand that I stop and fix my error, nor does it pop a box into my face saying "Didn't you really mean 'the'?" It doesn't even underline it with a red squiggle for me to come back to later. Instead, Word automatically fixes my typo, magically converting my "hte" into the "the" that I really meant. Word says with its actions, "No problemo Plattski, you sleep-deprived, hyper-caffeinated human being; I know what you meant. I'll take care of this silliness for you, you just keep ranting." And it does the same with every misspelling and other typo in its large and increasing memory.

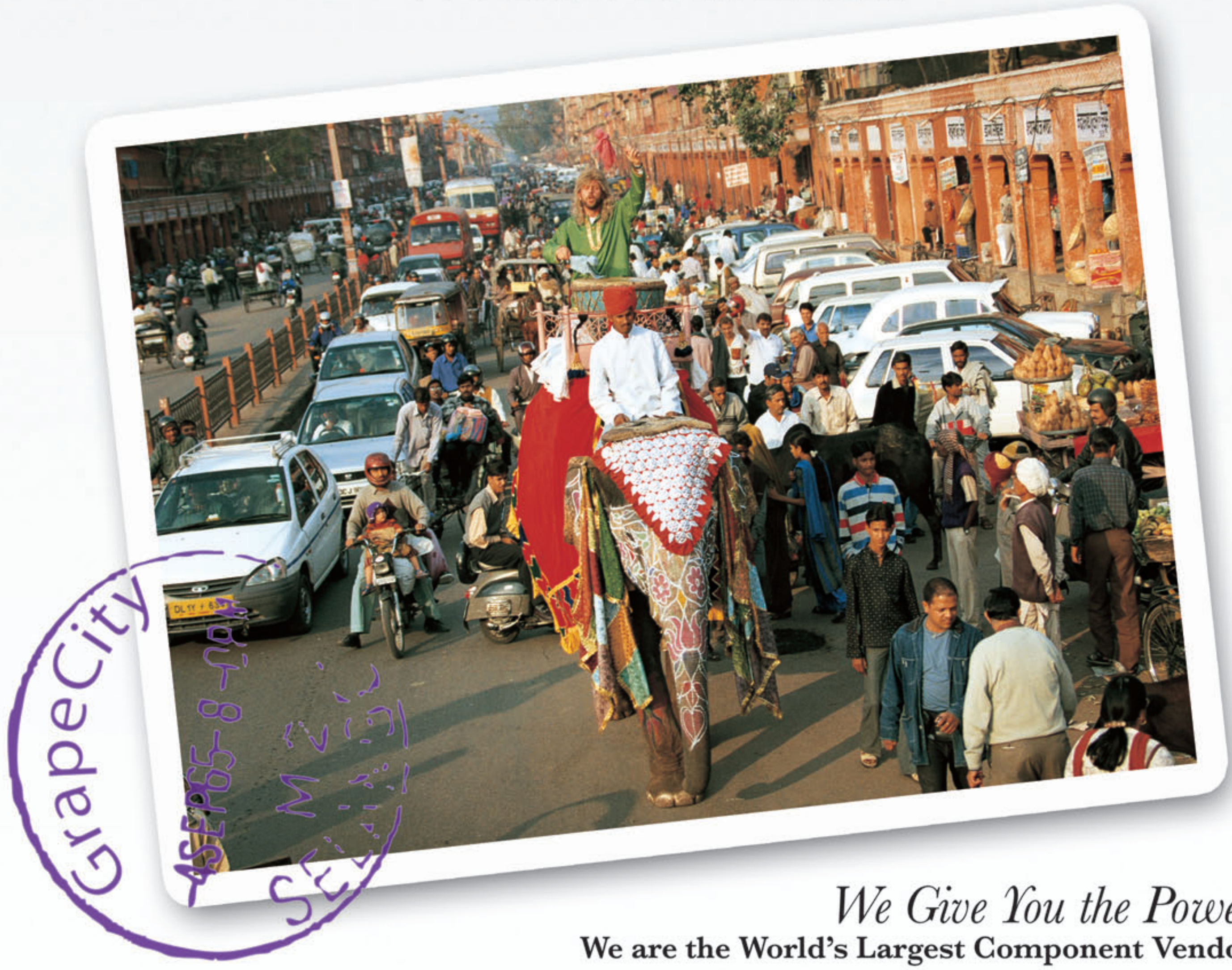
This feature uses the computer to do what computers do best, so that the human user can do what humans do best. It understands; it respects; it even enhances the humanity of the user. This is what computer software really can be and should be.

Think about it: two features in the same program. One demands that humans become more like computers, the other helps them be even more human. Which would you rather use? (OK, I know, you're a geek, that's why you're reading this magazine, but which would your paying customers rather use?) Humans are not going to stop being human any time soon, no matter how much you might wish they would evolve into something more logical. Good applications recognize this, and adjust to their human users, instead of hoping, futilely, for the opposite. ■

DAVID S. PLATT *teaches Programming .NET at Harvard University Extension School and at companies all over the world. He's the author of 11 programming books, including "Why Software Sucks" and "Introducing Microsoft .NET." Microsoft named him a Software Legend in 2002. He wonders whether he should tape down two of his daughter's fingers so she learns how to count in octal. You can contact him at rollthunder.com.*

GrapeCity

We Are Different.



We Give You the Power.

We are the World's Largest Component Vendor.

PowerTools

	SPREAD 5	ACTIVEREPORTS 6	ANALYSIS 1
Deep Functionality	100%	100%	100%
Award Winning	✓	✓	New
Category Leader	✓	✓	✓
Industry's Best Support	✓	✓	✓
Proven Track Record	✓	✓	✓

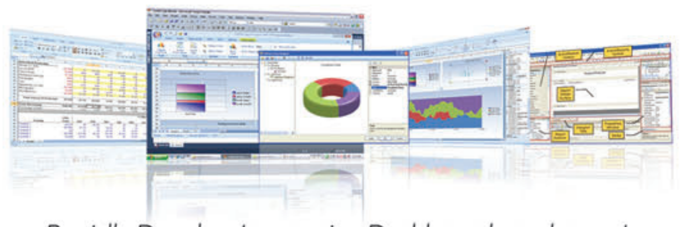
www.FarPointSpread.com & www.DataDynamics.com

The power to Analyze. The power to Report. The power to Excel.

The power to build better Applications.

Rapidly develop Business Intelligence Solutions using world-class Spreadsheets, Reporting and Analysis components.

Learn how to power up your applications today at FarPointSpread.com/BI



Rapidly Develop Impressive Dashboards and more!

Spreadsheets | Reporting | Analysis

Dundas
Data Visualization

The Evolution of Dashboarding

From the **industry leader** in data visualization technology comes an easy-to-integrate, customizable, turnkey **dashboard solution.**

- Rapid dashboard development
- Flexible integration and customization
- The latest Silverlight 3.0 technology

Dundas Dashboard

Powered by Microsoft Silverlight

For more information please visit
www.dundas.com/dashboard
(416) 467-5100 • (800) 463-1492