THE MICROSOFT JOURNAL FOR DEVELOPERS

JANUARY 2011 VOL 26 NO 1

msdn magazine

Microsoft

DESIGN DEVELOP EXPERIENCE

Quince

Using Quince™, you and your team can collaborate on the user interface using wireframes, designs and examples.

NetAdvantage®
for Silverlight Data Visualization
for WPF Data Visualization

Then use NetAdvantage® UI controls, like the map control used here, to bring the application to life quickly & easily.

NetAdvantage® ULTIMATE

for ASP.NET, Windows Forms, WPF, Silverlight, WPF Data Visualization, Silverlight Data Visualization

From start to finish, Infragistics gives you the tools to create impressive user experiences that'll make end users happy!

SEE HOW WE USE THE TOOLS TO CREATE THIS KILLER APP AT **INFRAGISTICS.COM/IMPRESS**

Infragistics®

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

Instantly Search Terabytes of Text

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

"Covers all data sources ... powerful Web-based engines"
eWEEK

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

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

Highlights hits in a wide range of data, using dtSearch's *own* file parsers and converters

- Supports MS Office through 2010 (Word, Excel, PowerPoint, Access), OpenOffice, ZIP, HTML, XML/XSL, PDF and more
- Supports Exchange, Outlook, Thunderbird and other popular email types, including nested and ZIP attachments
- Spider supports static and dynamic web data like ASP.NET, MS SharePoint, CMS, PHP, etc.
- API for SQL-type data, including BLOB data

25+ full-text & fielded data search options

- Federated searching
- Special forensics search options
- Advanced data classification objects

APIs for C++, Java and .NET through 4.x

- Native 64-bit and 32-bit Win / Linux APIs; .NET Spider API
- Content extraction only licenses available

Desktop with Spider	Web with Spider
Network with Spider	Engine for Win & .NET
Publish (portable media)	Engine for Linux

Ask about fully-functional evaluations!

The Smart Choice for Text Retrieval® since 1991

www.dtSearch.com • 1-800-IT-FINDS

msdn® magazine

JANUARY 2011 VOLUME 26 NUMBER 1

LUCINDA ROWLEY Director
DIEGO DAGUM Editorial Director/mmeditor@microsoft.com
KERI GRASSL Site Manager

KEITH WARD Editor in Chief/mmeditor@microsoft.com
TERRENCE DORSEY Technical Editor
DAVID RAMEL Features Editor
WENDY GONCHAR Managing Editor
KATRINA CARRASCO Associate Managing Editor

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

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

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

Abraham M. Langer Senior Vice President, Audience Development & Digital Media
Christopher M. Coates Vice President, Finance & Administration
Erik A. Lindgren Vice President, Information Technology & Application Development
Carmel McDonagh Vice President, Attendee Marketing
David F. Myers Vice President, Event Operations

Jeffrey S. Klein Chairman of the Board

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

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

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

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

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

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

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

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

Microsoft

BPA WORLDWIDE

Printed in the USA

Your best source for software development tools!

programmer's paradise®

LEADTOOLS Document Imaging SDK v17.0

by LEAD Technologies

LEADTOOLS Document Imaging has every component you need to develop powerful image-enabled business applications including specialized bi-tonal image processing, document clean up, annotations, high-speed scanning, advanced compression (CCITT G3/G4, JBIG2, MRC, ABC), and Win32/64 binaries for C/C++, .NET, Silverlight, WPF, WCF, & WF. Available add-ons include:

- Multi-threaded OCR/ICR/OMR/MICR/Barcodes (1D/2D)
- Forms Recognition/Processing
- Print Capture and Document Writers
- PDF, PDF/A and XPS

Paradise #
L05 03301A01
$2,007.99

programmers.com/LEAD

IDM UltraEdit

The #1 Best Selling Text Editor in the World

by IDM

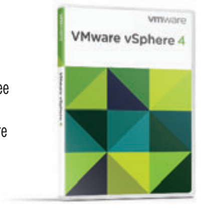
UltraEdit is the world's standard in text editors. Millions use UltraEdit as the ideal text/hex/programmers editor on any platform — Windows, Linux, or Mac!

Features include syntax highlighting for nearly any programming language; powerful Find, Replace, Find in Files, and Replace in Files; FTP support, sort, column mode, hex, macros/scripting, large file handling (4+ GB), projects, templates, Unicode, and more.

Named User
1-24 Users
Paradise #
I84 01201A01
$59.95

programmers.com/idm

VMware vSphere Essentials Kit Bundle



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

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

for 3 hosts
Paradise #
V55 85101C02
$446.99

programmers.com/vSphere

NEW RELEASE!

Embarcadero RAD Studio XE

by Embarcadero

Embarcadero RAD Studio XE is a comprehensive application development suite and the fastest way to visually build GUI-intensive, data-driven applications for Windows, .NET, PHP and the Web. RAD Studio includes Delphi®, C++Builder®, Delphi Prism™, and RadPHP™; providing powerful compiled, managed and dynamic language support, heterogeneous database connectivity, rich visual component frameworks, and a vast 3rd party ecosystem – enabling you to deliver applications up to 5x faster across multiple Windows, Web, and database platforms.

Paradise #
CGI 15401A01
$1,383.99

programmers.com/embarcadero

Enterprise Architect Corporate Edition

Visualize, Document and Control Your Software Project

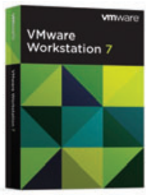
by Sparx Systems

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.

1-4 Licenses
Paradise #
SP6 03101A02
$182.99

programmers.com/sparxsystems

VMware Workstation 7



VMware Workstation 7 is the gold-standard virtualization software for desktop and laptop computers, with the richest feature set and broadest platform support available. VMware Workstation enables users to create and host multiple virtual machines on a single desktop, expanding the power of desktop systems for IT administrators; software development and test engineers; technical sales, training and support staff; and PC enthusiasts.

VMware Workstation transforms the way technical professionals develop, test, demo, and deploy software. Workstation's innovative features for VMware environments help to reduce hardware cost, save time, minimize risk, and streamline tasks that save time and improve productivity.

for Linux & Windows
Paradise #
V55 22301A04
$149.99

programmers.com/vmware

NEW VERSION 6!

ActiveReports 6

by GrapeCity

The de facto standard reporting tool for Microsoft Visual Studio.NET

- Fast and Flexible reporting engine
- Flexible event-driven API to completely control the rendering of reports
- Wide range of Export and Preview formats including Windows Forms Viewer, Web Viewer, Adobe Flash and PDF
- XCopy deployment
- Royalty-Free Licensing for Web and Windows applications

Professional Ed.
Paradise #
D03 04301A01
$1,310.99

programmers.com/grapecity

Microsoft Visual Studio Professional 2010 Upgrade

by Microsoft

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

with MSDN
Single user
Paradise #
M47 40201B02
$488.99

programmers.com/microsoft

TX Text Control 15.1

Word Processing Components

New Service Pack!

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
$1,220.99

Download a demo today.

programmers.com/textcontrol

NEW RELEASE!

New Intel Visual Fortran Compiler

by Intel

Intel® Visual Fortran Composer XE 2011 includes the latest generation of Intel® Fortran compilers, Intel® Visual Fortran Compiler XE 12.0 for Windows. Intel® Fortran Composer XE is available for Linux and Mac OS X. This package delivers advanced capabilities for development of application parallelism and winning performance for the full range of Intel® processor-based platforms and other compatible platforms. It includes the compiler's breadth of advanced optimization, multithreading, and processor support, as well as automatic processor dispatch, vectorization, and loop unrolling.

for Windows
Single (SSR)
Paradise #
I23 86101E03
$263.99

programmers.com/intel

Microsoft SQL Server Developer Edition 2008 R2

by Microsoft

SQL Server 2008 Developer enables developers to build and test applications that run on SQL Server on 32-bit, ia64, and x64 platforms. SQL Server 2008 Developer includes all of the functionality of Enterprise Edition, but is licensed only for development, test, and demo use. The license for SQL Server 2008 Developer entitles one developer to use the software on as many systems as necessary. For rapid deployment into production, instances of SQL Server 2008 Developer can easily be upgraded to SQL Server 2008 Enterprise without reinstallation.

2-bit/x64
IA64 DVD
Paradise #
M47 31101A04
$41.99

programmers.com/microsoft

Win an iPad

Place an Order for Software (or Hardware) with Programmer's Paradise and You'll be Entered for a Drawing to Win an iPad Wi-Fi 32GB.

Just Use the **Offer Code TRWD01** When You Place Your Order On-line or with Your Programmer's Paradise Representative.

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

Prices subject to change. Not responsible for typographical errors.

Change the World

The Internet, of course, has no end. But it did have a beginning. Nov. 12, 1990, was when Tim Berners-Lee detailed his plan for a "Hypertext Project" (w3.org/Proposal.html). For geeks like me, reading it can raise goosebumps. In a way, it's like reading the U.S. Constitution—this document established a framework for a new way of doing things.

Back in 1989-1990, Berners-Lee was looking for a way to more easily share information around the massive physics lab at which he worked. The idea was actually pretty basic, but of course what came out of it is not.

Here's a pretty nice summation of their plan. How can you not get excited while reading this description?

"HyperText is a way to link and access information of various kinds as a web of nodes in which the user can browse at will. Potentially, HyperText provides a single user-interface to many large classes of stored information such as reports, notes, data-bases, computer documentation and on-line systems help."

The point of our little trip in Doc Brown's DeLorean back to the Web's beginning: You never know where an idea will lead.

Being, among other things, a first-class software engineer, Berners-Lee had a specific problem in mind that he wanted to solve:

"There is a potential large benefit from the integration of a variety of systems in a way which allows a user to follow links pointing from one piece of information to another one. This forming of a web of information nodes rather than a hierarchical tree or an ordered list is the basic concept behind HyperText."

Systems integration never sounded so simple, did it? The key component to tie the information together is described here:

"A program which provides access to the hypertext world we call a browser. When starting a hypertext browser on your workstation, you will first be presented with a hypertext page which is personal to you: your personal notes, if you like. A hypertext page has pieces of text which refer to other texts. Such references are highlighted and can be selected with a mouse (on dumb terminals, they would appear in a numbered list and selection would be done by entering a number). When you select a reference, the browser presents you with the text which is referenced: you have made the browser follow a hypertext link. That text itself has links to other texts and so on."

And here, in a nutshell, is why it's so cool:

"The texts are linked together in a way that one can go from one concept to another to find the information one wants. The network of links is called a web. The web need not be hierarchical, and therefore it is not necessary to 'climb up a tree' all the way again before you can go down to a different but related subject."

And the rest is history. The point of our little trip in Doc Brown's DeLorean back to the Web's beginning: You never know where an idea will lead. But if you don't break through boundaries, don't ask "What if …?" enough, don't aspire to make a bigger impact, you surely won't.

It's now 2011. The year ahead looms. Will you be content with doing the same old thing you've been doing for weeks, months or years? Or will you invent something? You can't invent the Web, but you can invent a way to do it better. Can you make the world better with your software, even if it's only a tiny little corner of it? In other words—are you satisfied, or are you wanting to do more than you've done?

Stay thirsty, my friends.

Keith Ward

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

OnTime Now!

Cloud-based Scrum Tool for Developers

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

$39 / month for 5 users • Free 30-day Team Trial

axosoft

Software for Software Development

Subversion for Windows on IIS

Finally, you can utilize Subversion inside a Windows environment. Quick setup and intuitive management make RocketSVN the premiere Subversion solution for your team.

Server: 2 users Free • $79 each add'l user
Visual Studio Client: $49/user

Local Scrum Tool for Developers

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

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

www.axosoft.com
(800) 653-0024 • (480) 362-1900

TERRENCE DORSEY

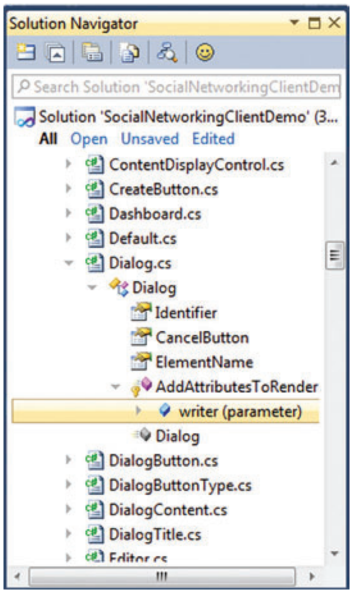
Visual Studio Tools and Extensions

Because you're reading this magazine, there's a good chance you sling code for a living. And if you sling code for a living, you probably spend a lot of time in your IDE … which is—because you're reading this magazine—probably Visual Studio.

Visual Studio 2010 is already an incredibly versatile coding tool. It does pretty much everything except write the code for you, and in many cases it's getting good at doing that, too. Still, Visual Studio can't do it all out of the box.

That's where extensions come to the rescue. Visual Studio 2010 provides robust support for extensibility via custom tools, templates and plug-ins. (Note, however, that the Express versions of Visual Studio don't support extensions.) If you can't find the feature you need in Visual Studio, chances are there's an extension that helps you customize the IDE or provides the tools you need to write code better and faster.

We'll cover a few of the most popular free extensions for Visual Studio 2010.



Solution Navigator in Productivity Power Tools

Power Tools for Visual Studio There are thousands of extensions out there, and it just so happens that one of the most robust extensions was created by the Visual Studio team. Visual Studio 2010 Productivity Power Tools (bit.ly/g4fUGG) is a package of 15 handy features that range from Solution Navigator (think Solution Explorer on steroids) to tab autocompletion and highly configurable enhancements to tabs.

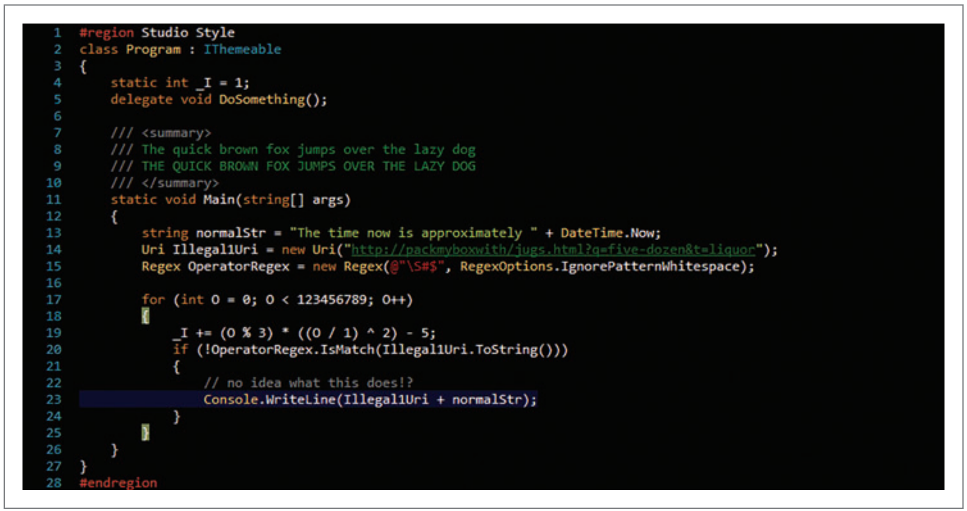
Scott Guthrie explains how each of the features in Productivity Power Tools works on his blog, so check that out for details (bit.ly/aopeNt).

PowerCommands 10.0 PowerCommands 10.0 (bit.ly/hUY9tT), like Productivity Power Tools, is a grab bag of useful extra tools that will speed up or simplify common tasks in the IDE. You get 25 features in the package; they include robust copy and paste enhancements (copying entire classes, for example). The package also includes the ability to format your code, sort using statements and remove unused using references when saving.

Team Foundation Server Power Tools September 2010 Don't feel left out if you're using Visual Studio Team Foundation Server (TFS). Microsoft has a set of Power Tools for you, too. This extension (bit.ly/hyUNqo) gives you 11 new features that include check-in policies and item templates, a process editor, TFS command-line tools and Windows Powershell cmdlets, team member management, Windows shell integration and automated database backup.

Visual Studio Color Theme Editor It may not sound as glamorous, but sometimes it's the little details that make coding that much easier. Take the colors used in the Visual Studio windows, tabs and menus, for instance. Do brighter colors cheer your mood? Are you particularly fond of magenta? Whatever you prefer, Visual Studio Color Theme Editor (bit.ly/fPKKEV) lets you customize all of the environment colors used in the IDE. You can also save themes and share them with your friends.

StudioStyles An even more personal choice is the colorization used for the code itself in your editor. StudioStyles (studiostyl.es) is a Web site that lets you download, create and share the .vssettings files that specify code colorization. Added bonus: These themes can be used with Visual Studio 2010, 2008, 2005 and even the Express versions.



```
#region Studio Style
class Program : IThemeable
{
    static int _I = 1;
    delegate void DoSomething();

    /// <summary>
    /// The quick brown fox jumps over the lazy dog
    /// THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG
    /// </summary>
    static void Main(string[] args)
    {
        string normalStr = "The time now is approximately " + DateTime.Now;
        Uri Illegal1Uri = new Uri("http://packmyboxwith/jugs.html?q=five-dozen&t=liquor");
        Regex OperatorRegex = new Regex(@"\S#$", RegexOptions.IgnorePatternWhitespace);

        for (int O = 0; O < 123456789; O++)
        {
            _I += (O % 3) * ((O / 1) ^ 2) - 5;
            if (!OperatorRegex.IsMatch(Illegal1Uri.ToString()))
            {
                // no idea what this does!?
                Console.WriteLine(Illegal1Uri + normalStr);
            }
        }
    }
}
#endregion
```

StudioStyles

360°

Application Transformation

Essential Studio Enterprise Edition:

- Powerful Silverlight 4 controls
- Complete set of ASP.NET MVC components
- Cutting edge Business Intelligence components

Syncfusion®

ASP.NET ASP.NET MVC Windows Forms WPF Silverlight

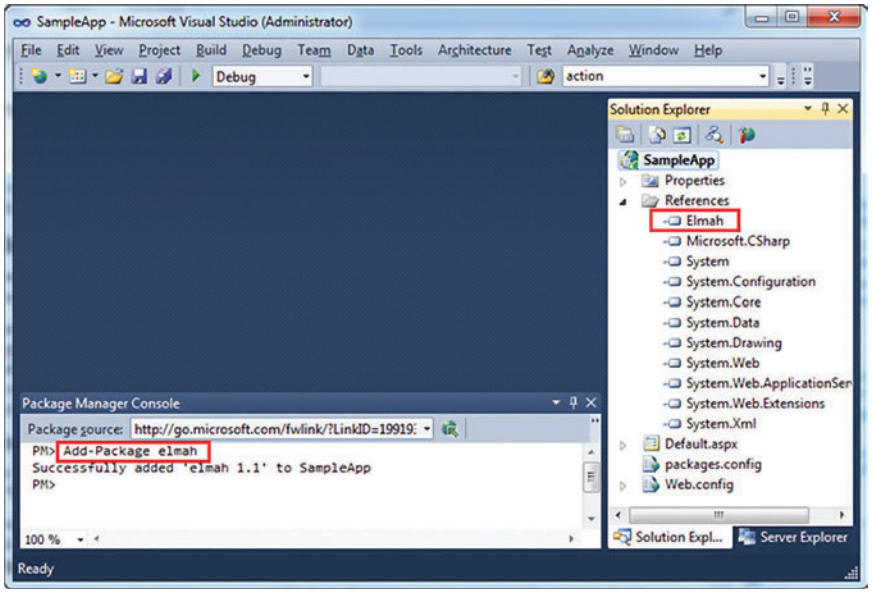
1-888-9DOTNET | www.syncfusion.com

WordLight Do you ever want to quickly find all the places you've used a method or variable name? WordLight (**code.google.com/p/wordlight**) is a simple extension for Visual Studio 2008 that lets you select some text and instantly highlights all other occurrences of that string in the code file. It also works in the Output, Command and Immediate windows.

Spell Checker If y0u tpye lke I do, the Spell Checker is a lifesaver. The Spell Checker extension (**bit.ly/aMrXoM**) looks for errors in the non-code portions of your files. It works in any plain-text files, for comments and strings in source code, and for non-tag elements of HTML and ASP files.

TortoiseSVN Add-in for Visual Studio So you've written and tested your code. If you're working on a team or open source project, you probably need to commit your source to a repository. There's a tool for that.

If you're using Apache Subversion (**subversion.apache.org**) source control along with a TortoiseSVN client for Windows (**tortoisesvn.tigris.org**), there are a couple of Visual Studio extensions that incorporate the TortoiseSVN functionality into the IDE (**tsvnaddin.codeplex.com**), saving you many steps in the commit process.



```
PM> Add-Package elmah
Successfully added 'elmah 1.1' to SampleApp
PM>
```

NuGet

If y0u tpye lke I do, the Spell Checker is a lifesaver. The Spell Checker extension (bit.ly/aMrXoM) looks for errors in the non-code portions of your files.

VsTortoise When using TFS, you'll need to add a layer such as SvnBridge (**svnbridge.codeplex.com**) that translates APIs between Subversion clients like TortoiseSVN (**vstortoise.codeplex.com**) and TFS.

Another popular source-code management system is Git (**git-scm.com**), and if that's your preferred repository, then there's an extension for you, too. Git Extensions (code.google.com/p/gitextensions) includes shell extensions for Windows Explorer and a Visual Studio plug-in. Plus, you can run most features from the command line.

NuGet Inspired by RubyGems and similar package-management systems from the Linux development world, NuGet (**nuget.codeplex.com/**) gives Microsoft .NET Framework developers the ability to easily incorporate libraries from source-code repositories directly into their local development projects. NuGet integrates with the Visual Studio 2010 IDE, and you can also run NuGet from the command line or via Windows PowerShell cmdlets.

Emacs and Vim Emulation In the beginning there was vi, and it was difficult to learn. Since those early days, Emacs and Vim have battled for supremacy as the One True Editor among coders. If you've chosen sides in that debate, yet find yourself using Visual Studio, then rejoice! The keybindings and many other features you know and love from Emacs and Vim are now available in extensions for Visual Studio.

You can follow the progress of VsVim (**bit.ly/e3GsMf**) developer Jared Parsons via his blog (**blogs.msdn.com/b/jaredpar/**). More information about Emacs emulation (**bit.ly/eXhalK**), along with lots of other great tips, can be found on the Visual Studio Team blog (**blogs.msdn.com/b/visualstudio/**).

A Gallery of Extensions This is just the tip of the iceberg as far as Visual Studio extensions are concerned. Thousands of templates, custom controls and extensions are available through the Visual Studio Gallery (**visualstudiogallery.msdn.microsoft.com**), and more are being added all the time. Many are free, and there are trial versions available for many of the commercial products.

Write Your Own Extensions Don't see what you need in the Visual Studio Gallery? Write your own! Visual Studio 2010 includes deep hooks for extensibility—anything from a custom project template to third-party tools that integrate directly with the IDE. Through the Extending Visual Studio developer center (**msdn.microsoft.com/vstudio/vextend**), MSDN Library articles and other resources in the Visual Studio community (**bit.ly/aT1bDe**), you'll find a vast amount of information to start creating custom Visual Studio extensions. You've already got the tools … start coding! ■

Terrence Dorsey is the technical editor of MSDN Magazine. *You can read his blog at terrencedorsey.com or follow him on Twitter at @tpdorsey.*

Blazing-Fast **Grid Controls**

Optimized for **.NET**

Award-Winning Development Tools by DevExpress

Learn more and *download* your free evaluation copy today

Visit **DEVEXPRESS.COM/GRIDS**

or **CALL US (818) 844-3383**

Devexpress™

Download • Compare • Decide!

PRESENTATION CONTROLS | REPORTING CONTROLS

IDE PRODUCTIVITY TOOLS | BUSINESS APP FRAMEWORKS

Interceptors in Unity

In last month's column, I briefly introduced the interception mechanism used in the Unity 2.0 dependency injection container. After illustrating the core principles of aspect-oriented programming (AOP), I presented a concrete example of interception that's probably very close to the needs of many developers today.

Have you ever wanted to extend the behavior of an existing piece of code without having to touch the source code in any way? Have you just wanted to be able to run some additional code around the existing code?

AOP was created to provide an approach that isolates core code from other concerns that crosscut the core business logic. Unity 2.0 offers a Microsoft .NET Framework 4-based framework for achieving this in a surprisingly quick and easy way.

To fully comprehend the purpose of this follow-up article, I'll begin by summing up what I discussed last month. As you'll find out, in last month's code I made some assumptions and used some default components. This month I'll step back and discuss in more detail the choices and options that you typically encounter along the way.

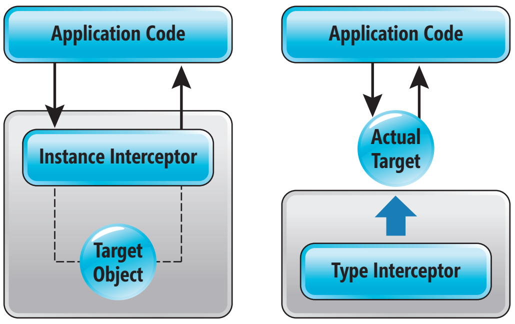


Figure 1 **Instance Interceptor and Type Interceptor**

AOP in Unity

Imagine you've a deployed application where, at some point, you perform some business-specific action. One day, your customer asks to extend that behavior to perform some more work. You grab the source code, modify it, and bill a few hours of consulting time for coding up and testing the new features. But wouldn't it be better if you could add new behaviors seamlessly and without touching the existing source code?

Imagine a slightly different scenario. First, what if you're not an independent consultant, but work full time for the company? The more requests for change you get, the more hours you spend outside your current project and, worse yet, you risk creating new (and not necessarily required) branches of your codebase. So, you'd heartily welcome any solutions that would let you add new behaviors seamlessly and with no exposure to the source code.

Finally, imagine that someone reports a bug or a serious performance hit. You need to investigate and fix the problem and you want it to be absolutely unobtrusive. In this case, too, it would be better to add new behaviors seamlessly and with no exposure to the source code.

AOP helps you in all of these scenarios.

Last month I demonstrated how to add pre- and post-processing code around an existing method without touching it by leveraging the interception API in Unity 2.0. That quick demo made a few assumptions, however.

First, it worked on a type registered with the Unity Inversion of Control (IoC) infrastructure and instantiated via the Unity factory layer.

Second, the pointcut was only defined through an interface. In AOP jargon, a pointcut represents a collection of places in the body of the target class where the framework will inject extra behaviors on demand. An interface-based pointcut means that only the members of the interface will be extended at runtime via code injection.

Third, I mostly focused on the configuration settings to enable interception and disregarded the fluent API that allows you to configure Unity in code.

In the rest of this article, I'll explore the fluent API and alternative ways of defining Unity interceptors.

Interceptable Instances

To add a new behavior to an existing or freshly created instance of a class, you must take some control over the factory. In other words, AOP is not pure magic and you'll never be able to hook up a plain CLR class instantiated via the standard new operator:

```
var calculator = new Calculator();
```

The way in which an AOP framework gains control over the instance may differ quite a bit. In Unity, you can resort to some explicit calls that return a proxy for the original object or keep the whole thing running behind an IoC framework. For this reason, most IoC frameworks offer AOP capabilities. Spring.NET and Unity are two examples. When AOP meets IoC, the result is a seamless, easy and effective coding experience.

Let's start with an example where no IoC capabilities are used. Here's some basic code that makes an existing instance of the Calculator class interceptable:

```
var calculator = new Calculator();
var calculatorProxy = Intercept.ThroughProxy<ICalculator>(calculator,
  new InterfaceInterceptor(), new[] { new LogBehavior() });
Console.WriteLine(calculatorProxy.Sum(2, 2));
```

You end up working with an interceptable proxy that wraps your original object. In this case, I'm assuming that the Calculator class implements the ICalculator interface. To be interceptable, a class must either implement an interface or inherit from MarshalByRefObject. If the class derives from MarshalByRefObject, then the interceptor must be of type TransparentProxyInterceptor:

Full-Featured **Reporting Controls** Optimized for **.NET**

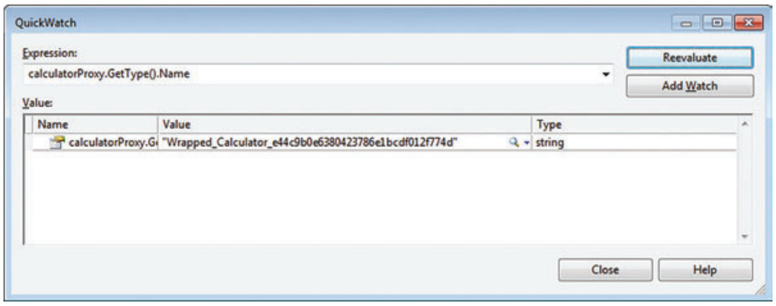
Award-Winning Development Tools by DevExpress

Awards 2009-10 ComponentSource #1 PRODUCT
Awards 2009-10 ComponentSource #1 PUBLISHER
MEMBERS CHOICE WINNER THE CODE PROJECT 2010
2010 READERS CHOICE Visual Studio MAGAZINE
DevProConnections 2010 Community Choice GOLD

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

DevExpress™
Download • Compare • Decide!

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



Expression:
calculatorProxy.GetType().Name

Value:

Name	Value	Type
calculatorProxy.G	"Wrapped_Calculator_e44c9b0e6380423786e1bcdf012f774d"	string

Figure 2 **Actual Type After Type Interception**

```
var calculator = new Calculator();
var calculatorProxy = Intercept.ThroughProxy(calculator,
  new TransparentProxyInterceptor(), new[] { new LogBehavior() });
Console.WriteLine(calculatorProxy.Sum(2, 2));
```

The Intercept class also offers a NewInstance method you can call to create an interceptable object in a more direct way. Here's how to use it:

```
var calculatorProxy = Intercept.NewInstance<Calculator>(
  new VirtualMethodInterceptor(), new[] { new LogBehavior() });
```

Note that when you use NewInstance, the interceptor component has to be slightly different—neither InterfaceInterceptor nor TransparentProxyInterceptor, but rather a VirtualMethodInterceptor object. So how many types of interceptors exist in Unity?

Instance and Type Interceptors

An interceptor is the Unity component responsible for capturing the original call to the target object and routing it through a pipeline of behaviors so that each behavior has its chance to run before or after the regular method call. Interception can be of two types: instance interception and type interception.

Instance interceptors create a proxy to filter incoming calls directed at the intercepted instance. Type interceptors generate a new class derived from the type being intercepted, and work on an instance of that derived type. Needless to say, the delta between the original and derived type is all in the logic required to filter incoming calls.

In case of instance interception, the application code first creates the target object using a classic factory (or the new operator), and then is forced to interact with it through the proxy provided by Unity.

In the case of type interception, the application creates the target object through the API or Unity, then works with that instance. (You can't create the object directly with the new operator and get type interception.) The target object, however, is not of the original type. The actual type is derived by Unity on the fly and incorporates interception logic (see **Figure 1**).

InterfaceInterceptor and TransparentProxyInterceptor are two Unity interceptors that belong to the instance interceptor category. VirtualMethodInterceptor belongs to the type interceptor category.

InterfaceInterceptor can intercept public instance methods on just one interface on the target object. The interceptor can be applied to new and existing instances.

TransparentProxyInterceptor can intercept public instance methods on more than one interface, and marshal-by-reference objects. This is the slowest approach for interception, but it can intercept the widest set of methods. The interceptor can be applied to new and existing instances.

VirtualMethodInterceptor can intercept virtual methods both public and protected. The interceptor can only be applied to new instances.

It should be noted that instance interception can be applied to any public instance methods, but not to constructors. This is fairly obvious for the scenario in which interception applies to an existing instance. It's a bit less obvious when interception is applied to a newly created instance. The implementation of instance interception is such that the constructor has already executed by the time the application code gets back an object to work with. As a result, any interceptable action necessarily follows the creation of the instance.

Type interception uses dynamic code generation to return an object that inherits from the original type. In doing so, any public and protected virtual methods are overridden to support interception. Consider the following code:

```
var calculatorProxy = Intercept.NewInstance<Calculator>(
  new VirtualMethodInterceptor(), new[] { new LogBehavior() });
```

The Calculator class looks like this:

```
public class Calculator {
  public virtual Int32 Sum(Int32 x, Int32 y) {
    return x + y;
  }
}
```

Figure 2 shows the actual name of the type that results from a dynamic inspection of the calculatorProxy variable.

It's also worth noting that there are some other significant differences between instance and type interception—for example, intercepting calls by the object on itself. If a method is calling another method on the same object when using type interception, then that self-call can be intercepted because the interception logic is in the same object. However, with instance interception, the interception only happens if the call goes through the proxy. Self-calls, of course, don't go through the proxy and therefore no interception happens.

Using the IoC Container

In last month's example, I used the IoC container of the Unity library to take care of object creation. An IoC container is an extra layer around object creation that adds flexibility to the application. This is even truer if you consider IoC frameworks with additional AOP capabilities. Furthermore—and as I see things—the level of code flexibility grows beyond imagination if you also combine IoC containers with offline configuration. However, let's start with an example that uses the Unity's container with code-based fluent configuration:

```
// Configure the IoC container
var container = UnityStarter.Initialize();

// Start the application
var calculator = container.Resolve<ICalculator>();
var result = calculator.Sum(2, 2);
```

Any code needed to bootstrap the container can be isolated in a distinct class and invoked once at application startup. The bootstrap code will instruct the container how to resolve types around the application and how to deal with interception. A call to the Resolve method shields you from all the details of interception. **Figure 3** shows a possible implementation of the bootstrap code.

The nice thing is that this code can be moved to a separate assembly and loaded or changed dynamically. More importantly, you have a single place to configure Unity. This won't happen as long as you stick to the Intercept class that behaves like a smart factory and needs to be prepared every time you use it. So if you need AOP

Powerhouse **Analytics Controls**
Optimized for **.NET**

Award-Winning Development Tools by DevExpress

Awards 2009-10 ComponentSource #1 PRODUCT
Awards 2009-10 ComponentSource #1 PUBLISHER
MEMBERS CHOICE WINNER THE CODE PROJECT 2010
2010 READERS CHOICE Visual Studio MAGAZINE
DevProConnections 2010 Community Choice GOLD

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

Devexpress™
Download • Compare • Decide!

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

Figure 3 **Bootstrapping Unity**

```
public class UnityStarter {
  public static UnityContainer Initialize() {
    var container = new UnityContainer();

    // Enable interception in the current container
    container.AddNewExtension<Interception>();

    // Register ICalculator with the container and map it to
    // an actual type. In addition, specify interception details.
    container.RegisterType<ICalculator, Calculator>(
      new Interceptor<VirtualMethodInterceptor>(),
      new InterceptionBehavior<LogBehavior>());

    return container;
  }
}
```

Figure 4 **Adding Interception Details Via Configuration**

```
<unity xmlns="http://schemas.microsoft.com/practices/2010/unity">
  <assembly name="SimplestWithConfigIoC"/>
  <namespace name="SimplestWithConfigIoC.Calc"/>
  <namespace name="SimplestWithConfigIoC.Behaviors"/>

  <sectionExtension
    type="Microsoft.Practices.Unity.
      InterceptionExtension.Configuration.
      InterceptionConfigurationExtension,
      Microsoft.Practices.Unity.Interception.Configuration" />

  <container>
    <extension type="Interception" />

    <register type="ICalculator" mapTo="Calculator">
      <interceptor type="InterfaceInterceptor"/>
      <interceptionBehavior type="LogBehavior"/>
      <interceptionBehavior type="BinaryBehavior"/>
    </register>

    <register type="LogBehavior">
    </register>

    <register type="BinaryBehavior">
    </register>

  </container>
</unity>
```

Figure 5 **A Sample Behavior**

```
public class BinaryBehavior : IInterceptionBehavior {
  public IEnumerable<Type> GetRequiredInterfaces() {
    return Type.EmptyTypes;
  }

  public bool WillExecute {
    get { return true; }
  }

  public IMethodReturn Invoke(
    IMethodInvocation input,
    GetNextInterceptionBehaviorDelegate getNext) {

    // Perform the operation
    var methodReturn = getNext().Invoke(input, getNext);

    // Grab the output
    var result = methodReturn.ReturnValue;

    // Transform
    var binaryString = ((Int32)result).ToBinaryString();

    // For example, write it out
    Console.WriteLine("Rendering {0} as binary = {1}",
      result, binaryString);

    return methodReturn;
  }
}
```

in your applications, by all means get it via an IoC container. The same solution can be implemented in an even more flexible way by moving the configuration details off to the app.config file (or web.config if it's a Web application). In this case, the bootstrap code consists of the following two lines:

```
var container = new UnityContainer();
container.LoadConfiguration();
```

Figure 4 shows the script you need to have in the configuration file. Here I registered two behaviors for the ICalculator type. This means that any calls to public members of the interface will be pre- and post-processed by LogBehavior and BinaryBehavior.

Note that, because LogBehavior and BinaryBehavior are concrete types, you actually don't need to register them at all. Unity's defaults will automatically work for them.

Behaviors

In Unity, behaviors are objects that actually implement the crosscutting concerns. A class that implements the IInterceptionBehavior interface, a behavior rewrites the execution cycle of the intercepted method and can modify method parameters or return values. Behaviors can even stop the method from being called at all or call it multiple times.

A behavior is made of three methods. **Figure 5** shows a sample behavior that intercepts the method Sum and rewrites its return value as a binary string. The method WillExecute is simply a way to optimize the proxy. If it returns false, the behavior won't execute.

This is actually a bit subtler. Invoke will always be called, so your behavior will in fact execute even if you returned false. However, when the proxy or derived type is being created, if all the behaviors registered for the type have WillExecute set to false, then the proxy itself won't be created and you'll be working with the raw object again. It's really about optimizing proxy creation.

The GetRequiredInterfaces method allows the behavior to add new interfaces to the target object; interfaces returned from this method will be added to the proxy. Hence, the core of a behavior is the Invoke method. The parameter input gains you access to the method being called on the target object. The parameter getNext is the delegate for you to move to the next behavior in the pipeline and eventually execute the method on the target.

The Invoke method determines the actual logic used to execute a call to a public method on the target object. Note that all intercepted methods on the target object will execute according to the logic expressed in Invoke.

What if you want to use more specific matching rules? With plain interception as I described in this article, all you can do is run through a bunch of IF statements to figure out which method is actually being invoked, like so:

```
if(input.MethodBase.Name == "Sum") {
  ...
}
```

Next month I'll resume from here to discuss more effective ways to apply interception and define matching rules for intercepted methods.■

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

THANKS to the following technical expert for reviewing this article: *Chris Tavares*

Elegant **Charting Controls**
Optimized for **.NET**

Award-Winning Development Tools by DevExpress

2009-10 ComponentSource #1 PRODUCT | 2009-10 ComponentSource #1 PUBLISHER | Members Choice Winner The Code Project 2010 | 2010 Readers Choice Visual Studio Magazine | DevProConnections 2010 Community Choice Gold

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

DevExpress™
Download • Compare • Decide!

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

JOSEPH FULTZ

Branch-Node Synchronization with SQL Azure

In my years prior to joining Microsoft and for the first few years thereafter, I was heavily involved in the retail industry. During this time, I found it almost humorous to see how many times the branch-node synchronization problem gets resolved as technologies advance.

In my current role, I've had a fair amount of exposure to the oil and gas (O&G) industry, and I've found that it has a similar problem of synchronizing data between nodes. Much like retail chains, O&G companies have a wide range of devices and connectivity challenges. From a latent satellite connection on an offshore oil platform to an engineer in an oil field, the requirement for timely, accurate data doesn't change.

So, with both retail and O&G in mind, I'm taking a look at the challenge again, but this time with a little help from SQL Azure and the Sync Framework. I'll discuss how the cloud will help solve the problem of moving data between the datacenter (corporate), the branches (for example, store, rig, hub and so on) and to individual devices (handheld, shared terminal, specific equipment and more).

This month, I'll focus a little more on the general architecture and a little less on the implementation. I'll still give a few code examples for setting up synchronization between nodes and SQL Azure and filtering content as a way to reduce traffic and time required for synchronization. In next month's column, I'll examine using a service-based synchronization approach to provide a scalable synchronization solution beyond splitting data across SQL Azure databases based on content or geographic distribution.

While the core problem hasn't changed, what have changed are the additional requirements that get added to the mix as technology becomes more advanced. Instead of solving the simple problem of moving data between nodes, we start adding things we'd like to have, such as increasing the data volume to get more detail, inclusion of various devices for collecting and displaying data, and closer-to-real-time feeds.

Let's face it, the more we have, the more we want. In most cases, solving the data flow problems from a decade ago would be simple, but in today's world that solution would only represent the substrate of a more robust solution. For retail, the data flow can be pretty easy—taking the form of pushing catalog-type data (menu, warehouse and so on) down and t-logs back up—to quite complex, by frequently updating inventory levels, real-time loss-prevention analysis, and manual product entries from the branch to corporate and between branches. For the most part, O&G companies have the same patterns, but they have some added complexities related to the operation, evaluation and adjustment of equipment in use.

I think about synchronization in the following ways to get a rough idea of the level of complexity (each one is subsequently more complex to implement and support):

1. Push read-only data from corporate to branch and onward.
2. Two one-way pushes on different data; one from corporate to branch (for example, catalog) and one from branch to corporate (for example, t-logs and inventory); this includes branch to branch—the focus is on the fact that it's basically two or more one-way syncs.
3. Bidirectional data synchronization between corporate and nodes (for example, manual product entry or employee information).
4. Peer synchronization between branches and between branches and corporate.

Type 4 is by far the most complex problem and typically leads to many conflicts. Therefore, I try to avoid this pattern, and the only two criteria that would force it are the need for real-time updates between nodes or the ability to sync branches if the corporate data store isn't accessible. Because near-real-time or real-time updates among too many nodes would generally create too much traffic and isn't typically a reasonable solution, the only criterion to which I really pay attention is the ability to sync without the master. In some cases, real-time information is needed between nodes, but this isn't generally the case for data synchronization. Rather, it's an event notification scenario, and a different tack is taken to address the need.

In some cases, real-time information is needed between nodes, but this isn't generally the case for data synchronization.

Defining the Solution Architecture

Generally, the most prevalent pattern that I see is to push data directly from the corporate master database (via a distributor of some type) down to servers at the branches and to mobile users. The distribution to workstations, point-of-sale (POS) terminals and other such devices is typically done from the server at the branch location (commonly called "back-of-house servers"), whereas the

WE ARE

REPORTING

The defacto standard reporting tool for Microsoft Visual Studio.NET

- Fast and Flexible reporting engine
- Flexible event-driven API to completely control the rendering of reports
- Wide range of Export and Preview formats including Windows Forms Viewer, Web Viewer, Adobe Flash and PDF
- XCopy deployment
- Royalty-Free Licensing for Web and Windows applications

ACTIVEREPORTS

ACTIVEREPORTS SPREAD DATA DYNAMICS REPORTS ACTIVEANALYSIS

GrapeCity PowerTools
Report Analyze Excel

www.GCPowerTools.com

WE ARE
SPREADSHEETS

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

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

SPREAD

ACTIVEREPORTS SPREAD DATA DYNAMICS REPORTS ACTIVEANALYSIS

GrapeCity PowerTools
Report Analyze Excel

www.GCPowerTools.com

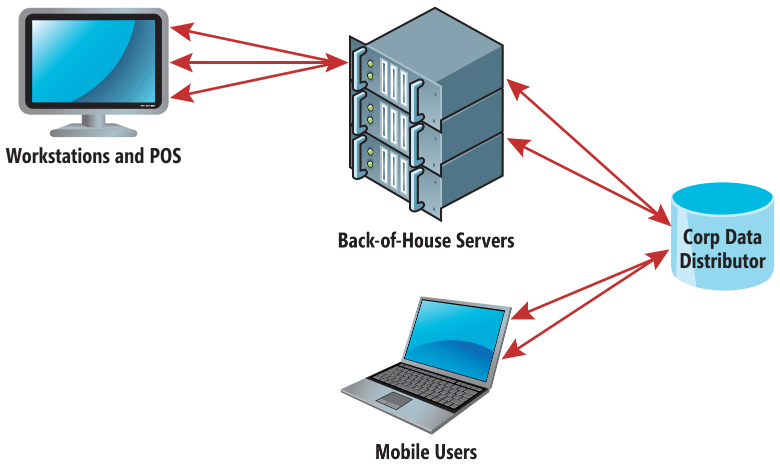


Figure 1 **Typical Architecture for Data Distribution**

synchronization to mobile users (for example, laptops) is done directly from corporate to the machine via a client-initiated synchronization process (see **Figure 1**).

Some organizations do this via the built-in replication features of the relational database management system (RDBMS), while others build processes to handle the distribution and collection of data. I'll maintain the pattern, but use an instance of SQL Azure in place of the distributor; in place of replication, I'll use the Sync Framework, which supports SQL Azure. Thus, I simply add a layer between the distributor and the branches (see **Figure 2**).

What do I get by inserting SQL Azure? Some of the benefits in a branch-node scenario are:

1. Scaling the data service without having to grow the datacenter footprint.
2. High availability of the data without additional cost and effort.
3. Potentially less security sensitivity, because it isn't the master data store.

Consider that in the first scenario, if the corporate connection or data store is down, all of the clients will have to hold on to their transactions. This could easily lead to lost data due to losing the device while waiting for the connection or due to simply running out of space on the device to store the transactions, as mentioned earlier. Additionally, if the branches have common data (for example, warehouse inventory data), they'll be working off of old data until corporate is back up. While there's no perfect solution, SQL Azure addresses this scenario by automatically making copies of the data and providing automatic failover. Also, by segmenting the flow of data through multiple SQL Azure databases, I can reduce the risk of being down and reduce load exposure further by using not only a separate instance but also different datacenters.

As a point of design, I must consider the impact of initiating synchronization from the branches or from the server. If the application is designed to sync from the master or distributor to the nodes, I enjoy the advantage of fewer points of management and support; on the downside, it puts some technical strains on the implementation, requiring:

1. Knowing the endpoints.
2. Knowing the appropriate scopes for each target.
3. Complexity in the synchronization process in order to make the synchronization of multiple nodes happen in parallel; the API semantics are really one pair of endpoints and one scope at a time.

By initiating synchronization from the target (node or branch, for example) the complexity is diminished for the synchronization code, as it:

- Can focus on the scope(s) for the application/device.
- More easily handles being occasionally connected.
- Only has to know and manage a few endpoints by which the distribution data is synchronized.

However, it will put a bit more complexity into the applications on the target device and could complicate support and maintenance by potentially having to debug the sync process or agent at each device. Ideally, if data must be synchronized for different applications, a separate process should be created that manages the synchronization based on a configuration file where scope, frequency and connection strings are defined for a sync agent to run. This sync agent would exist externally to the applications that are the data consumers on the devices, although the process would provide a means for a given application to initiate synchronization of its data. This gives the benefit of initiating synchronization from the node, but it also reduces the support and maintenance aspect, because it's rolled up into a single process.

Using the Sync Framework, I tend to start with a mixed model of synchronization by initiating from the master data store to SQL Azure and subsequently initiating from the nodes to sync between

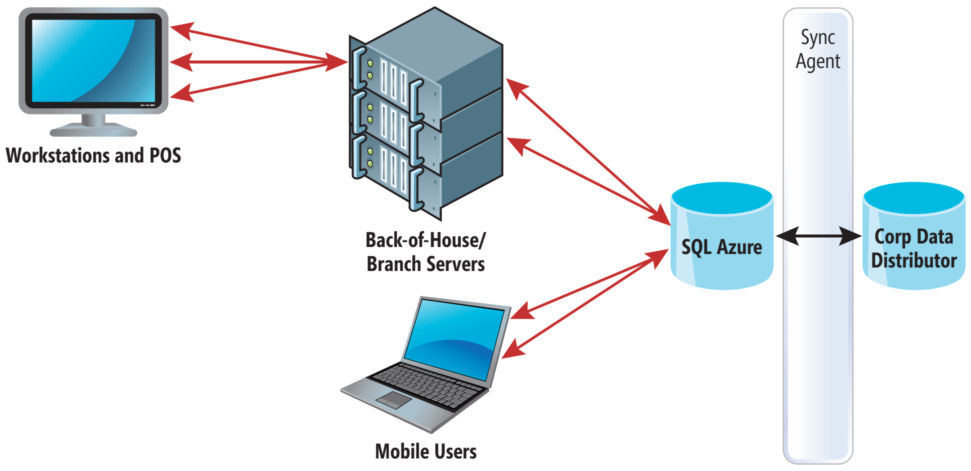


Figure 2 **Base Architecture Using SQL Azure and the Sync Framework**

THANKS for Making DevExpress #1

BESTSELLING PRODUCT AWARDS Awards 2009-10 ComponentSource® The Definitive Source of Software Components

#1 PRODUCT

ComponentSource® Awards 2009-10

BESTSELLING PUBLISHER AWARDS Awards 2009-10 ComponentSource® The Definitive Source of Software Components

#1 PUBLISHER

ComponentSource® Awards 2009-10

WinForms Controls	ASP.NET AJAX Controls	Silverlight Controls	WPF Controls	Visual Studio Tools	Application Frameworks	Delphi VCL Controls
WIN	ASP	SL	WPF	IDE		VCL

Award-Winning Development Tools

For a Free Trial Version Visit: www.DevExpress.com

Feature-Complete Presentation Components • Easy-to-Use Reporting Controls
IDE Productivity Tools • Business Application Frameworks

DevExpress™
Download • Compare • Decide!

Figure 3 **Creating a Synchronization Scope**

```
SqlConnection azureConn = new SqlConnection(AzureConnectionString);
SqlConnection onPremiseConn = new SqlConnection(LocalConnectionString);

// List of columns to include
Collection<string> columnsToInclude = new Collection<string>();
columnsToInclude.Add("au_id");
columnsToInclude.Add("au_lname");
columnsToInclude.Add("au_fname");
columnsToInclude.Add("phone");
columnsToInclude.Add("address");
columnsToInclude.Add("city");
columnsToInclude.Add("state");
columnsToInclude.Add("zip");
columnsToInclude.Add("contact");

// Definition for authors from local DB
DbSyncTableDescription authorsDescription =
  SqlSyncDescriptionBuilder.GetDescriptionForTable("authors",
  columnsToInclude, onPremiseConn);

// Create a scope and add tables to it
DbSyncScopeDescription authorScopeDesc = new DbSyncScopeDescription(ScopeName);

// Add the authors table to the sync scope
authorsScopeDesc.Tables.Add(authorsDescription);
```

the node and SQL Azure. Restated, one might say that data is pushed from master and pulled from the branches, with SQL Azure becoming the highly available central hub between the master and branches. Based on the needs and constraints of the solution being considered, I think about the costs and benefits of moving the synchronization process control from one point in the chain to another (for example, device to cloud or corporate to cloud). Just a few of these considerations are questions such as:

- Is there a place to host the process for master?
- Are there security policies that conflict with hosting the sync process in SQL Azure?
- How many nodes at each level are synchronizing?
- Can the target device realistically support a sync process?
- What's the requirement with regard to timeliness of data sync?

SQL Azure becomes the highly available central hub between the master and branches.

What's more, each of these questions has multiple layers that need to be considered and against which possible solution designs must be vetted. While there isn't a one-for-all design, I like to start with the model described earlier and either synchronize multiple one-way syncs to accomplish something similar to bidirectional data sync, or use bidirectional synchronization between the device/corporate database and SQL Azure. After which, I look for scenarios that invalidate and force a modification to the design. Generally, the only synchronization style I attempt to avoid is peer-to-peer.

Setting up Synchronization

There are two methods to set up synchronization using Sync Framework 2.1: sync client in the cloud and sync client on the local

SQL Azure Data Sync

SQL Azure Data Sync is a cloud-based service hosted in Windows Azure that enables synchronization of entire databases or specific tables between SQL Server and SQL Azure. At the Microsoft Professional Developers Conference 2010, we announced an update to this service called SQL Azure Data Sync Community Technology Preview (CTP) 2. This update provides organizations the ability to easily extend on-premises SQL Server databases to the cloud, allowing for phased migrations of applications to the cloud. Solutions leveraging SQL Azure Data Sync will allow users to continue to access local data and have changes seamlessly synchronized to SQL Azure as they occur. Similarly, any changes made by applications to SQL Azure are synchronized back to the on-premises SQL Server.

Keeping Data in Sync

SQL Azure Data Sync provides a central cloud-based management system for all synchronization relationships. From any browser, administrators can connect to the public service and manage and monitor the various database endpoints. In addition, SQL Azure Data Sync provides a scheduling service that allows synchronization to take place as often as every five minutes, or less frequently if the preference is to run synchronization at off-peak times.

In the recent SQL Azure Data Sync CTP 2 update, we also introduced a new component called the SQL Azure Data Sync Agent. This agent is a Windows Service that's installed on-premises and links the local SQL Server databases to SQL Azure Data Sync through a secure outbound HTTPS connection, meaning there are no requirements from a firewall or security-configuration standpoint—which makes setup a snap. The Agent's job is to monitor and log tasks as well as to initiate synchronization requests from the SQL Azure Data Sync.

New Scenarios

With SQL Azure Data Sync, synchronization between SQL Server and SQL Azure databases provides for a wealth of new scenarios that, in the past, were quite difficult to build. Imagine you wanted to share data with your branch offices or retail store databases. With SQL Azure Data Sync, this is easy because administrators create "Sync Groups" that define data to be shared among databases. These Sync Groups could contain a corporate SQL Server that synchronizes data to a centralized SQL Azure "Data Hub." Then, from this Data Hub, all of the remote or regional SQL Server databases can synchronize data changes, enabling them to bring data closer to the users, while also greatly reducing bandwidth and requirements for Virtual Private Networks, or VPNs.

In addition, the ability to synchronize across multiple SQL Azure datacenters makes it easy to scale out loads across geographies. Imagine you have quarterly reporting requirements that put a huge cyclical load on your SQL Server database. Why not synchronize some of that data to your SQL Azure databases around the world when needed? Then users could access the data closest to them while reducing scalability requirements on your local SQL Server.

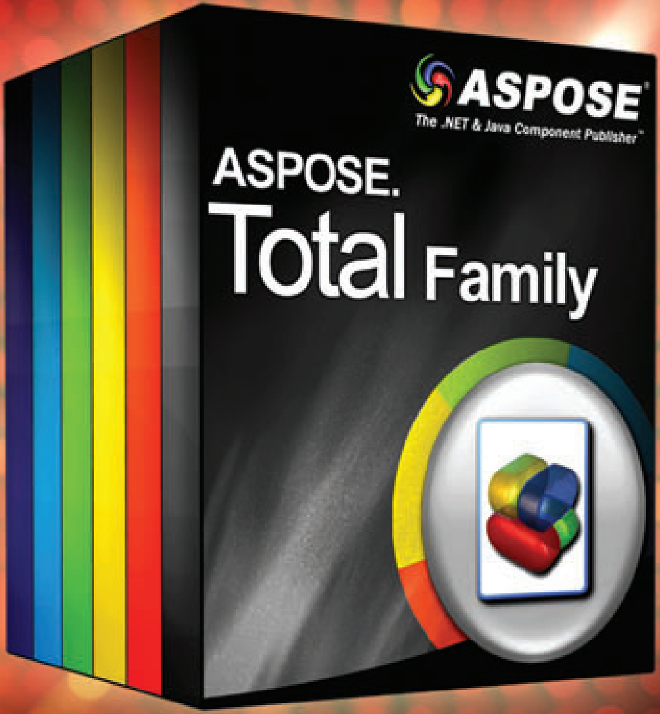
For more information and to register for participation in the CTP 2, please visit microsoft.com/en-us/sqlazure/datasync.aspx.

—Liam Cavanagh, Senior Program Manager, SQL Azure Data Sync

ASPOSE®

Your File Format Experts

Aspose Total Product Family



Every Aspose Component combined in one powerful suite!

Serving more than 50% of the Fortune 100 Companies!

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

DOCX PDF PPT ODF Report SWF InfoPath

XLSX BarCode MPP(Project) MSG(Outlook) ++

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

Get your FREE evaluation copy at www.aspose.com

ASPOSE

Your File Format Experts

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

Figure 4 **Provisioning Scope**

```
// Create a provisioning object for "customers" and
// apply it to the on-premises database
SqlSyncScopeProvisioning onPremScopeConfig =
  new SqlSyncScopeProvisioning(onPremiseConn, authorsScopeDesc);
if (!(onPremScopeConfig.ScopeExists(authorsScopeDesc.ScopeName)))
{
  onPremScopeConfig.Apply():
}
// Provision the SQL Azure database from the on-premises SQL Server database
SqlSyncScopeProvisioning azureScopeConfig =
  new SqlSyncScopeProvisioning(azureConn, authorsScopeDesc);
if (!(azureScopeConfig.ScopeExists(authorsScopeDesc.ScopeName)))
{
  azureScopeConfig.Apply();
}
```

machine. I'll focus on the latter for the moment. At its simplest, here are the steps to set up a synchronization relationship:

1. Identify the data to be synchronized and the direction of data flow. This will be used to define the scopes (SqlSyncScopeProvisioning) used to synchronize the data.
2. Download and install the Sync Framework 2.1 (bit.ly/gKQODZ). *Note: If x64 is the target platform, a build target for x64 will need to be added or the SyncOrchestrator won't be able to resolve its dependencies.*
3. Provision the databases and tables for synchronization; the entire database could be provisioned, or only specific tables, or it can be limited to given columns.
4. Add necessary filters. In the case that it's desirable to horizontally partition or otherwise filter the data, filters may be used.
5. Create and run the process to synchronize.

I'll be rather specific in this example, as it helps convey the message, and I'll start with databases in place at both ends. I create a connection to the local database and retrieve a definition (DbSyncTableDescription) for the table to be synchronized and add that table to the scope (DbSyncScopeDescription). In addition, I'll specify the particular columns, but this isn't necessary if the desire is to simply synchronize the entire table. Limiting the sync relationship to specific columns is a good way to optimize bandwidth usage and speed up processes (see **Figure 3**).

For each structure that needs to be synchronized, a bit of code will need to be added to get the description; you must subsequently add it to the scope. The next step is to grab a scope-provisioning object and use it to provision each database if the scope doesn't already exist in that database, as shown in **Figure 4**.

Because this is the first time that the scope has been provisioned in the database, there will be some new tables for storing scope information and also a table specifically for tracking the Authors scope that was provisioned in the databases.

A good example of a console app to provision or sync a local and SQL Azure database can be found on the Sync Framework Team Blog at bit.ly/dCt6T0.

Synchronizing the Data

Once the databases are properly provisioned, it's pretty simple to get them synchronizing. It requires the creation of a SqlSyncProvider for each end of the activity with the scope specified. This involves the use of the SyncOrchestrator object, which is the magic behind the curtains that identifies the changes and moves the changes between them. That code looks something like this:

```
SqlConnection LocalConnection = new SqlConnection(LocalConnectionString);
SqlConnection AzureConnection = new SqlConnection(AzureConnectionString);

SqlSyncProvider LocalProvider = new SqlSyncProvider(ScopeName, LocalConnection);
SqlSyncProvider AzureProvider = new SqlSyncProvider(ScopeName, AzureConnection);

SyncOrchestrator orch= new SynOrchestrator();
orch.LocalProvider = new SqlSyncProvider(ScopeName, LocalConnection);
orch.RemoteProvider = new SqlSyncProvder(ScopeName, AzureConnection);
orch.Direction = SyncDirectionOrder.DownloadAndUpload;
orch.Synchronize();
```

By segmenting the flow of data through multiple SQL Azure databases, I can reduce the risk of being down.

Data and Geographic Dispersion

With simple replication of data handled, I can focus on optimizing the deployment architecture and data flow. Using the Sync Framework, I can specify filters; this in combination with SQL Azure can really be a huge benefit in branch-node architectures. Using the combination of the two, I can put the data closer to the ultimate consumer and optimize the bandwidth usage (and hence charges) by only synchronizing the data that matters for that

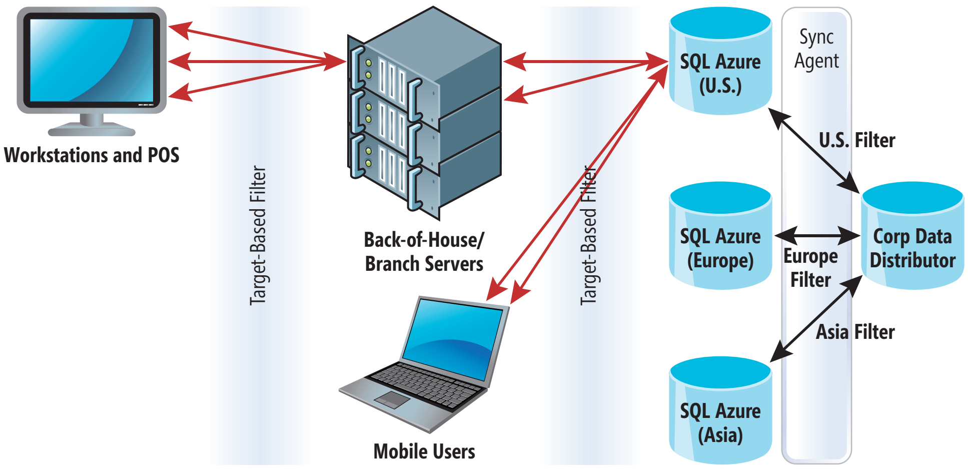


Figure 5 **Synchronizing Data with Filters**

region or data segmentation. Instead of using data servers in various geographical areas, data can simply be synchronized to an instance of SQL Azure in that geographical area, and those clients in that area can synchronize to it.

By spreading the data around geographically and implementing scopes that make sense for synchronizing particular data with a particular frequency, one can achieve fine-grained control over what, how, when and how much data flows across the wire, improving the user experience as it relates to data availability and freshness. Additionally, for end users who might travel between locations where it would be nice to be location-aware, the sync agent could locate itself and reach out to sync data specifically for the current location. A couple of examples of this are current stats or alerts for workers walking into a manufacturing/plant environment and current-day sales for regional managers of retail chains (see **Figure 5**).

Once the databases are properly provisioned, it's pretty simple to get them synchronizing.

Enabling filtering is no harder than provisioning a synchronization scope. Thus, there could be multiple scopes in existence that have different filters—or have none. The needed change is simply to add two lines of code for each filter that's being added: one line to add a filter column to the table and a second to add the filter clause, which is basically a "where" condition. For my sample, I'm adding a filter based on state and synchronizing only changes for the state of Utah, or UT, like so:

```
onPremScopeConfig.Tables["authors"].AddFilterColumn("state");
onPremScopeConfig.Tables["authors"].FilterClause = "[authors].[state] = 'UT'";
```

If I want it to synchronize in both directions based on the filter, it will need to be added to both scopes as they're provisioned on each end.

Go Forth and Spread the Data

Adding SQL Azure to the mix, whether a single instance or multiple ones, can really enhance the data availability and overall performance when synchronizing to nodes by adding that ever-important layer of indirection. Because it's SQL Azure, one gets the performance, scalability and reliability without all of the headaches of designing, provisioning and managing the infrastructure. Look for next month's column, where I'll expand on the implementation and show how Windows Azure can be added into the mix for synchronization using the latest Sync Framework 4.0 CTP released in October (bit.ly/dpyMP8). ■

JOSEPH FULTZ *is an architect at the Microsoft Technology Center in Dallas, where he works with both enterprise customers and ISVs designing and prototyping software solutions to meet business and market demands. He's spoken at events such as Tech·Ed and similar internal training events.*

THANKS to the following technical expert for reviewing this article: *David Browne*

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

Scalable, Long-Running Workflows with Windows Server AppFabric

Rafael Godinho

Business processes can cover a wide variety of application scenarios. They can include human workflows, business logic exposed through services, coordination of presentation layers and even application integration.

Although those scenarios are different, successful business processes have a few things in common. They need to be simple to build, use and modify. They need to be scalable to meet the changing needs of the business. And, often, they need some form of logging for status, compliance and debugging.

Workflows are a good example of business processes that have been codified into applications. They embody all of those elements I mentioned: human business needs, business logic, coordination between people and applications, and the ability to easily enter data and retrieve status. That's a lot for an application to do, and a lot to code, too.

Fortunately, the Microsoft .NET Framework and Windows Server AppFabric provide the tools you need to to create, deploy and configure trackable, long-running workflow services. You're probably already familiar with the .NET Framework. Windows Server AppFabric is a set of extensions for Windows Server that includes caching and hosting services for services based on Windows Communication Foundation (WCF) and Windows Workflow Foundation (WF).

In this article I'll walk you through the process of building a simple scalable workflow service using WCF, WF and Windows Server AppFabric.

This article discusses:

- Creating a workflow service
- Simplifying workflow persistence
- Workflow tracking
- Scaling the workflow service

Technologies discussed:

Windows Server AppFabric, Windows Communication Foundation, Windows Workflow Foundation

Code download available at:

code.msdn.microsoft.com/mag201101AppFabric

Creating a Workflow Service

To create a workflow service you need to combine two technologies: WCF and WF. This integration is seamless to the developer and is done using specific messaging activities to receive WCF messages in the workflow. The workflow is hosted in a workflow-specific WCF ServiceHost (the WorkflowServiceHost), which exposes WCF endpoints for these messages.. Among the messaging activities group, two of them can be used to receive information, allowing the workflow to accept messages from external clients as Web service calls: the Receive activity and the ReceiveAndSendReply template.

A Receive activity is used to receive information to be processed by the workflow. It can receive almost any kind of data, like built-in data types, application-defined classes or even XML-serializable types. **Figure 1** shows an example of a Receive activity on the workflow designer.

This type of activity has many properties, but four of them are extremely important to remember:

- CanCreateInstance is used to determine if the workflow runtime must create a new workflow instance to process the incoming message, or if it will reuse an existing one using correlation techniques. I'll discuss correlation in more detail later. You'll probably want to set it to true on the first Receive activity of the workflow.
- OperationName specifies the service operation name implemented by this Receive activity.
- Content indicates the data to be received by the service. This is much like WCF service operation contract parameters.
- ServiceContractName is used to create service contracts grouping service operations inside the generated Web Services Description Language (WSDL).

If used alone, the Receive activity implements a one-way message-exchange pattern, which is used to receive information from clients, but does not send them a reply. This kind of activity can also be used to implement a request-response pattern by associating it with a SendReply activity.

To help implement the request-response pattern, WF adds an option to the Visual Studio toolbox called ReceiveAndSendReply. When dropped on the workflow designer, it automatically creates a pair of pre-configured Receive and SendReplyToReceive activities within a Sequence activity (see **Figure 2**).

The idea behind the ReceiveAndSendReply template is to do some processing between the Receive and SendReplyToReceive actions. However, it's important to notice that persistence is not allowed between the Receive and SendReplyToReceive pair. A no-persist zone is created and lasts until both activities have completed, meaning if the workflow instance becomes idle, it won't persist even if the host is configured to persist workflows when they become idle. If an activity attempts to explicitly persist the workflow instance in the no-persist zone, a fatal exception is thrown, the workflow aborts and an exception is returned to the caller.

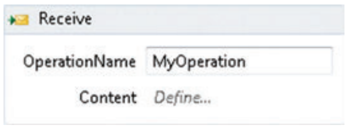

Receive
OperationName MyOperation
Content Define...

Figure 1 **A Receive Activity on the Workflow Designer**

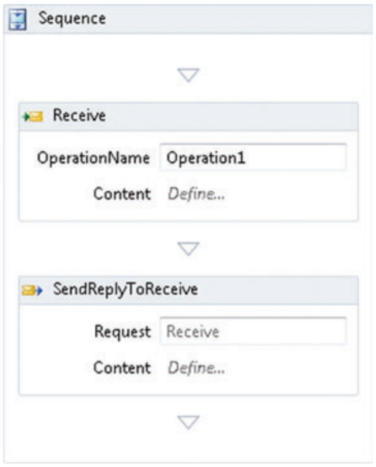

Sequence
Receive
OperationName Operation1
Content Define...
SendReplyToReceive
Request Receive
Content Define...

Figure 2 **ReceiveAndSendReply on the Workflow Designer**

Correlating Calls

Sometimes a business process can receive more than one external call. When that happens, a new workflow instance is created at the first call, its activities are executed and the workflow stays idle, waiting for subsequent calls. When a later call is made, the workflow instance leaves the idle state and continues to be executed.

In this way, the workflow runtime must have a way to use information received on later calls and distinguish between the previously created workflow instances to continue processing. Otherwise, it could call any instance, leaving the whole process consistency at risk. This is called correlation—you correlate subsequent calls to the pending workflow with which the call is associated.

A correlation is represented as an XPath query to identify particular data in a specific message. It can be initialized using an InitializeCorrelation activity or by adding a value to the CorrelationInitializers, a property of some activities, such as: Receive, SendReply, Send and ReceiveReply.

This initialization process can be done in code or using the workflow designer from Visual Studio 2010. Because Visual Studio has a wizard to help create the XPath query, it's the easier—and probably the preferable—way for most developers.

A possible scenario to use correlation is an expense report workflow. First, an employee submits the expense report data. Later, his manager can review the report and approve or deny the expenses (see **Figure 3**).

In this scenario the correlation is created when the workflow returns the response to the employee client application. To create a correlation you need some context-identifying information, like the expense report ID (which is probably a unique ID already). Then the workflow instance becomes idle, waiting for the manager to approve or deny the expense report. When the approval call is made by the manager client application, the workflow runtime correlates the received expense report ID with the previously created workflow instance to continue the process.

To create a correlation in Visual Studio 2010, first select in the workflow designer the activity where the correlation is going to

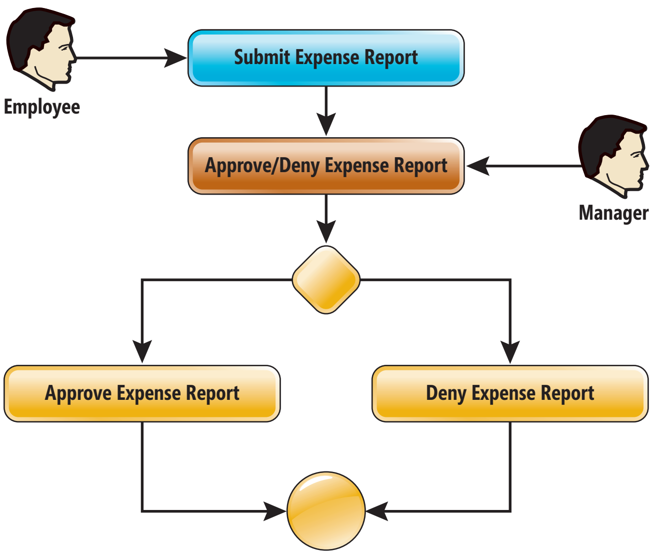
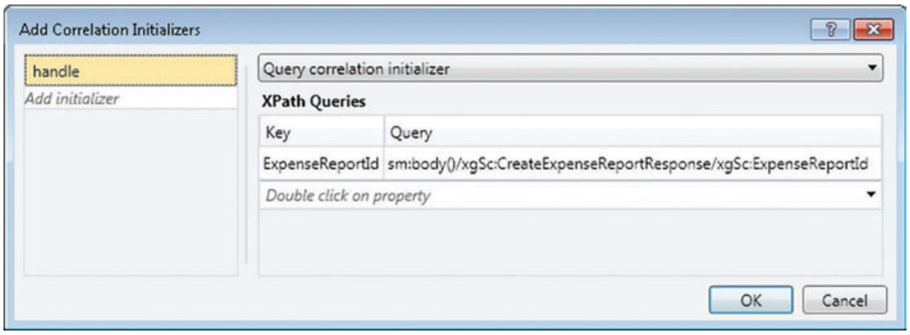


Figure 3 **Expense Report Sample Scenario**


Add Correlation Initializers

handle

Add initializer

Query correlation initializer

XPath Queries

Key	Query
ExpenseReportId	sm:body()/xgSc:CreateExpenseReportResponse/xgSc:ExpenseReportId
Double click on property	

Figure 4 **Setting the XPath Query Correlation**

be initialized. In my example, this is the activity that returns the expense report ID to the client. In the SendReply activity, I set the CorrelationInitializers property in the Properties window by clicking the ellipsis button. This displays the Add Correlation Initializers dialog box (see **Figure 4**) where you can configure the correlation.

Three items must be set: the correlation handle, the correlation type and the XPath Queries. The correlation handle is a variable the workflow runtime uses to store the correlation data and is automatically created by Visual Studio.

The next step is to set the correlation type. The .NET Framework has some types of correlation, but because I need to query part of the information exchanged with the client—in other words, a content-based correlation—my best option is to use the Query correlation initializer. After doing that, the XPath queries can be set to the expense report ID. When I click the arrow, Visual Studio checks the message content and shows me a list to select the appropriate information.

To continue the workflow after the expense approval is made, the correlation must be used by the corresponding Receive activity. This is done by setting the CorrelatesOn property. Just click the ellipsis button near the property in the Properties window to open the CorrelatesOn Definition dialog box (see **Figure 5**). From this dialog, the CorrelatesWith property needs to be set to the same handle used to initialize the correlation for the SendReplyToReceive activity, and the XPath Queries property must be set to the same key and expense report ID received on the expense report approval message.

WF comes with a set of general-purpose activities called Base Activity Library (BAL), some of which I've used to send and receive information here. Though they are useful, sometimes activities more related to business rules are needed. Based on the scenario I've discussed so far, there are three activities needed for submitting and approving expense reports: Create, Approve and Deny expense report. Because all of those activities are pretty similar, I'm only going to show the code of CreateExpenseReportActivity:

```
public sealed class CreateExpenseReportActivity
  : CodeActivity<int> {
  public InArgument<decimal> Amount { get; set; }
  public InArgument<string> Description { get; set; }
  protected override int Execute(CodeActivityContext context) {
    Data.ExpenseReportManager expenseReportManager =
      new Data.ExpenseReportManager();
    return expenseReportManager.CreateExpenseReport(
      Amount.Get(context), Description.Get(context));
  }
}
```

The activity receives the expense amount and description, both declared as InArgument. Most of the heavy lifting is done in the Execute method. It accesses a class that uses the Entity Framework to handle database access and to save the expense report information, and on the other end the Entity Framework returns the expense report ID. Because I only need to execute CLR code and don't need to interact with the WF runtime, the easiest option to create an activity is to inherit from CodeActivity. The complete workflow can be seen in **Figure 6**.

Hosting the Workflow Service

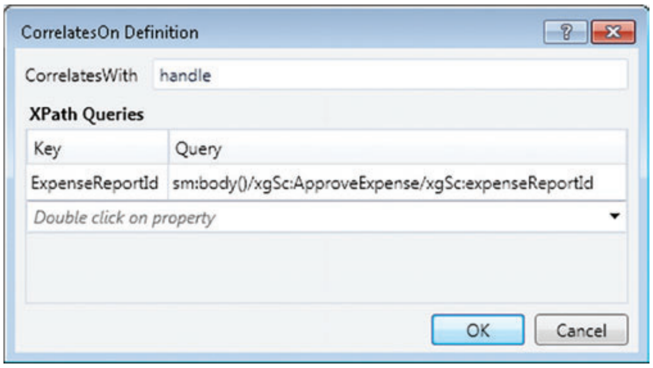
After the workflow service is created, you need to decide where it will run. The traditional choice has been to run it on your own hosting environment, IIS or Windows Process Activation Services (WAS). Another option, however, is to take advantage of Windows Server AppFabric, an enhancement to the Application Server role in Windows Server 2008 R2 for hosting, managing, securing and scaling services created with WCF or WF. You can also employ Windows Server AppFabric on PCs running Windows Vista or Windows 7 for development and testing.

Though IIS and WAS already support service hosting, Windows Server AppFabric offers a more useful and manageable environment that integrates WCF and WF features such as persistence and tracking with IIS Manager.

Simplified Workflow Persistence

Computers still have a limited set of resources to process all of your business processes, and there's no reason to waste computing resources on idle workflows. For long-running processes, you may have no control over the total amount of time from the beginning of the process to its end. It can take minutes, hours, days or even longer, and if it depends on external entities, such as other systems or end users, most of the time it can be idle simply waiting for a response.

WF provides a persistence framework capable of storing a durable capture of a workflow instance's state—independent of process or computer information—into instance stores. WF 4 already has a SQL Server instance store to be used out of the box. However, because WF is very extensible, I could create my own instance store to persist the workflow instance state if I wanted to. Once the work-


CorrelatesOn Definition

CorrelatesWith: handle

XPath Queries

Key	Query
ExpenseReportId	sm:body()/xgSc:ApproveExpense/xgSc:expenseReportId
Double click on property	

Figure 5 **CorrelatesOn Definition**

flow instance is idle and has been persisted, it can be unloaded to preserve memory and CPU resources, or eventually it could be moved from one node to another in a server farm.

Windows Server AppFabric has an easy way to set up and maintain integration with WF persistence features. The whole process is transparent to the workflow runtime, which delegates the persistence tasks to AppFabric, extending the default WF persistence framework.

The first step to configure persistence is to set up the SQL Server database using the Windows Server AppFabric Configuration Wizard or Windows PowerShell cmdlets. The wizard can create the persistence database if it doesn't exist, or just create the AppFabric schema. With the database already created, all the other steps are accomplished with IIS Manager.

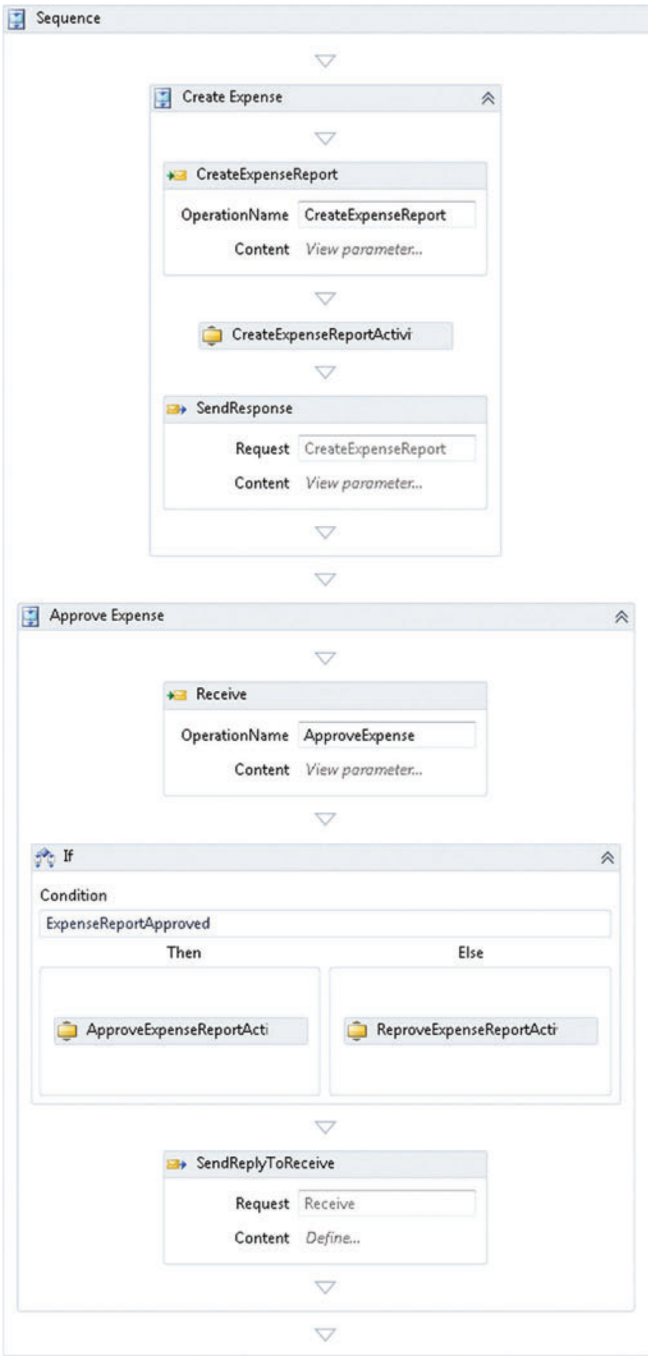
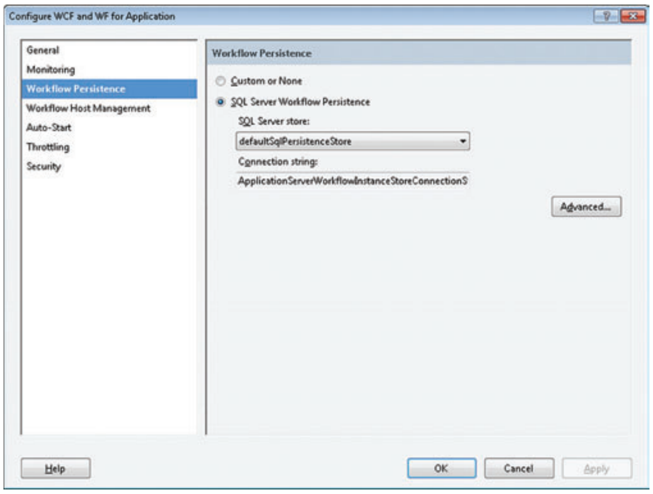


Figure 6 **Complete Expense Report Workflow**



Configure WCF and WF for Application

General
Monitoring
Workflow Persistence
Workflow Host Management
Auto-Start
Throttling
Security

Workflow Persistence

Custom or None [✓ SQL Server Workflow Persistence]

SQL Server store: defaultSqlPersistenceStore

Connection string: ApplicationServerWorkflowInstanceStoreConnectionS

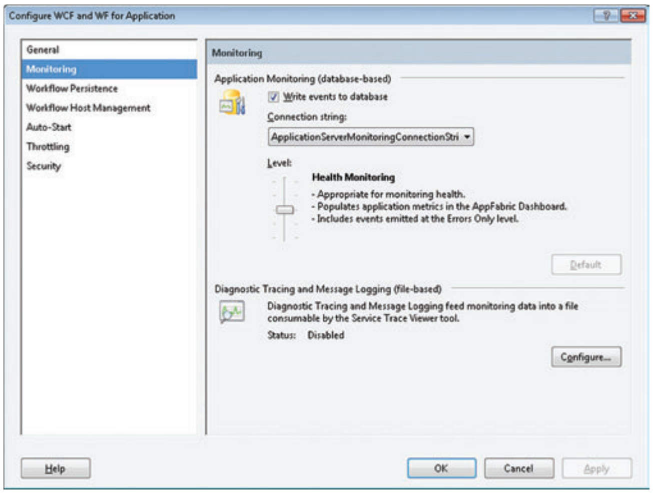
Figure 7 **Configuring Workflow Persistence**

In IIS Manager, right-click the node you want to configure (server, Web site or application) and choose Manage WCF and WF Services | Configure to open the Configure WCF and WF for Application dialog, then click Workflow Persistence (see **Figure 7**). You can see that you have the option to enable or disable workflow persistence.

You also have the option to set how long the workflow runtime will take to unload the workflow instance from memory and persist it on the database when the workflow becomes idle. The default value is 60 seconds. If you set the value to zero it will be persisted immediately. This is especially important for scaling out via a load balancer.

Workflow Tracking

Sometimes something can go wrong with processes that interact with external users and applications. Due to the detached nature of long-running processes, it can be even worse on those scenarios. When a problem occurs, as a developer you usually need to analyze a bunch of logs to discover what happened, how to reproduce it and, most important, how to correct the problem and keep the system up. If you



Configure WCF and WF for Application

General
Monitoring
Workflow Persistence
Workflow Host Management
Auto-Start
Throttling
Security

Monitoring

Application Monitoring (database-based)

[✓] Write events to database

Connection string: ApplicationServerMonitoringConnectionStri

Level:

Health Monitoring
- Appropriate for monitoring health.
- Populates application metrics in the AppFabric Dashboard.
- Includes events emitted at the Errors Only level.

Diagnostic Tracing and Message Logging (file-based)

Diagnostic Tracing and Message Logging feed monitoring data into a file consumable by the Service Trace Viewer tool.

Status: Disabled

Figure 8 **Enabling Tracking on Windows Server AppFabric**

use WF, you already get this kind of logging built into the framework.

The same way WF has an extensible framework to persist idle instances, it also has an extensible framework to provide visibility into workflow execution. This framework is called tracking, which transparently instruments a workflow, recording key events during its execution. Windows Server AppFabric uses this extensibility to improve the built-in WF tracking functionality, recording execution events on a SQL Server database.

The Windows Server AppFabric tracking configuration is similar to that used for persistence and can be accessed via either the Windows Server AppFabric Configuration Wizard or Windows PowerShell cmdlets. In the Configure WCF and WF for Application dialog discussed earlier, click Monitoring. Now you can choose to enable or disable the tracking and also the tracking level, as shown in **Figure 8**.

While configuring tracking in Windows Server AppFabric, you can choose five monitoring levels:

- Off has the same effect as disabling monitoring and is best used in scenarios that need minimal tracking overhead.
- Error Only gives visibility to only critical events like errors and warnings. This mode is best for high-performance scenarios that need only minimal error logging.
- Health Monitoring is the default monitoring level and contains all the data captured at the Errors Only level, plus some additional processing data.
- End-to-End Monitoring contains all data from level Health Monitoring plus additional information to reconstruct the entire message flow. This is used in scenarios where a service calls another service.
- Troubleshooting, as the name suggests, is the most verbose level and is useful in scenarios where an application is in an unhealthy state and needs to be fixed.

Scaling the Workflow Service

Because Windows Server AppFabric extends the Application Server Role from Windows Server, it inherits the highly scalable infrastructure from its predecessor and can be run on a server farm behind a network load balancer (NLB). You've also seen that it has the ability to persist and track workflow instances when needed. As a result, Windows Server AppFabric is an excellent choice to host long-running workflow processes and support a great number of requests from clients.

An example of a workflow service scalable environment can be seen in **Figure 9**. It has two Windows Server AppFabric instances, both running copies of the same workflow definition. NLB routes requests to the available AppFabric instance.

On the expense report scenario, when a client first accesses the service to create an expense report, the balancer redirects the requests to an available Windows Server AppFabric instance, which saves the expense data in the database, returns the generated ID to the client and, because the workflow becomes idle waiting for the expense approval from the workflow runtime, persists the running instance in the database.

Later, when the client application accesses the service to approve or deny the expense report, the NLB redirects the request to an available Windows Server AppFabric instance (it can be a different server from the first service call), and the server correlates the request and restores the workflow instance from the persistence database. Now, the instance in memory continues processing, saves the approval on the database and returns to the client when it's done.

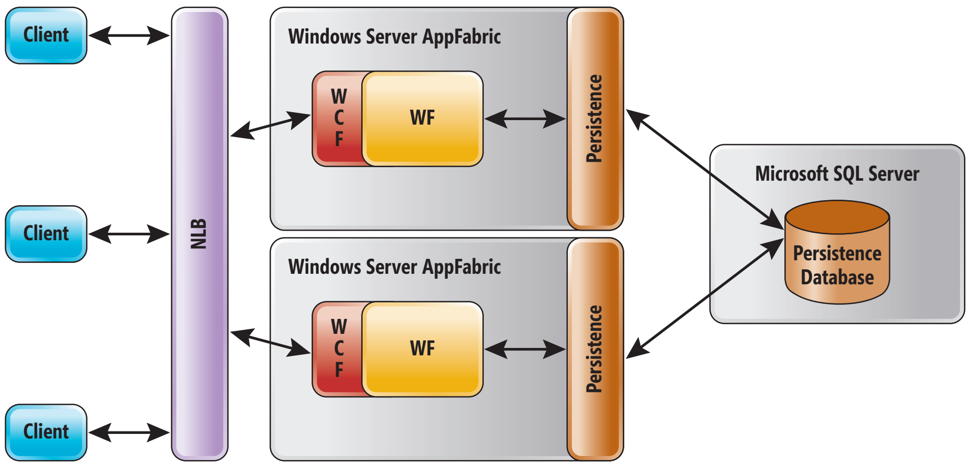


Figure 9 **Workflows in a Scalable Environment**

Closing Notes

As you've seen, the use of workflow services with correlation, persistence and tracking on a load-balancing environment is a powerful technique for running those services in a scalable manner. The combination of these features can increase operations productivity, allowing proactive actions on running services, and spreading workflows across threads, processes and even machines. This allows developers to create a fully scalable solution that's ready to run on a single machine—or even large server farms—with no worries about infrastructure complexity.

For further information about designing workflows with WCF and WF, be sure to read Leon Welicki's article, "Visual Design of Workflows with WCF and WF 4," from the May 2010 issue of *MSDN Magazine* (msdn.microsoft.com/magazine/ff646977). And for a deeper discussion of long-running processes and workflow persistence, see Michael Kennedy's article, "Web Apps That Support Long-Running Operations," from the January 2009 issue (msdn.microsoft.com/magazine/dd296718).

For details about Windows Server AppFabric, see the Windows Server Developer Center at msdn.microsoft.com/windowsserver/ee695849. ■

RAFAEL GODINHO *is an ISV developer evangelist at Microsoft Brazil helping local partners adopt Microsoft technology. You can contact Godinho through his blog at blogs.msdn.com/rafaelgodinho.*

THANKS to the following technical experts for reviewing this article: *Dave Cliffe, Ron Jacobs and Leon Welicki*

LEADTOOLS® version 17

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

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.**

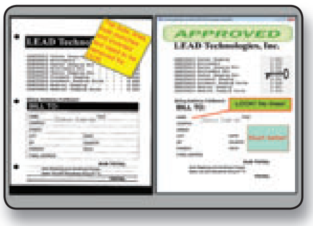
- •**Silverlight:** 100% pure Silverlight 3, 4 and Windows Phone Imaging SDK.
- •**Image Formats & Compression:** Supports 150+ image formats and compressions including TIFF, EXIF, PDF, JPEG2000, JBIG2 and CCITT G3/G4.
- •**Scanning:** TWAIN & WIA (32 & 64-bit), auto-detect optimum driver settings for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

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

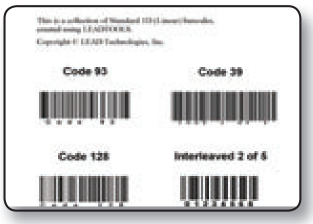
Free 60 Day Evaluation!
www.leadtools.com/msdn
(800) 637-1840

LEAD TECHNOLOGIES INCORPORATED

Document

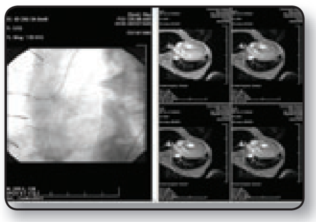


Barcode

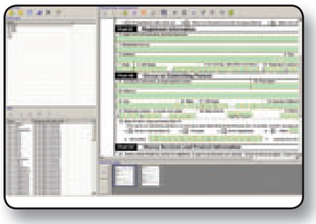


Multimedia

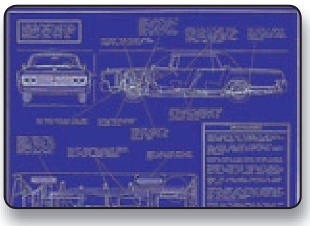
DICOM Medical



Form Recognition & Processing



Vector



Authoring Control Flow Activities in WF 4

Leon Welicki

Control flow refers to the way in which the individual instructions in a program are organized and executed. In Windows Workflow Foundation 4 (WF 4), control flow activities govern the execution semantics of one or more child activities. Some examples in the WF 4 activity toolbox include Sequence, Parallel, If, ForEach, Pick, Flowchart and Switch, among others.

The WF runtime doesn't have first-class knowledge of any control flow like Sequence or Parallel. From its perspective, everything is just activities. The runtime only enforces some simple rules (for example, "an activity can't complete if any child activities are still running"). WF control flow is based on hierarchy—a WF program is a tree of activities.

Control flow options in WF 4 are not limited to the activities shipped in the framework. You can write your own and use them in combination with the ones provided in the box, and that's what this article will discuss. You'll learn how to write your own control flow activities following a "crawl, walk, run" approach: we'll start with a very simple control flow activity and add richness as we progress, ending up with a new and useful control flow activity. The source code for all of our examples is available for download.

But first, let's start with some basic concepts about activities to set some common ground.

Activities

Activities are the basic unit of execution in a WF program; a workflow program is a tree of activities that are executed by the WF runtime. WF 4 includes more than 35 activities, a comprehensive set that can be used to model processes or create new activities. Some of those activities govern the semantics of how other activities are executed (such as Sequence, Flowchart, Parallel and ForEach), and are known as *composite* activities. Others perform a single atomic task (WriteLine, InvokeMethod and so forth). We call these *leaf* activities.

WF activities are implemented as CLR types, and as such they derive from other existing types. You can author activities visually and declaratively using the WF designer, or imperatively writing CLR code. The base types available to create your own custom activity are defined in the activities type hierarchy in **Figure 1**. You'll find a detailed explanation of this type hierarchy in the MSDN Library at msdn.microsoft.com/library/dd560893.

In this article I'll focus on activities that derive from NativeActivity, the base class that provides access to the full breadth of the WF

This article discusses:

- The activities type hierarchy in Windows Workflow Foundation 4
- Composite and leaf activities
- Bookmarks and execution properties
- GoTo activities

Technologies discussed:

Windows Workflow Foundation 4.0

runtime. Control flow activities are composite activities that derive from the NativeActivity type because they need to interact with the WF runtime. Most commonly this is to schedule other activities (for example, Sequence, Parallel or Flowchart), but it may also include implementing custom cancellation using CancellationScope or Pick, creating bookmarks with Receive and persisting using Persist.

The activity data model defines a crisp model for reasoning about data when authoring and consuming activities. Data is defined using arguments and variables. Arguments are the binding terminals of an activity and define its public signature in terms of what data can be passed to the activity (input arguments) and what data will be returned by the activity when it completes its execution (output arguments). Variables represent temporary storage of data.

Activity authors use arguments to define the way data flows in and out of an activity, and they use variables in a couple of ways:

- To expose a user-editable collection of variables on an activity definition that can be used to share variables among multiple activities (such as a Variables collection in Sequence and Flowchart).
- To model the internal state of an activity.

Workflow authors use arguments to bind activities to the environment by writing expressions, and they declare variables at different scopes in the workflow to share data between activities. Together, variables and arguments combine to provide a predictable model of communication between activities.

Now that I've covered some core activity basics, let's start with the first control flow activity.

A Simple Control Flow Activity

I'll begin by creating a very simple control flow activity called ExecuteIfTrue. There's not much to this activity: It executes a contained activity if a condition is true. WF 4 provides an If activity that includes Then and Else child activities; often we only want to provide the Then, and the Else is just overhead. For those cases we want an activity that executes another activity based on the value of a Boolean condition.

Here's how this activity should work:

- The activity user must provide a Boolean condition. This argument is required.
- The activity user can provide a body—the activity to be executed if the condition is true.
- At execution time: If the condition is true and the body is not null, execute the body.

Here's an implementation for an ExecuteIfTrue activity that behaves in exactly this way:

```
public class ExecuteIfTrue : NativeActivity
{
  [RequiredArgument]
  public InArgument<bool> Condition { get; set; }

  public Activity Body { get; set; }

  public ExecuteIfTrue() { }

  protected override void Execute(NativeActivityContext context)
  {
    if (context.GetValue(this.Condition) && this.Body != null)
      context.ScheduleActivity(this.Body);
  }
}
```

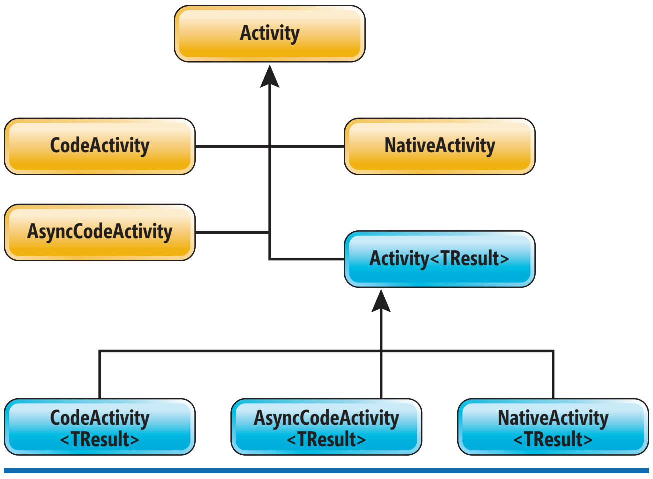


Figure 1 **Activity Type Hierarchy**

This code is very simple, but there's more here than meets the eye. ExecuteIfTrue executes a child activity if a condition is true, so it needs to schedule another activity. Therefore, it derives from NativeActivity because it needs to interact with the WF runtime to schedule children.

Once you've decided the base class for an activity, you have to define its public signature. In ExecuteIfTrue, this consists of a Boolean input argument of type InArgument<bool> named Condition that holds the condition to be evaluated, and a property of type Activity named Body with the activity to be executed if the condition is true. The Condition argument is decorated with the RequiredArgument attribute that indicates to the WF runtime that it must be set with an expression. The WF runtime will enforce this validation when preparing the activity for execution:

```
[RequiredArgument]
public InArgument<bool> Condition { get; set; }

public Activity Body { get; set; }
```

The most interesting piece of code in this activity is the Execute method, which is where the "action" happens. All NativeActivities *must* override this method. The Execute method receives a NativeActivityContext argument, which is our point of interaction with the WF runtime as activity authors. In ExecuteIfTrue, this context is used to retrieve the value of the Condition argument (context.GetValue(this.Condition)) and to schedule the Body using the ScheduleActivity method. Notice that I say *schedule* and not *execute*. The WF runtime does not execute the activities right away; instead it adds them to a list of work items to be scheduled for execution:

```
protected override void Execute(NativeActivityContext context)
{
    if (context.GetValue(this.Condition) && this.Body != null)
        context.ScheduleActivity(this.Body);
}
```

Note as well that the type has been designed following the create-set-use pattern. The XAML syntax is based on this pattern for designing types, where the type has a public default constructor and public read/write properties. This means the type will be XAML serialization-friendly.

The following code snippet shows how to use this activity. In this example, if the current day is Saturday, you write the string "Rest!" to the console:

Figure 2 **The SimpleSequence Activity**

```
public class SimpleSequence : NativeActivity
{
  // Child activities collection
  Collection<Activity> activities;
  Collection<Variable> variables;

  // Pointer to the current item in the collection being executed
  Variable<int> current = new Variable<int>() { Default = 0 };

  public SimpleSequence() { }

  // Collection of children to be executed sequentially by SimpleSequence
  public Collection<Activity> Activities
  {
    get
    {
      if (this.activities == null)
        this.activities = new Collection<Activity>();

      return this.activities;
    }
  }

  public Collection<Variable> Variables
  {
    get
    {
      if (this.variables == null)
        this.variables = new Collection<Variable>();

      return this.variables;
    }
  }

  protected override void CacheMetadata(NativeActivityMetadata metadata)
  {
    metadata.SetChildrenCollection(this.activities);
    metadata.SetVariablesCollection(this.variables);
    metadata.AddImplementationVariable(this.current);
  }

  protected override void Execute(NativeActivityContext context)
  {
    // Schedule the first activity
    if (this.Activities.Count > 0)
      context.ScheduleActivity(this.Activities[0], this.
OnChildCompleted);
  }

  void OnChildCompleted(NativeActivityContext context, ActivityInstance
completed)
  {
    // Calculate the index of the next activity to scheduled
    int currentExecutingActivity = this.current.Get(context);
    int next = currentExecutingActivity + 1;

    // If index within boundaries...
    if (next < this.Activities.Count)
    {
      // Schedule the next activity
      context.ScheduleActivity(this.Activities[next], this.
OnChildCompleted);

      // Store the index in the collection of the activity executing
      this.current.Set(context, next);
    }
  }
}
```

```
var act = new ExecuteIfTrue
{
  Condition = new InArgument<bool>(c => DateTime.Now.DayOfWeek == DayOfWeek.Tuesday),
  Body = new WriteLine { Text = "Rest!" }
};

WorkflowInvoker.Invoke(act);
```

The first control flow activity has been created in 15 lines of code. But don't be fooled by the simplicity of that code—it's actually a fully functional control flow activity!

Scheduling Multiple Children

The next challenge is to write a simplified version of the Sequence activity. The goal of this exercise is to learn how to write a control flow activity that schedules multiple child activities and executes in multiple episodes. This activity is almost functionally equivalent to the Sequence shipped in the product.

Here's how this activity should work:

- The activity user must provide a collection of children to be executed sequentially through the Activities property.
- At execution time:
  - The activity contains an internal variable with the index of the last item in the collection that has been executed.
  - If there are items in the children collection, schedule the first child.
  - When the child completes:
    - Increment the index of the last item executed.
    - If the index is still within the boundaries of the children collection, schedule the next child.
    - Repeat.

The code in **Figure 2** implements a SimpleSequence activity that behaves exactly as described.

Again, a fully functional control flow activity has been written in just a few lines of codes—in this case, about 50 lines. The code is simple, but it introduces some interesting concepts.

SimpleSequence executes a collection of child activities in sequential order, so it needs to schedule other activities. Therefore, it derives from NativeActivity because it needs to interact with the runtime to schedule children.

The next step is to define the public signature for SimpleSequence. In this case it consists of a collection of activities (of type Collection<Activity>) exposed through the Activities property, and a collection of variables (of type Collection<Variable>) exposed through the Variables property. Variables allow sharing of data among all child activities. Note that these properties only have "getters" that expose the collections using a "lazy instantiation" approach (see **Figure 3**), so accessing these properties never results in a null reference. This makes these properties compliant with the create-set-use pattern.

There's one private member in the class that's not part of the signature: a Variable<int> named "current" that holds the index of the activity being executed:

```
// Pointer to the current item in the collection being executed
Variable<int> current = new Variable<int>() { Default = 0 };
```

Because this information is part of the internal execution state for SimpleSequence, you want to keep it private and not expose it to users of SimpleSequence. You also want it to be saved and restored when the activity is persisted. You use an ImplementationVariable for this.

Implementation variables are variables that are internal to an activity. They're intended to be consumed by the activity author, not the activity user. Implementation variables are persisted when the activity is persisted and restored when the activity is reloaded, without requiring any work on our part. To make this clear—and continuing with the Sequence example—if an instance of SimpleSequence is persisted, when it wakes up it will "remember" the index of the last activity that has been executed.

The WF runtime can't automatically know about implementation variables. If you want to use an ImplementationVariable in an activity, you need to explicitly inform the WF runtime. You do this during CacheMetadata method execution.

Despite its rather frightening name, CacheMetadata is not that difficult. Conceptually, it's actually simple: It's the method where an activity "introduces itself" to the runtime. Think about the If activity for a moment. In CacheMetadata this activity would say: *"Hi, I'm the If activity and I have an input argument named Condition, and two children: Then and Else."* In the SimpleSequence case, SimpleSequence is saying: *"Hi, I'm SimpleSequence and I have a collection of child activities, a collection of variables and an implementation variable."* There's no more than that in the SimpleSequence code for CacheMetadata:

```
protected override void CacheMetadata(NativeActivityMetadata metadata)
{
  metadata.SetChildrenCollection(this.activities);
  metadata.SetVariablesCollection(this.variables);
  metadata.AddImplementationVariable(this.current);
}
```

The default implementation of CacheMetadata uses reflection to get this data from the activity. In the ExecuteIfTrue example, I didn't implement CacheMetadata and relied on the default implementation to reflect on public members. For SimpleSequence, in contrast, I did need to implement it because the default implementation can't "guess" about my desire to use implementation variables.

The next interesting piece of code in this activity is the Execute method. In this case, if there are activities in the collection, you tell the WF runtime: *"Please execute the first activity in the collection of activities and, when you're done, invoke the OnChildCompleted method."* You say this in WF terms using NativeActivityContext.ScheduleActivity. Notice that when you schedule an activity, you supply a second argument that's a CompletionCallback. In simple terms, this is a method that will be called once the activity execution is completed. Again, it's important to be aware of the difference between scheduling and execution. The CompletionCallback won't be called when the activity is scheduled; it will be called when the scheduled activity execution has been completed:

```
protected override void Execute(NativeActivityContext context)
{
  // Schedule the first activity
  if (this.Activities.Count > 0)
    context.ScheduleActivity(this.Activities[0], this.OnChildCompleted);
}
```

The OnChildCompleted method is the most interesting part of this activity from a learning perspective, and it's actually the main reason I've included SimpleSequence in this article. This method gets the next activity in the collection and schedules it. When the next child is scheduled, a CompletionCallback is provided, which in this case points to this same method. Thus, when a child completes, this method will execute again to look for the next child and execute it. Clearly, execution is happening in pulses or episodes. Because workflows can be persisted and unloaded from memory, there can be a large time difference between two pulses of execution. Moreover, these pulses can be executed in different threads, processes or even machines (as persisted instances of a workflow can be reloaded in a different process or machine). Learning how to program for multiple pulses of execution is one of the biggest challenges in becoming an expert control flow activity author:

```
void OnChildCompleted(NativeActivityContext context, ActivityInstance
completed)
{
  // Calculate the index of the next activity to scheduled
  int currentExecutingActivity = this.current.Get(context);
  int next = currentExecutingActivity + 1;

  // If index within boundaries...
  if (next < this.Activities.Count)
  {
    // Schedule the next activity
    context.ScheduleActivity(this.Activities[next], this.OnChildCompleted);

    // Store the index in the collection of the activity executing
    this.current.Set(context, next);
  }
}
```

The following code snippet shows how to use this activity. In this example, I'm writing three strings to the console ("Hello," "Workflow" and "!"):

```
var act = new SimpleSequence()
{
  Activities =
  {
    new WriteLine { Text = "Hello" },
    new WriteLine { Text = "Workflow" },
    new WriteLine { Text = "!" }
  }
};

WorkflowInvoker.Invoke(act);
```

I've authored my own SimpleSequence! Now it's time to move on to the next challenge.

Implementing a New Control Flow Pattern

Next, I'll create a complex control flow activity. As I mentioned earlier, you're not limited to the control flow activities shipped in WF 4. This section demonstrates how to build your own control flow activity to support a control flow pattern that's not supported out-of-the-box by WF 4.

The new control flow activity will be called Series. Its goal is simple: to provide a Sequence with support for GoTos, where the

Figure 3 **A Lazy Instantiation Approach**

```
public Collection<Activity> Activities
{
  get
  {
    if (this.activities == null)
      this.activities = new Collection<Activity>();

    return this.activities;
  }
}

public Collection<Variable> Variables
{
  get
  {
    if (this.variables == null)
      this.variables = new Collection<Variable>();

    return this.variables;
  }
}
```

next activity to be executed can be manipulated either explicitly from inside the workflow (through a GoTo activity) or from the host (by resuming a well-known bookmark).

To implement this new control flow, I'll need to author two activities: Series, a composite activity that contains a collection of activities and executes them sequentially (but allows jumps to any item in the sequence), and GoTo, a leaf activity that I'll use inside of Series to explicitly model the jumps.

To recap, I'll enumerate the goals and requirements for the custom control activity:

1. It's a Sequence of activities.
2. It can contain GoTo activities (at any depth), which change the point of execution to any direct child of the Series.
3. It can receive GoTo messages from the outside (for example, from a user), which can change the point of execution to any direct child of the Series.

I'll start by implementing the Series activity. Here's the execution semantics in simple terms:

- The activity user must provide a collection of children to be executed sequentially through the Activities property.
- In the execute method:
  - Create a bookmark for the GoTo in a way that's available to child activities.
  - The activity contains an internal variable with the activity instance being executed.
  - If there are items in the children collection, schedule the first child.
  - When the child completes:
    - Lookup the completed activity in the Activities collection.
    - Increment the index of the last item executed.
    - If the index is still within the boundaries of the children collection, schedule the next child.
    - Repeat.
- If the GoTo bookmark is resumed:
  - Get the name of the activity we want to go to.
  - Find that activity in the activities collection.
  - Schedule the target activity in the set for execution and register a completion callback that will schedule the next activity.
  - Cancel the activity that's currently executing.
  - Store the activity that's currently being executed in the "current" variable.

The code example in **Figure 4** shows the implementation for a Series activity that behaves exactly as described.

Some of this code will look familiar from the previous examples. I'll discuss the implementation of this activity.

Series derives from NativeActivity because it needs to interact with the WF runtime to schedule child activities, create bookmarks, cancel children and use execution properties.

As before, the next step is to define the public signature for Series. As in SimpleSequence, there are Activities and Variables collection properties. There's also a string input argument named BookmarkName (of type InArgument<string>), with the name of the bookmark to be created for host resumption. Again, I'm following the create-set-use pattern in the activity type.

Series has a private member named "current" that contains the ActivityInstance being executed, instead of just a pointer to an item in a collection, as in SimpleSequence. Why is current a Variable<ActivityInstance> and not a Variable<int>? Because I need to get ahold of the currently executing child later in this activity during the GoTo method. I'll explain the actual details later; the important thing to understand now is that I'll have an implementation variable that holds the activity instance being executed:

```
Variable<ActivityInstance> current = new Variable<ActivityInstance>();
```

In CacheMetadata you provide runtime information about your activity: the children and variables collections, the implementation variable with the current activity instance and the bookmark name argument. The only difference from the previous example is that I'm manually registering the BookmarkName input argument within the WF runtime—adding a new instance of RuntimeArgument to the activity metadata:

```
protected override void CacheMetadata(NativeActivityMetadata metadata)
{
  metadata.SetVariablesCollection(this.Variables);
  metadata.SetChildrenCollection(this.Activities);
  metadata.AddImplementationVariable(this.current);
  metadata.AddArgument(new RuntimeArgument("BookmarkName",
                                        typeof(string),
ArgumentDirection.In));
}
```

The next new thing is the CanInduceIdle property overload. This is just more metadata that the activity provides to the WF runtime. When this property returns true, I'm telling the runtime that this activity can cause the workflow to become idle. I need to override this property and return true for activities that create bookmarks, as they'll make the workflow go idle waiting for its resumption. The default value for this property is false. If this property returns false and we create a bookmark, I'll have an InvalidOperationException exception when executing the activity:

```
protected override bool CanInduceIdle { get { return true; } }
```

Things get more interesting in the Execute method, where I create a bookmark (internalBookmark) and store it in an execution property. Before going further, though, let me introduce bookmarks and execution properties.

Bookmarks are the mechanism by which an activity can passively wait to be resumed. When an activity wishes to "block" pending a certain event, it registers a bookmark and then returns an execution status of continuing. This signals the runtime that although the activity's execution is not complete, it doesn't have any more work to do as part of the current work item. When you use bookmarks, you can author your activities using a form of reactive execution: when the bookmark is created the activity yields, and when the bookmark is resumed a block of code (the bookmark resumption callback) is invoked in reaction to the bookmark resumption.

Unlike programs directly targeting the CLR, workflow programs are hierarchically scoped trees that execute in a thread-agnostic environment. This implies that the standard thread local storage (TLS) mechanisms can't be directly leveraged to determine what context is in scope for a given work item. The workflow execution context introduces *execution properties* to an activity's environment, so that an activity can declare properties that are in scope for its sub-tree and share them with its children. As a result, an activity can share data with its descendants through these properties.

Figure 4 **The Series Activity**

```
public class Series : NativeActivity
{
  internal static readonly string GotoPropertyName =
    "Microsoft.Samples.CustomControlFlow.Series.Goto";

  // Child activities and variables collections
  Collection<Activity> activities;
  Collection<Variable> variables;

  // Activity instance that is currently being executed
  Variable<ActivityInstance> current = new Variable<ActivityInstance>();

  // For externally initiated goto's; optional
  public InArgument<string> BookmarkName { get; set; }

  public Series() { }

  public Collection<Activity> Activities
  {
    get {
      if (this.activities == null)
        this.activities = new Collection<Activity>();

      return this.activities;
    }
  }

  public Collection<Variable> Variables
  {
    get {
      if (this.variables == null)
        this.variables = new Collection<Variable>();

      return this.variables;
    }
  }

  protected override void CacheMetadata(NativeActivityMetadata metadata)
  {
    metadata.SetVariablesCollection(this.Variables);
    metadata.SetChildrenCollection(this.Activities);
    metadata.AddImplementationVariable(this.current);
    metadata.AddArgument(new RuntimeArgument("BookmarkName",
typeof(string),
                                              ArgumentDirection.In));
  }

  protected override bool CanInduceIdle { get { return true; } }

  protected override void Execute(NativeActivityContext context)
  {
    // If there activities in the collection...
    if (this.Activities.Count > 0)
    {
      // Create a bookmark for signaling the GoTo
      Bookmark internalBookmark = context.CreateBookmark(this.Goto,
                BookmarkOptions.MultipleResume | BookmarkOptions.
NonBlocking);

      // Save the name of the bookmark as an execution property
      context.Properties.Add(GotoPropertyName, internalBookmark);

      // Schedule the first item in the list and save the resulting
      // ActivityInstance in the "current" implementation variable
      this.current.Set(context, context.ScheduleActivity(this.Activities[0],
                                  this.OnChildCompleted));

      // Create a bookmark for external (host) resumption
      if (this.BookmarkName.Get(context) != null)
        context.CreateBookmark(this.BookmarkName.Get(context), this.Goto,
            BookmarkOptions.MultipleResume | BookmarkOptions.NonBlocking);
    }
  }

  void Goto(NativeActivityContext context, Bookmark b, object obj)
  {
    // Get the name of the activity to go to
    string targetActivityName = obj as string;

    // Find the activity to go to in the children list
    Activity targetActivity = this.Activities
                              .Where<Activity>(a =>
                                     a.DisplayName.
Equals(targetActivityName))
                              .Single();

    // Schedule the activity
    ActivityInstance instance = context.ScheduleActivity(targetActivity,
                                                      this.
OnChildCompleted);

    // Cancel the activity that is currently executing
    context.CancelChild(this.current.Get(context));

    // Set the activity that is executing now as the current
    this.current.Set(context, instance);
  }

  void OnChildCompleted(NativeActivityContext context, ActivityInstance
completed)
  {
    // This callback also executes when cancelled child activities complete
    if (completed.State == ActivityInstanceState.Closed)
    {
      // Find the next activity and execute it
      int completedActivityIndex = this.Activities.IndexOf(completed.Activity);
      int next = completedActivityIndex + 1;

      if (next < this.Activities.Count)
          this.current.Set(context,
                           context.ScheduleActivity(this.Activities[next],
                           this.OnChildCompleted));
    }
  }
}
```

Now that you know about bookmarks and execution properties, let's get back to the code. What I'm doing at the beginning of the Execute method is creating a bookmark (using context.CreateBookmark) and saving it into an execution property (using context.Properties.Add). This bookmark is a multiple-resume bookmark, meaning it can be resumed multiple times and it will be available while its parent activity is in an executing state. It's also NonBlocking, so it won't prevent the activity from completing once it's done with its job. When that bookmark is resumed, the GoTo method will be called because I provided a BookmarkCompletionCallback to CreateBookmark (the first parameter). The reason to save it into an execution property is to make it available to all child activities. (You'll see later how the GoTo activity uses this bookmark.) Notice that execution properties have names. Because that name is a string, I defined a constant (GotoPropertyName) with the name for the property in the activity. That name follows a fully qualified name approach. This is a best practice:

```
internal static readonly string GotoPropertyName =
                                "Microsoft.Samples.CustomControlFlow.
Series.Goto";

...
...

// Create a bookmark for signaling the GoTo
Bookmark internalBookmark = context.CreateBookmark(this.Goto,
                       BookmarkOptions.MultipleResume | BookmarkOptions.
NonBlocking);

// Save the name of the bookmark as an execution property
context.Properties.Add(GotoPropertyName, internalBookmark);
```

Once I've declared the bookmark, I'm ready to schedule my first activity. I'm already familiar with this, because I did it in my previous

Figure 5 **The GoTo Activity**

```
public class GoTo : NativeActivity
{
  public GoTo()
  { }

  [RequiredArgument]
  public InArgument<string> TargetActivityName { get; set; }

  protected override bool CanInduceIdle { get { return true; } }

  protected override void Execute(NativeActivityContext context)
  {
    // Get the bookmark created by the parent Series
    Bookmark bookmark = context.Properties.Find(Series.GotoPropertyName)
as Bookmark;

    // Resume the bookmark passing the target activity name
    context.ResumeBookmark(bookmark, this.TargetActivityName.
Get(context));

    // Create a bookmark to leave this activity idle waiting when it does
    // not have any further work to do. Series will cancel this activity
    // in its GoTo method
    context.CreateBookmark("SyncBookmark");
  }

}
```

activities. I'll schedule the first activity in the collection and tell the runtime to invoke the OnChildCompleted method when the activity is done (as I did in SimpleSequence). Context.ScheduleActivity returns an ActivityInstance that represents an instance of an activity being executed, which I assign to our current implementation variable. Let me clarify this a bit. Activity is the definition, like a class; ActivityInstance is the actual instance, like an object. We can have multiple ActivityInstances from the same Activity:

```
// Schedule the first item in the list and save the resulting
// ActivityInstance in the "current" implementation variable
this.current.Set(context, context.ScheduleActivity(this.Activities[0],
                                              this.OnChildCompleted));
```

Finally, we create a bookmark that can be used by the host to jump to any activity within the series. The mechanics for this are simple: Because the host knows the name of the bookmark, it can resume it with a jump to any activity within the Series:

```
// Create a bookmark for external (host) resumption
 if (this.BookmarkName.Get(context) != null)
     context.CreateBookmark(this.BookmarkName.Get(context), this.Goto,
                         BookmarkOptions.MultipleResume |
BookmarkOptions.NonBlocking);
```

The OnChildCompleted method should be straightforward now, as it's very similar to the one in SimpleSequence: I look up the next element in the activities collection and schedule it. The main difference is that I only schedule the next activity if the current activity successfully completed its execution (that is, reached the closed state and hasn't been canceled or faulted).

The GoTo method is arguably the most interesting. This is the method that gets executed as the result of the GoTo bookmark being resumed. It receives some data as input, which is passed when the bookmark is resumed. In this case, the data is the name of the activity we want to go to:

```
void Goto(NativeActivityContext context, Bookmark b, object data)
{
  // Get the name of the activity to go to
  string targetActivityName = data as string;

  ...
 }
```

The target activity name is the DisplayName property of the activity. I look up the requested activity definition in the "activities" collection. Once I find the requested activity, I schedule it, indicating that when the activity is done, the OnChildCompleted method should be executed:

```
// Find the activity to go to in the children list
Activity targetActivity = this.Activities
                              .Where<Activity>(a =>
                                       a.DisplayName.
Equals(targetActivityName))
                              .Single();
// Schedule the activity
ActivityInstance instance = context.ScheduleActivity(targetActivity,
                                                    this.
OnChildCompleted);
```

Next, I cancel the activity instance that's currently being executed and set the current activity being executed to the ActivityInstance scheduled in the previous step. For both these tasks I use the "current" variable. First, I pass it as a parameter of the CancelChild method of NativeActivityContext, and then I update its value with the ActivityInstance that has been scheduled in the previous block of code:

```
// Cancel the activity that is currently executing
context.CancelChild(this.current.Get(context));

// Set the activity that is executing now as the current
this.current.Set(context, instance);
```

The GoTo Activity

The GoTo activity can be used only inside of a Series activity to jump to an activity in its Activities collection. It's similar to a GoTo statement in an imperative program. The way it works is very simple: It resumes the GoTo bookmark created by the Series activity in which it's contained, indicating the name of the activity we want to go to. When the bookmark is resumed, the Series will jump to the activity indicated.

The GoTo method is arguably the most interesting.

Here's a simple description of the execution semantics:

- The activity user must provide a string TargetActivityName. This argument is required.
- At execution time:
  - The GoTo activity will locate the "GoTo" bookmark created by the Series activity.
  - If the bookmark is found, it will resume it, passing the TargetActivityName.
  - It will create a synchronization bookmark, so the activity does not complete.
    - It will be cancelled by the Series.

The code in **Figure 5** shows the implementation for a GoTo activity that behaves exactly as described.

GoTo derives from NativeActivity because it needs to interact with the WF runtime to create and resume bookmarks and use execution properties. Its public signature consists of the TargetActivityName string input argument that contains the name of the activity we want to jump to. I decorated that argument with

DESIGN

Design Applications That Help Run the Business

Our xamMap™ control in Silverlight and WPF lets you map out any geospatial data like this airplane seating app to manage your business. Come to infragistics.com to try it today!

NetAdvantage® ULTIMATE
for ASP.NET, Windows Forms, WPF, Silverlight, WPF Data Visualization, Silverlight Data Visualization

Infragistics®

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

the RequiredArgument attribute, meaning that the WF validation services will enforce that it's set with an expression.

I rely on the default implementation of CacheMetadata that reflects on the public surface of the activity to find and register runtime metadata.

The most important part is in the Execute method. I first look for the bookmark created by the parent Series activity. Because the bookmark was stored as an execution property, I look for it in context.Properties. Once I find that bookmark, I resume it, passing the TargetActivityName as input data. This bookmark resumption will result in the Series.Goto method being invoked (because it's the bookmark callback supplied when the bookmark was created). That method will look for the next activity in the collection, schedule it and cancel the activity that's currently executing:

```
// Get the bookmark created by the parent Series
Bookmark bookmark = context.Properties.Find(Series.GotoPropertyName) as
Bookmark;

// Resume the bookmark passing the target activity name
context.ResumeBookmark(bookmark, this.TargetActivityName.Get(context));
```

The final line of code is the trickiest one: creating a bookmark for synchronization that will keep the GoTo activity running. Hence, when the GoTo.Execute method is complete, this activity will still be in executing state, waiting for a stimulus to resume the bookmark. When I discussed the code for Series.Goto, I mentioned that it cancelled the activity being executed. In this case, Series.Goto is actually canceling a Goto activity instance that's waiting for this bookmark to be resumed.

Figure 6 **Using GoTo in a Series**

```
var counter = new Variable<int>();

var act = new Series
{
  Variables = { counter},
  Activities =
  {
    new WriteLine
    {
      DisplayName = "Start",
      Text = "Step 1"
    },
    new WriteLine
    {
      DisplayName = "First Step",
      Text = "Step 2"
    },
    new Assign<int>
    {
      To = counter,
      Value = new InArgument<int>(c => counter.Get(c) + 1)
    },
    new If
    {
      Condition = new InArgument<bool>(c => counter.Get(c) == 3),
      Then = new WriteLine
      {
        Text = "Step 3"
      },
      Else = new GoTo { TargetActivityName = "First Step" }
    },
    new WriteLine
    {
      Text = "The end!"
    }
  }
};

WorkflowInvoker.Invoke(act);
```

References

Windows Workflow Foundation 4 Developer Center
msdn.microsoft.com/netframework/aa663328

Endpoint.tv: Activities Authoring Best Practices
channel9.msdn.com/shows/Endpoint/endpointtv-Workflow-and-Custom-Activities-Best-Practices-Part-1/

Designing and Implementing Custom Activities
msdn.microsoft.com/library/dd489425

ActivityInstance Class
msdn.microsoft.com/library/system.activities.activityinstance

RuntimeArgument Class
msdn.microsoft.com/library/dd454495

To explain in more detail: The instance of GoTo activity was scheduled by the Series activity. When this activity completes, the completion callback in Series (OnChildCompleted) looks for the next activity in Series.Activities collection and schedules it. In this case I don't want to schedule the next activity—I want to schedule the activity referenced by TargetActivityName. This bookmark enables this because it keeps the GoTo activity in an executing state while the target activity is being scheduled. When GoTo is cancelled, there's no action in the Series.OnChildCompleted callback because it only schedules the next activity if the completion state is Closed (and, in this case, is Cancelled):

```
// Create a bookmark to leave this activity idle waiting when it does
// not have any further work to do. Series will cancel this activity
// in its GoTo method
context.CreateBookmark("SyncBookmark");
```

Figure 6 shows an example using this activity. In this case, I'm looping back to a previous state according to the value of a variable. This is a simple example to illustrate the basic use of Series, but this activity can be used to implement complex, real-world business scenarios where you need to skip, redo or jump to steps in a sequential process.

Go with the Flow

In this article I presented the general aspects of writing custom control flow activities. In WF 4, the control flow spectrum is not fixed; writing custom activities has been dramatically simplified. If the activities provided out-of-the-box don't meet your needs, you can easily create your own. In this article I started with a simple control flow activity and then worked my way up to implement a custom control flow activity that adds new execution semantics to WF 4. If you want to learn more, a Community Technology Preview for State Machine is available at CodePlex with full source code. You'll also find a series of Channel 9 videos on activity authoring best practices. By writing your own custom activities, you can express any control flow pattern in WF and accommodate WF to the particulars of your problem. ■

LEON WELICKI *is a program manager in the Windows Workflow Foundation (WF) team at Microsoft working on the WF runtime. Prior to joining Microsoft, he worked as lead architect and dev manager for a large Spanish telecom company and as an external associate professor on the graduate computer science faculty at the Pontifical University of Salamanca at Madrid.*

THANKS to the following technical experts for reviewing this article:
Joe Clancy, Dan Glick, Rajesh Sampath, Bob Schmidt and Isaac Yuen

DEVELOP

Rich Business Intelligence Applications in WPF and Silverlight

Robust Pivot Grids for WPF and Silverlight let your users analyze data to make key business decisions. Visit infragistics.com to try it today!

Net**Advantage**®
for Silverlight Data Visualization

Net**Advantage**®
for WPF Data Visualization

Infragistics®

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

Using MEF to Expose Interfaces in Your Silverlight MVVM Apps

Sandrino Di Mattia

While many developers may think of Silverlight as a Web-centric technology, in practice it has become a great platform for building any type of application. Silverlight has built-in support for concepts such as data binding, value converters, navigation, out-of-browser operation and COM Interop, making it relatively straightforward to create all kinds of applications. And when I say all kinds, I also mean enterprise applications.

Creating a Silverlight application with the Model-View-ViewModel (MVVM) pattern gives you, in addition to the features already in Silverlight, the advantages of greater maintainability, testability and separation of your UI from the logic behind it. And, of course, you don't have to figure all this out on your own. There's a wealth of information and tools out there to help you get started. For example, the MVVM Light Toolkit (mvvmlight.codeplex.com) is a lightweight framework for implementing MVVM with Silverlight and Windows Presentation Foundation (WPF), and WCF RIA Services (silverlight.net/getstarted/riaservices) helps you easily access Windows Communication Foundation (WCF) services and databases thanks to code generation.

You can take your Silverlight application a step further with the Managed Extensibility Framework (mef.codeplex.com), also known as MEF. This framework provides the plumbing to create extensible applications using components and composition.

In the rest of the article I'll show you how to use MEF to get centralized management of the View and ViewModel creation. Once you have this, you can go much further than just putting a ViewModel in the DataContext of the View. All this will be done by customizing the built-in Silverlight navigation. When the user navigates to a given URL, MEF intercepts this request, looks at the route (a bit like ASP.NET MVC), finds a matching View and ViewModel, notifies the ViewModel of what's happening and displays the View.

This article discusses:

- Getting started with MEF
- Extending Silverlight and MEF
- Custom navigation
- Handling the View and ViewModel

Technologies discussed:

ASP.NET, Silverlight

Code download available at:

code.msdn.microsoft.com/mag201101MEF

Getting Started with MEF

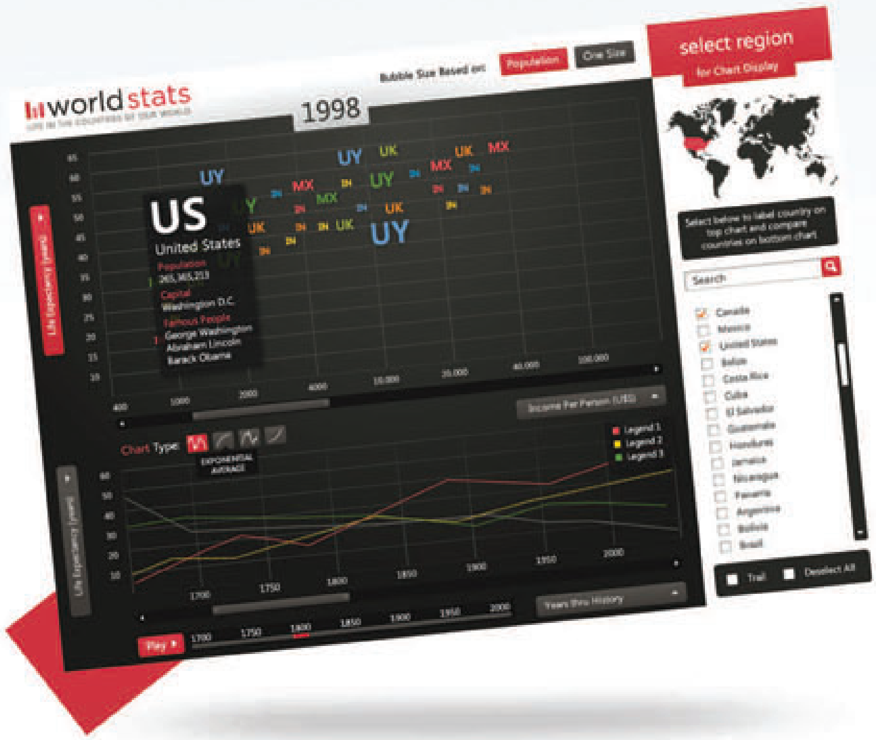
Because MEF is the engine that will connect all the parts of this example, it's best to start with it. If you're not familiar with MEF already, start by reading Glenn Block's article, "Building Composable Apps in .NET 4 with the Managed Extensibility Framework," in the February 2010 issue of *MSDN Magazine* (msdn.microsoft.com/magazine/ee291628).

First you need to correctly configure MEF when the application starts by handling the Startup event of the App class:

```
private void OnStart(object sender, StartupEventArgs e) {
  // Initialize the container using a deployment catalog.
  var catalog = new DeploymentCatalog();
  var container = CompositionHost.Initialize(catalog);
  // Export the container as singleton.
  container.ComposeExportedValue<CompositionContainer>(container);
  // Make sure the MainView is imported.
  CompositionInitializer.SatisfyImports(this);
}
```

EXPERIENCE

Beautiful Data Visualizations That Bring Your Data to Life



Use our Motion Framework™ to see your data over time and give your users new insight into their data. Visit infragistics.com to try it today!

NetAdvantage® ULTIMATE

for ASP.NET, Windows Forms, WPF, Silverlight, WPF Data Visualization, Silverlight Data Visualization

Infragistics®

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

Figure 1 **CompositionNavigationBehavior**

```
public class CompositionNavigationBehavior : Behavior<Frame> {
  private bool processed;
  [Import]
  public CompositionContainer Container {
    get; set;
  }

  [Import]
  public CompositionNavigationContentLoader Loader {
    get; set;
  }

  public CompositionNavigationBehavior() {
    if (!DesignerProperties.IsInDesignTool)
      CompositionInitializer.SatisfyImports(this);
  }

  protected override void OnAttached() {
    base.OnAttached();
    if (!processed) {
       this.RegisterNavigationService();
       this.SetContentLoader();
       processed = true;
    }
  }

  private void RegisterNavigationService() {
    var frame = AssociatedObject;
    var svc = new NavigationService(frame);
    Container.ComposeExportedValue<INavigationService>(svc);
  }

  private void SetContentLoader() {
    var frame = AssociatedObject;
    frame.ContentLoader = Loader;
    frame.JournalOwnership = JournalOwnership.Automatic;
  }
}
```

The deployment catalog makes sure all assemblies are scanned for exports and is then used to create a CompositionContainer. Because the navigation will require this container to do some work later on, it's important to register the instance of this container as an exported value. This will allow the same container to be imported whenever required.

Another option would be to store the container as a static object, but this would create tight coupling between the classes, which isn't suggested.

Extending Silverlight Navigation

Silverlight Navigation Application is a Visual Studio template that enables you to quickly create applications that support navigation using a Frame that hosts the content. The great thing about Frames is that they integrate with the Back and Forward buttons of your browser and they support deep linking. Look at the following:

```
<navigation:Frame x:Name="ContentFrame"
  Style="{StaticResource ContentFrameStyle}"
  Source="Customers"
  NavigationFailed="OnNavigationFailed">
  <i:Interaction.Behaviors>
    <fw:CompositionNavigationBehavior />
  </i:Interaction.Behaviors>
</navigation:Frame>
```

This is just a regular frame that starts by navigating to Customers. As you can see, this Frame doesn't contain a UriMapper (where you could link Customers to a XAML file, such as /Views/Customers.aspx). The only thing it contains is my custom behavior, CompositionNavigation-Behavior. A behavior (from the System.Windows.Interactivity assembly) allows you to extend existing controls, such as a Frame in this case.

Figure 1 shows the behavior. Let's take a look at what this CompositionNavigationBehavior does. The first thing you can see is that the behavior wants a CompositionContainer and a CompositionNavigationLoader (more on this later) because of the Import attributes. The constructor then forces the Import using the SatisfyImports method on the CompositionInitializer. Note that you should only use this method when you don't have another choice, as it actually couples your code to MEF.

MEF is the engine that will connect all the parts of this example.

When the Frame is attached, a NavigationService is created and wrapped around the Frame. Using ComposeExportedValue, the instance of this wrapper is registered in the container.

When the container was created, the instance of this container was also registered in itself. As a result, an Import of Composition-Container will always give you the same object; this is why I used ComposeExportedValue in the Startup event of the App class. Now the CompositionNavigationBehavior asks for a CompositionContainer using the Import attribute and will get it after SatisfyImports runs.

When registering the instance of INavigationService the same thing happens. It's now possible from anywhere in the application to ask for an INavigationService (that wraps around a Frame). Without having to couple your ViewModels to a frame you get access to the following:

```
public interface INavigationService {
  void Navigate(string path);
  void Navigate(string path, params object[] args);
}
```

Figure 2 **Creating the ViewModelExportAttribute**

```
[MetadataAttribute]
[AttributeUsage(AttributeTargets.Class, AllowMultiple = false)]
public class ViewModelExportAttribute :
  ExportAttribute, IViewModelMetadata {
..public Type ViewModelContract { get; set; }
  public string NavigationPath { get; set; }
  public string Key { get; set; }

  public ViewModelExportAttribute(Type viewModelContract,
    string navigationPath) : base(typeof(IViewModel)) {

    this.NavigationPath = navigationPath;
    this.ViewModelContract = viewModelContract;
    if (NavigationPath != null &&
      NavigationPath.Contains("/")) {
      // Split the path to get the arguments.
      var split = NavigationPath.Split(new char[] { '/' },
        StringSplitOptions.RemoveEmptyEntries);
      // Get the key.
      Key = split[0];
    }
    else {
      // No arguments, use the whole key.
      Key = NavigationPath;
    }
  }
}
```

Figure 3 **Custom INavigationContentLoader**

```
[Export] public class CompositionNavigationContentLoader :
  INavigationContentLoader {
  [ImportMany(typeof(IView))]
  public IEnumerable<ExportFactory<IView, IViewMetadata>>
    ViewExports { get; set; }

  [ImportMany(typeof(IViewModel))]
  public IEnumerable<ExportFactory<IViewModel, IViewModelMetadata>>
    ViewModelExports { get; set; }

  public bool CanLoad(Uri targetUri, Uri currentUri) {
    return true;
  }

  public void CancelLoad(IAsyncResult asyncResult) {
    return;
  }

  public IAsyncResult BeginLoad(Uri targetUri, Uri currentUri,
    AsyncCallback userCallback, object asyncState) {
    // Convert to a dummy relative Uri so we can access the host.
    var relativeUri = new Uri("http://" + targetUri.OriginalString,
      UriKind.Absolute);

    // Get the factory for the ViewModel.
    var viewModelMapping = ViewModelExports.FirstOrDefault(o =>
      o.Metadata.Key.Equals(relativeUri.Host,
      StringComparison.OrdinalIgnoreCase));

    if (viewModelMapping == null)
      throw new InvalidOperationException(
        String.Format("Unable to navigate to: {0}. " +
          "Could not locate the ViewModel.",
          targetUri.OriginalString));

    // Get the factory for the View.
    var viewMapping = ViewExports.FirstOrDefault(o =>
      o.Metadata.ViewModelContract ==
      viewModelMapping.Metadata.ViewModelContract);

    if (viewMapping == null)
      throw new InvalidOperationException(
        String.Format("Unable to navigate to: {0}. " +
          "Could not locate the View.",
          targetUri.OriginalString));

    // Resolve both the View and the ViewModel.
    var viewFactory = viewMapping.CreateExport();
    var view = viewFactory.Value as Control;
    var viewModelFactory = viewModelMapping.CreateExport();
    var viewModel = viewModelFactory.Value as IViewModel;

    // Attach ViewModel to View.
    view.DataContext = viewModel;
    viewModel.OnLoaded();

    // Get navigation values.
    var values = viewModelMapping.Metadata.GetArgumentValues(targetUri);
    viewModel.OnNavigated(values);

    if (view is Page) {
      Page page = view as Page;
      page.Title = viewModel.GetTitle();
    }
    else if (view is ChildWindow) {
      ChildWindow window = view as ChildWindow;
      window.Title = viewModel.GetTitle();
    }

    // Do not navigate if it's a ChildWindow.
    if (view is ChildWindow) {
      ProcessChildWindow(view as ChildWindow, viewModel);
      return null;
    }
    else {
      // Navigate because it's a Control.
      var result = new CompositionNavigationAsyncResult(asyncState, view);
      userCallback(result);
      return result;
    }
  }

  private void ProcessChildWindow(ChildWindow window,
    IViewModel viewModel) {
    // Close the ChildWindow if the ViewModel requests it.
    var closableViewModel = viewModel as IClosableViewModel;

    if (closableViewModel != null)  {
      closableViewModel.CloseView += (s, e) => { window.Close(); };
    }

    // Show the window.
    window.Show();
  }

  public LoadResult EndLoad(IAsyncResult asyncResult) {
    return new LoadResult((asyncResult as
      CompositionNavigationAsyncResult).Result);
  }
}
```

Now, let's assume you have a ViewModel showing all of your customers and this ViewModel should be able to open a specific customer. This could be achieved using the following code:

```
[Import]
public INavigationService NavigationService {
  get; set;
}

private void OnOpenCustomer() {
  NavigationService.Navigate(
    "Customer/{0}", SelectedCustomer.Id);
}
```

Your attribute should implement an interface with the values you'll want to expose as metadata.

But before jumping ahead, let's discuss the SetContentLoader method in the CompositionNavigationBehavior. It changes the ContentLoader of the Frame. This is a perfect example of the support for extensibility in Silverlight. You can provide your own ContentLoader (that implements the INavigationContentLoader interface) to really provide something to show in the Frame.

Now that you can see how things start falling into place, the following topic—extending MEF—will become clear.

Back to Extending MEF

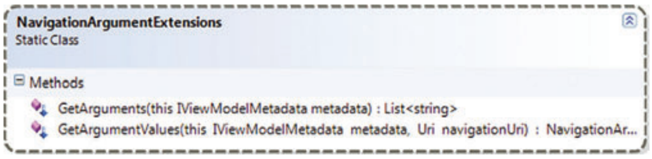
The goal here is that you can navigate to a certain path (be it from the ViewModel or your browser address bar) and the Composition-NavigationLoader does the rest. It should parse the URI, find a matching ViewModel and a matching View, and combine them.

Normally you'd write something like this:

```
[Export(typeof(IMainViewModel))]
public class MainViewModel
```

In this case it would be interesting to use the Export attribute with some extra configuration, referred to as metadata. **Figure 2** shows an example of a metadata attribute.

This attribute doesn't do anything special. In addition to the ViewModel interface, it allows you to define a navigation path such


NavigationArgumentExtensions
Static Class

Methods
GetArguments(this IViewModelMetadata metadata) : List<string>
GetArgumentValues(this IViewModelMetadata metadata, Uri navigationUri) : NavigationAr...

Figure 4 **Extension Methods for Navigation Arguments**

as Customer/{Id}. Then it will process this path using Customer as Key and {Id} as one of the arguments. Here's an example of how this attribute is used:

```
[ViewModelExport(typeof(ICustomerDetailViewModel),
  "Customer/{id}")]
public class CustomerDetailViewModel
  : ICustomerDetailViewModel
```

Before continuing, there are a few important things to note. First, your attribute should be decorated with the [MetadataAttribute] to work correctly. Second, your attribute should implement an interface with the values you'll want to expose as metadata. And finally, mind the constructor of the attribute—it passes a type to the base constructor. The class that's decorated with this attribute will be exposed using this type. In the case of my example, this would be IViewModel.

That's it for exporting the ViewModels. If you want to import them somewhere, you should be writing something like this:

```
[ImportMany(typeof(IViewModel))]
public List<Lazy<IViewModel, IViewModelMetadata>> ViewModels {
  get;
  set;
}
```

This will give you a list that contains all exported ViewModels with their respective metadata, allowing you to enumerate the list and maybe pick out only the ones of interest to you (based on the metadata). In fact, the Lazy object will make sure that only the ones of interest are actually instantiated.

The View will need something similar:

```
[MetadataAttribute]
[AttributeUsage(AttributeTargets.Class, AllowMultiple = false)]
public class ViewExportAttribute :
  ExportAttribute, IViewMetadata {

  public Type ViewModelContract { get; set; }
  public ViewExportAttribute() : base(typeof(IView)) {
  }
}
```

There's nothing special in this example, either. This attribute allows you to set the contract of the ViewModel to which the View should be linked.

Here's an example of AboutView:

```
[ViewExport(ViewModelContract = typeof(IAboutViewModel))]
public partial class AboutView : Page, IView {
  public AboutView() {
    InitializeComponent();
  }
}
```

A Custom INavigationContentLoader

Now that the overall architecture has been set up, let's take a look at controlling what's loaded when a user navigates. To create a custom content loader, the following interface needs to be implemented:

```
public interface INavigationContentLoader {
  IAsyncResult BeginLoad(Uri targetUri, Uri currentUri,
    AsyncCallback userCallback, object asyncState);
  void CancelLoad(IAsyncResult asyncResult);
  bool CanLoad(Uri targetUri, Uri currentUri);
  LoadResult EndLoad(IAsyncResult asyncResult);
}
```

The most important part of the interface is the BeginLoad method, because this method should return an AsyncResult containing the item that will be displayed in the Frame. **Figure 3** shows the implementation of the custom INavigationContentLoader.

As you can see, a lot happens in this class—but it's actually simple. The first thing to notice is the Export attribute. This is required to be able to import this class in the CompositionNavigationBehavior.

The most important parts of this class are the ViewExports and ViewModelExports properties. These enumerations contain all exports for the Views and the ViewModels, including their metadata. Instead of using a Lazy object I'm using an ExportFactory. This is a huge difference! Both classes will only instantiate the object when required, but the difference is that with the Lazy class you can only create a single instance of the object. The ExportFactory (named after the Factory pattern) is a class that allows you to request a new instance of the type of object whenever you feel like it.

Finally, there's the BeginLoad method. This is where the magic happens. This is the method that will provide the Frame with the content to display after navigating to a given URI.

Instead of using a Lazy object I'm using an ExportFactory.

Creating and Processing Objects

Let's say you tell the frame to navigate to Customers. This will be what you'll find in the targetUri argument of the BeginLoad method. Once you have this you can get to work.

The first thing to do is find the correct ViewModel. The ViewModelExports property is an enumeration that contains all the exports with their metadata. Using a lambda expression you can find the correct ViewModel based on its key. Remember the following:

```
[ViewModelExport(typeof(ICustomersViewModel), "Customers")]
public class CustomersViewModel :
  ContosoViewModelBase, ICustomersViewModel
```

Well, imagine you navigate to Customers. Then the following code will find the right ViewModel:

```
var viewModelMapping = ViewModelExports.FirstOrDefault(o => o.Metadata.
Key.Equals("Customers",
  StringComparison.OrdinalIgnoreCase));
```

Once the ExportFactory is located, the same thing should happen for the View. However, instead of looking for the navigation key, you look for the ViewModelContract as defined in both the ViewModelExportAttribute and the ViewModelAttribute:

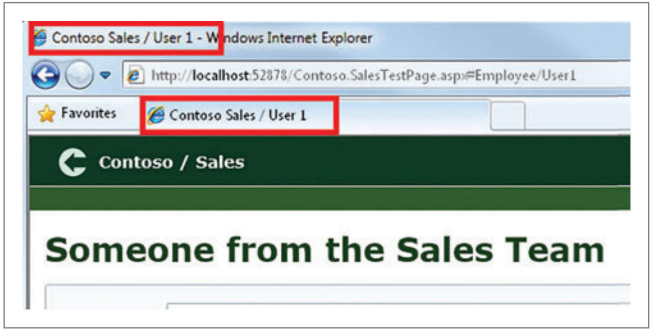

Contoso Sales / User 1 - Windows Internet Explorer
http://localhost:52878/Contoso.SalesTestPage.aspx#Employee/User1
Favorites
Contoso Sales / User 1
Contoso / Sales
Someone from the Sales Team

Figure 5 **Setting a Custom Window Title**

```
[ViewExport(ViewModelContract = typeof(IAboutViewModel))
public partial class AboutView : Page
```

Once both ExportFactories are found, the hard part is over. Now the CreateExport method allows you to create a new instance of the View and the ViewModel:

```
var viewFactory = viewMapping.CreateExport();
var view = viewFactory.Value as Control;
var viewModelFactory = viewModelMapping.CreateExport();
var viewModel = viewModelFactory.Value as IViewModel;
```

After both the View and the ViewModel have been created, the ViewModel is stored in the DataContext of the View, starting the required data bindings. And the OnLoaded method of the ViewModel is called to notify the ViewModel that all the heavy lifting has been done, and also that all Imports—if there are any—have been imported.

You shouldn't underestimate this last step when you're using the Import and ImportMany attributes. In many cases you'll want to do something when creating a ViewModel, but only when everything has been loaded correctly. If you're using an ImportingConstructor you definitely know when all Imports were imported (that would be when the constructor is called). But when working with the Import/ImportMany attributes, you should start writing code in all your properties to set flags in order to know when all properties have been imported.

In this case the OnLoaded method solves this issue for you.

Now you've seen the whole process of how the View and ViewModel are constructed.

Passing Arguments to the ViewModel

Take a look at the IViewModel interface, and pay attention to the OnNavigated method:

```
public interface IViewModel {
  void OnLoaded();
  void OnNavigated(NavigationArguments args);
  string GetTitle();
}
```

When you navigate to Customers/1, for example, this path is parsed and the arguments are combined in the NavigationArguments class (this is just a Dictionary with extra methods like GetInt, GetString and so on). Because it's mandatory that each ViewModel implements the IViewModel interface, it's possible to call the OnNavigated method after resolving the ViewModel:

```
// Get navigation values.
var values = viewModelMapping.Metadata.GetArgumentValues(targetUri);
viewModel.OnNavigated(values);
```

When the CustomersViewModel wants to open a CustomerDetailViewModel, the following happens:

```
NavigationService.Navigate("Customer/{0}", SelectedCustomer.Id);
```

These arguments then arrive in the CustomerDetailViewModel and can be used to pass to the DataService, for example:

```
public override void OnNavigated(NavigationArguments args) {
  var id = args.GetInt("Id");
  if (id.HasValue) {
    Customer = DataService.GetCustomerById(id.Value);
  }
}
```

To find the arguments, I wrote a class containing two extension methods that do some work based on the information in the ViewModel metadata (see **Figure 4**). This proves again that the metadata concept in MEF is really useful.

The Final Chores

If the View is a Page or a ChildWindow, the title of this control will also be extracted from the IViewModel object. This allows you to dynamically set the titles of your Pages and ChildWindows based on the current customer, as shown in **Figure 5**.

After all these great little things, there's one last step. If the View is a ChildWindow the window should be displayed. But if the ViewModel implements IClosableViewModel, the CloseView event of this ViewModel should be linked to the Close method on the ChildWindow.

The IClosableViewModel interface is simple:

```
public interface IClosableViewModel : IViewModel {
  event EventHandler CloseView;
}
```

Processing the ChildWindow is also trivial. When the ViewModel raises the CloseView event, the Close method of the ChildWindow gets called. This allows you to indirectly connect the ViewModel to the View:

```
// Close the ChildWindow if the ViewModel requests it.
var closableViewModel = viewModel as IClosableViewModel;
if (closableViewModel != null) {
  closableViewModel.CloseView += (s, e) => {
    window.Close();
  };
}

// Show the window.
window.Show();
```

If the View isn't a ChildWindow, then it should simply be made available in the IAsyncResult. This will show the View in the Frame.

There. Now you've seen the whole process of how the View and ViewModel are constructed.

Using the Example Code

The code download for this article contains an MVVM application using this type of custom navigation with MEF. The solution contains the following examples:

- Navigating to a regular UserControl
- Navigating to a regular UserControl by passing arguments (.../#Employee/DiMattia)
- Navigating to a ChildWindow by passing arguments (.../#Customer/1)
- Imports of the INavigationService, the IDataService, ...
- Examples of ViewExport and ViewModelExport configuration

This article should have provided a good idea of how the sample works. For a deeper understanding, play with the code and customize it for your own applications. You'll see how powerful and flexible MEF can be. ■

SANDRINO DI MATTIA *is a software engineer at RealDolmen and has a passion for everything that is Microsoft. He also participates in user groups and writes articles on his blog at blog.sandrinodimattia.net.*

THANKS to the following technical experts for reviewing this article: *Glenn Block and Daniel Plaisted*

Date	Country	Product
Wed Sep 1 2009 2:09:35 AM	US	🟦 Product A
Wed Sep 1 2009 6:03:14 AM	US	🟥 Product B
Wed Sep 1 2009 9:31:23 AM	Canada	🟨 Product C
Wed Sep 1 2009 10:56:54 AM	US	🟨 Product C
Wed Sep 1 2009 11:51:18 AM	US	🟨 Product C
Wed Sep 1 2009 1:20:20 PM	US	🟥 Product B
Wed Sep 1 2009 2:29:44 PM	France	🟦 Product A

Visitors	Signups	Downloa
68	16	
67	11	
48	13	
47	14	
23	12	
21	9	

Microsoft Silverlight Microsoft Visual Studio

Your Data in a Whole New Light

ComponentArt

Data Visualization for .NET

ComponentArt's latest Data Visualization technology allows you to present, navigate and visualize your data like never before. Visit our website and experience a whole new level of interactivity, flexibility and performance.

Charting Gauges Maps GridView TimeNavigator CalcEngine

www.componentart.com

"How do I get my data presented like this?" That is typically the first question we get from business users who have seen the interactive power of our technology. Rest assured, ComponentArt dashboards work with any data source. Contact us today to learn how you can start presenting your data in a whole new light.

ComponentArt
Build Something Amazing

Data Processing: Parallelism and Performance

Johnson M. Hart

Processing data collections is a fundamental computing task, and a number of practical problems are inherently parallel, potentially enabling improved performance and throughput on multicore systems. I'll compare several distinct Windows-based methods to solve problems with a high degree of data parallelism.

The benchmark I'll use for this comparison is a search problem ("Geonames") from Chapter 9 of Troy Magennis' book, "LINQ to Objects Using C# 4.0" (Addison-Wesley, 2010). The alternative solutions are:

- Parallel Language Integrated Query (PLINQ) and C# 4.0, with and without enhancements to the original code.
- Windows native code using C, the Windows API, threads and memory-mapped files.
- Windows C#/Microsoft .NET Framework multithreaded code.

The source code for all solutions is available on my Web site (jmhartsoftware.com). Other parallelism techniques, such as the Windows Task Parallel Library (TPL), are not examined directly, although PLINQ is layered on the TPL.

This article discusses:

- Evaluation criteria for alternative parallelism strategies
- Optimizing a benchmark data search problem
- A performance comparison of different approaches

Technologies discussed:

PLINQ, C#, Microsoft .NET Framework

Code download available at:

jmhartsoftware.com

Comparing and Evaluating Alternative Solutions

The solution evaluation criteria, in order of importance, are:

- Total performance in terms of elapsed time to complete the task.
- Scalability with the degree of parallelism (number of tasks), core count and data collection size.
- Code simplicity, elegance, ease of maintenance and similar intangible factors.

Results Summary

This article will show that for a representative benchmark search problem:

- You can successfully exploit multicore 64-bit systems to improve performance in many data-processing problems, and PLINQ can be part of the solution.
- Competitive, scalable PLINQ performance requires indexed data collection objects; supporting the IEnumerable interface isn't sufficient.
- The C#/.NET and native code solutions are fastest.
- The original PLINQ solution is slower by a factor of nearly 10, and it doesn't scale beyond two tasks, whereas the other solutions scale well up to six tasks with six cores (the maximum tested). However, code enhancements improve the original solution significantly.
- The PLINQ code is the simplest and most elegant in all respects because LINQ provides a declarative query capability for memory-resident and external data. The native code is ungainly, and the C#/.NET code is considerably better than—but not as simple as—the PLINQ code.
- All methods scale well with file size up to the limits of the test system physical memory.

The Benchmark Problem: Geonames

The idea for this article came from Chapter 9 of Magennis' LINQ book, which demonstrates PLINQ by searching a geographic database containing more than 7.25 million place names in an 825MB file (that's more than one location for every 1,000 people). Each place name is represented by a UTF-8 (en.wikipedia.org/wiki/UTF-8) text line record (variable length) with more than 15 tab-separated data columns. *Note: The UTF-8 encoding assures that a tab (0x9) or line feed (0xA) value won't occur as part of a multi-byte sequence; this is essential for several implementations.*

Magennis' Geonames program implements a hard-coded query to identify all locations with an elevation (column 15) greater than 8,000 meters, displaying the location name, country and elevation, sorted by decreasing elevation. In case you're wondering, there are 16 such locations, and Mt. Everest is the highest at 8,848 meters.

Magennis reports elapsed times of 22.3 seconds (one core) and 14.1 seconds (two cores). Previous experience (for example, see my article "Windows Parallelism, Fast File Searching and Speculative Processing" at informit.com/articles/article.aspx?p=1606242) shows that files of this size can be processed in a few seconds and performance scales well with the core count. Therefore, I decided to attempt to replicate that experience and also try to enhance Magennis' PLINQ code for better performance. The initial PLINQ enhancements almost doubled the performance but didn't improve scalability; further enhancements, however, do yield performance nearly as good as native and C# multithreaded code.

A number of practical problems are inherently parallel.

This benchmark is interesting for several reasons:

- The subject (geographical places and attributes) is inherently interesting, and it's easy to generalize the query.
- There's a high degree of data parallelism; in principle, every record could be processed concurrently.
- The file size is modest by today's standards, but it's simple to test with larger files simply by concatenating the Geonames allCountries.txt file to itself multiple times.
- The processing is not stateless; it's necessary to determine the line and field boundaries in order to partition the file, and the lines need to be processed to identify the individual fields.

An Assumption: Assume that the number of identified records (in this instance, locations higher than 8,000 meters) is small, so that sorting and display time are minimal relative to the overall processing time, which involves examining every byte.

Another Assumption: The performance results represent time required to process memory-resident data collections, such as data produced by a prior program step. The benchmark program does read a file, but the test programs are run several times to assure that the file is memory-resident. However, I'll mention the time required to load the file initially, and this time is approximately the same for all solutions.

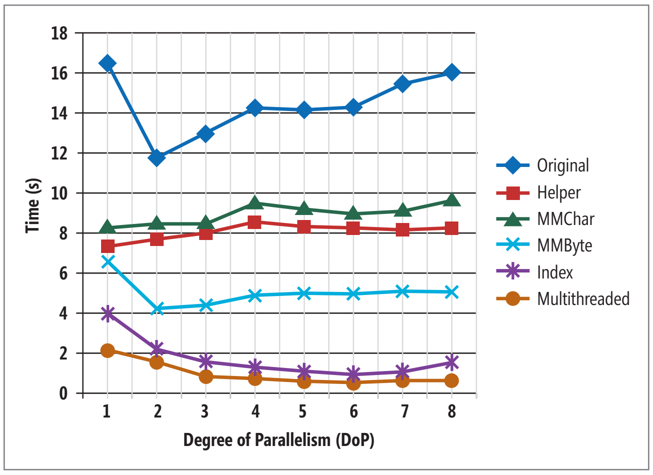


Figure 1 **Geonames Performance as a Function of Degree of Parallelism**

Performance Comparison

The first test system is a six-core desktop system running Windows 7 (AMD Phenom II, 2.80 GHz, 4GB RAM). Later, I'll present results for three other systems with hyper-threading, or HT (en.wikipedia.org/wiki/Hyper-threading), and different core counts.

Figure 1 shows the results for six different Geonames solutions, with elapsed time (seconds) as a function of "Degree of Parallelization," or DoP (the number of parallel tasks, which can be set to higher than the number of processors); the test system has six cores, but the implementations control the DoP. Six tasks are optimal; using more than six tasks degraded performance. All tests use the original Geonames 825MB allCountries.txt data file.

The implementations are (fuller explanations will follow):

1. *Geonames Original*. This is Magennis' original PLINQ solution. The performance isn't competitive and doesn't scale with processor count.
2. *Geonames Helper*. This is a performance-enhanced version of *Geonames Original*.

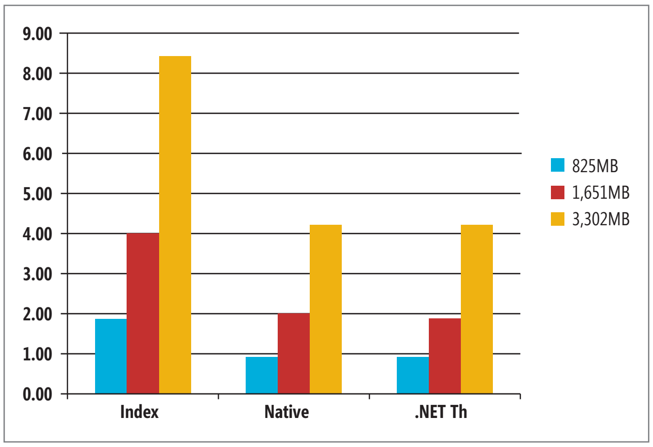


Figure 2 **Geonames Performance as a Function of File Size**

Figure 3 **Geonames Helper with Highlighted Changes to the Original PLINQ Code**

```
 1  class Program
 2  {
 3    static void Main(string[] args)
 4    {
 5      const int nameColumn = 1;
 6      const int countryColumn = 8;
 7      const int elevationColumn = 15;
 8
 9      String inFile = "Data/AllCountries.txt";
10      if (args.Length >= 1) inFile = args[0];
11
12      int degreeOfParallelism = 1;
13      if (args.Length >= 2) degreeOfParallelism = int.Parse(args[1]);
14      Console.WriteLine("Geographical data file: {0}.
          Degree of Parallelism: {1}.", inFile, degreeOfParallelism);
15
16      var lines = File.ReadLines(Path.Combine(
17        Environment.CurrentDirectory, inFile));
18
19      var q = from line in
          lines.AsParallel().WithDegreeOfParallelism(degreeOfParallelism)
20          let elevation =
              Helper.ExtractIntegerField(line, elevationColumn)
21          where elevation > 8000 // elevation in meters
22          orderby elevation descending
23          select new
24          {
25            elevation = elevation,
26            thisLine = line
27           };
28
29      foreach (var x in q)
30      {
31        if (x != null)
32        {
33          String[] fields = x.thisLine.Split(new char[] { '\t' });
34          Console.WriteLine("{0} ({1}m) - located in {2}",
35            fields[nameColumn], fields[elevationColumn],
              fields[countryColumn]);
36        }
37      }
38    }
39  }
```

3. *Geonames MMChar*. This was an unsuccessful attempt to enhance *Geonames Helper* with a memory-mapped file class similar to that used in *Geonames Threads. Note: Memory mapping allows a file to be referenced as if it was in memory without explicit I/O operations, and it can provide performance benefits.*
4. *Geonames MMByte*. This solution modifies *MMChar* to process the individual bytes of the input file, whereas the previous three solutions convert UTF-8 characters to Unicode (at 2 bytes each). The performance is the best of the first four solutions and is more than double that of *Geonames Original*.
5. *Geonames Threads* doesn't use PLINQ. This is the C#/.NET implementation using threads and a memory-mapped file. The performance is faster than *Index* (the next item) and about the same as *Native*. This solution and *Geonames Native* provide the best parallelism scalability.
6. *Geonames Index*. This PLINQ solution preprocesses the data file (requiring about nine seconds) to create a memory-resident List<byte[]> object for subsequent PLINQ processing. The preprocessing cost can be amortized over multiple queries, giving performance that's only slightly slower than *Geonames Native* and *Geonames Threads*.
7. *Geonames Native* (not shown in **Figure 1**) doesn't use PLINQ. This is the C Windows API implementation using threads and a memory-mapped file as in Chapter 10 of my book, "Windows System Programming" (Addison-Wesley, 2010). Full compiler optimization is essential for these results; the default optimization provides only about half the performance.

All implementations are 64-bit builds. 32-bit builds work in most cases, but they fail for larger files (see **Figure 2**). **Figure 2** shows performance using DoP 4 and larger files.

You can successfully exploit multicore, 64-bit systems to improve performance in many data-processing problems, and PLINQ can be part of the solution.

The test system in this case has four cores (AMD Phenom Quad-Core, 2.40 GHz, 8GB RAM). The larger files were created by concatenating multiple copies of the original file. **Figure 2** shows just the three fastest solutions, including *Geonames Index*—the fastest PLINQ solution (not counting file preprocessing)—and performance scales with the file size up to the limits of physical memory.

I'll now describe implementations two through seven and discuss the PLINQ techniques in more detail. Following that, I'll discuss results on other test systems and summarize the findings.

The Enhanced PLINQ Solutions: Geonames Helper

Figure 3 shows *Geonames Helper* with my changes (in bold face) to the *Geonames Original* code.

As many readers may not be familiar with PLINQ and C# 4.0, I'll provide a few comments about **Figure 3**, including descriptions of the enhancements:

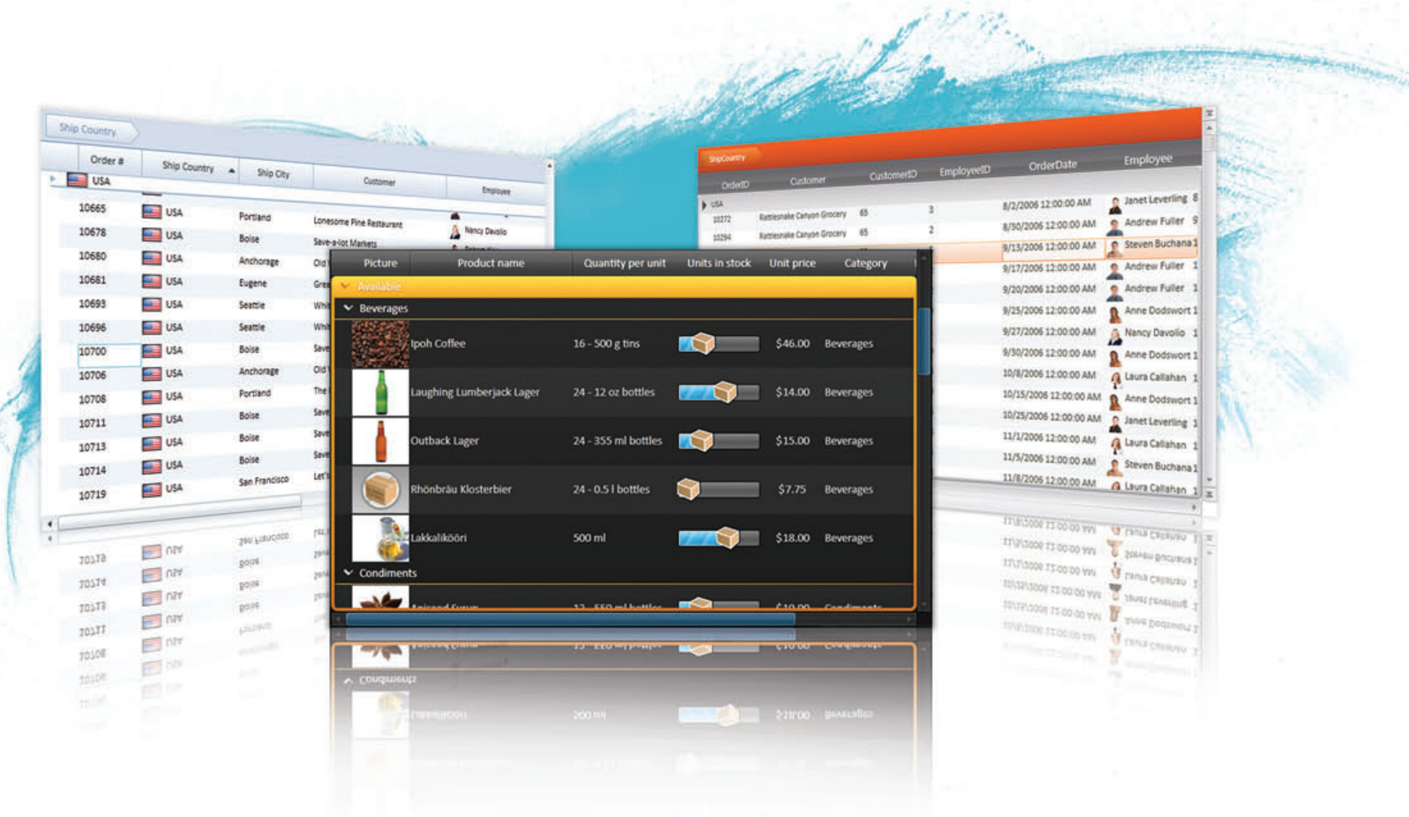
- Lines 9-14 allow the user to specify the input file name and the degree of parallelism (the maximum number of concurrent tasks) on the command line; these values are hardcoded in the original.
- Lines 16-17 start to read the file lines asynchronously and implicitly type lines as a C# String array. The lines values aren't used until Lines 19-27. Other solutions, such as *Geonames MMByte*, use a different class with its own ReadLines method, and these code lines are the only lines that need to be changed.
- Lines 19-27 are LINQ code along with the PLINQ AsParallel extension. The code is similar to SQL, and the variable "q" is implicitly typed as an array of objects consisting of an integer elevation and a String. Notice that PLINQ per-

Datagrid speed, Xceed style

XCEED
DataGrid
for Silverlight

The only Silverlight datagrid that:

- Lets end-users access remote data as fast as local.
- Uses an intelligent background data retrieval system.
- Saves end-users from experiencing lag in the UI.



Ship Country

Order #	Ship Country	Ship City	Customer	Employee
USA				
10665	USA	Portland	Lonesome Pine Restaurant	
10678	USA	Boise	Save-a-lot Markets	Nancy Davolio
10680	USA	Anchorage	Old	
10681	USA	Eugene	Gre	
10693	USA	Seattle	Whi	
10696	USA	Seattle	Whi	
10700	USA	Boise	Save	
10706	USA	Anchorage	Old	
10708	USA	Portland	The	
10711	USA	Boise	Save	
10713	USA	Boise	Save	
10714	USA	Boise	Save	
10719	USA	San Francisco	Let's	

ShipCountry

OrderID	Customer	CustomerID	EmployeeID	OrderDate	Employee
USA					
10272	Rattlesnake Canyon Grocery	65	3	8/2/2006 12:00:00 AM	Janet Leverling
10294	Rattlesnake Canyon Grocery	65	2	8/30/2006 12:00:00 AM	Andrew Fuller
				9/13/2006 12:00:00 AM	Steven Buchana
				9/17/2006 12:00:00 AM	Andrew Fuller
				9/20/2006 12:00:00 AM	Andrew Fuller
				9/25/2006 12:00:00 AM	Anne Dodswort
				9/27/2006 12:00:00 AM	Nancy Davolio
				9/30/2006 12:00:00 AM	Anne Dodswort
				10/8/2006 12:00:00 AM	Laura Callahan
				10/15/2006 12:00:00 AM	Anne Dodswort
				10/25/2006 12:00:00 AM	Janet Leverling
				11/1/2006 12:00:00 AM	Laura Callahan
				11/5/2006 12:00:00 AM	Steven Buchana
				11/8/2006 12:00:00 AM	Laura Callahan

Picture	Product name	Quantity per unit	Units in stock	Unit price	Category
Available					
Beverages					
	Ipoh Coffee	16 - 500 g tins		$46.00	Beverages
	Laughing Lumberjack Lager	24 - 12 oz bottles		$14.00	Beverages
	Outback Lager	24 - 355 ml bottles		$15.00	Beverages
	Rhönbräu Klosterbier	24 - 0.5 l bottles		$7.75	Beverages
	Lakkalikööri	500 ml		$18.00	Beverages
Condiments					

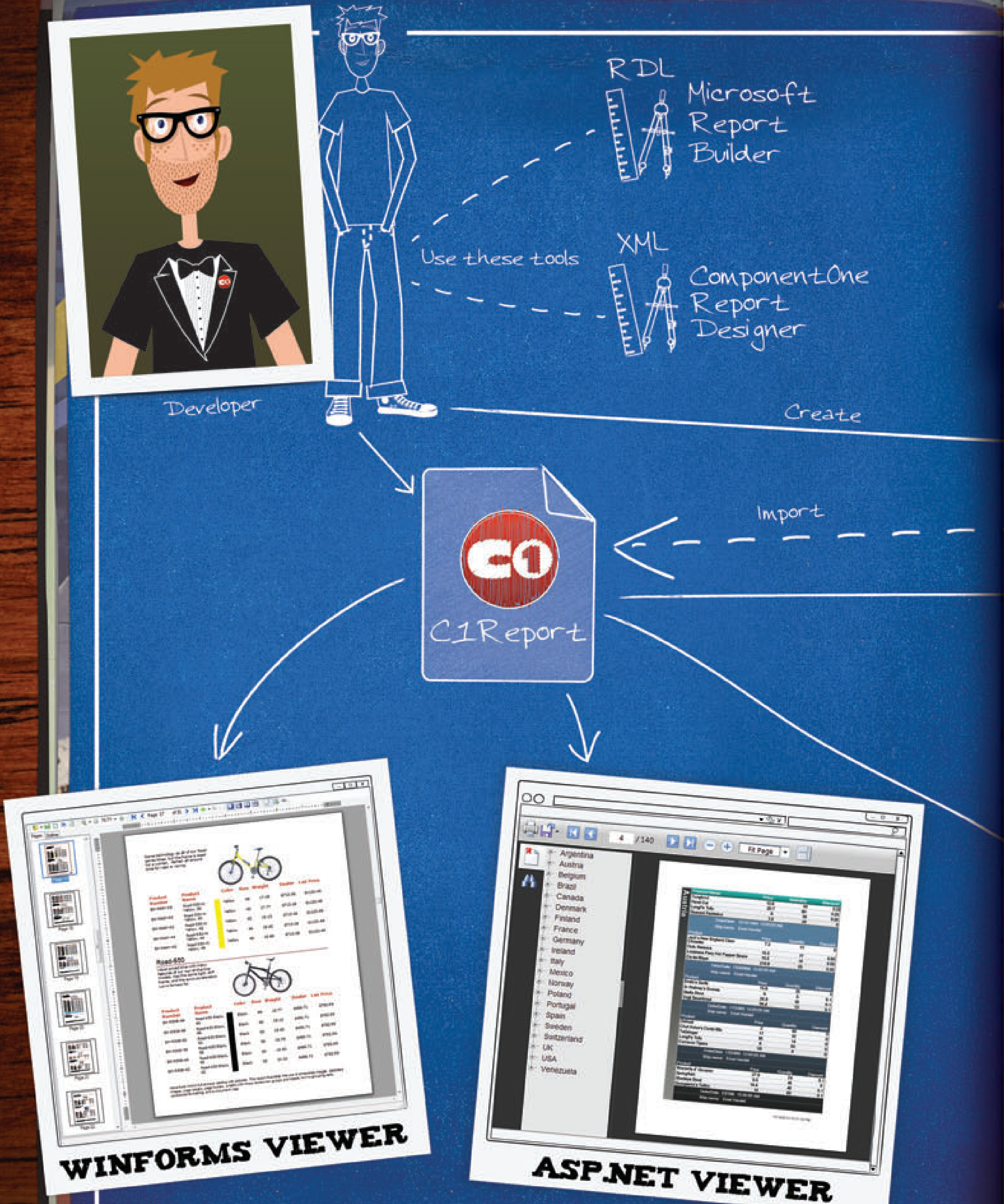
There's fast and then there's Xceed fast. Going beyond current-generation Silverlight datagrids, Xceed DataGrid for Silverlight is fast where others are slow: when working with remote data. With advanced, background record retrieval from remote sources and a responsive UI that minimizes perceived lag, the control keeps the end-user experience smooth and snappy, no matter where the data is.

Microsoft® Visual Studio® PARTNER

Datagrids, transformed

Try it live on xceed.com

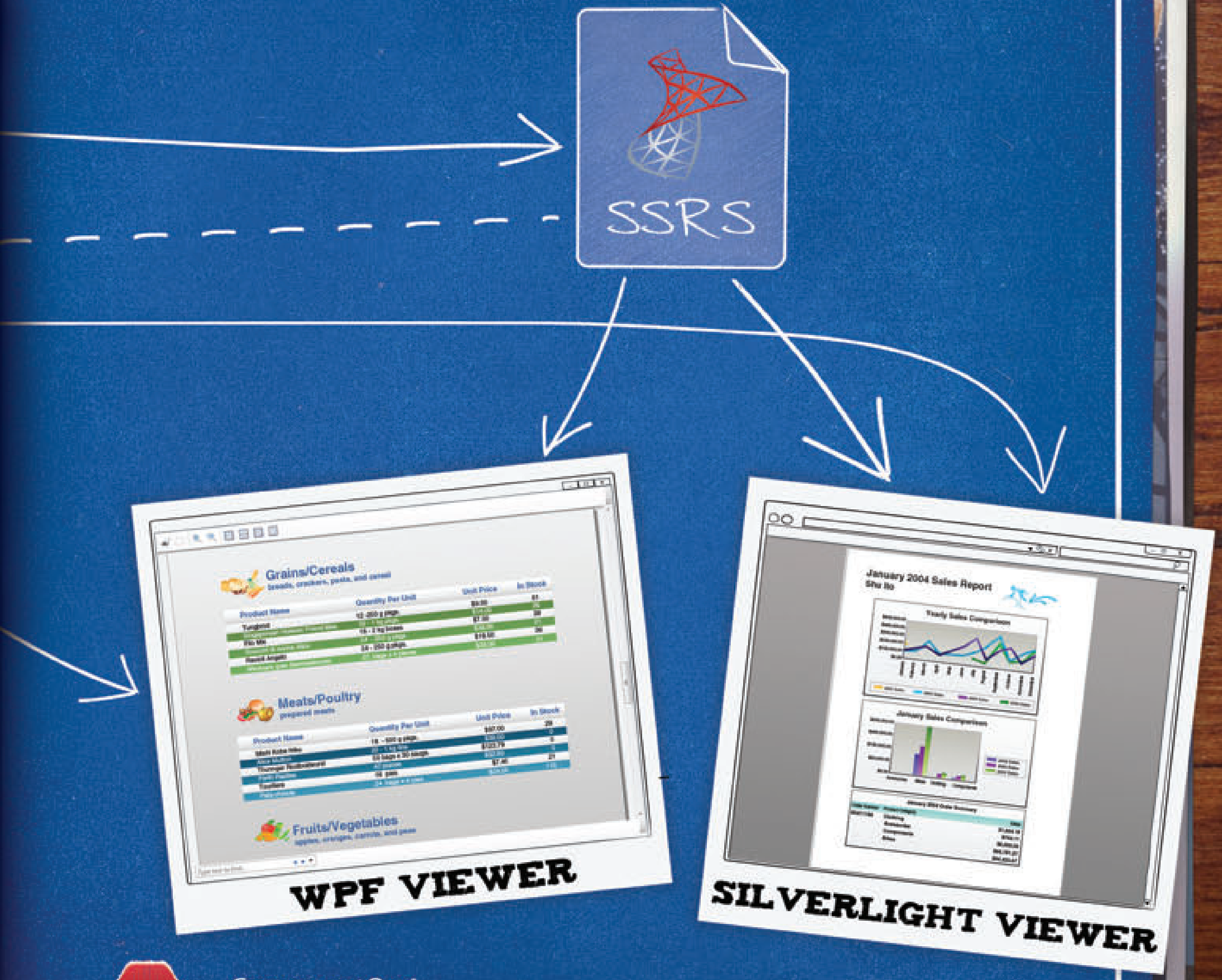
XCEED
MULTI-TALENTED COMPONENTS



ComponentOne

YOUR GUIDE TO REPORTS

Developers, just like you, continue to turn to ComponentOne Reports™ for their enterprise solutions. Our flexible reports support Microsoft SQL Server Reporting Services (SSRS) with a fully customizable object model, report designers, schedulers, viewers for all .NET platforms, and more. Start your blueprint today; download ComponentOne Studio® Enterprise for high quality, professional apps.



ComponentOne® Studio® Enterprise 2010

Download your free trial @

www.componentone.com/blueprint

forms all the thread-management work; the AsParallel method is all that's needed to turn serial LINQ code into PLINQ code.
- Line 20. **Figure 4** shows the Helper.ExtractIntegerField method. The original program uses the String.Split method in a manner similar to that used to display the results in Line 33 (**Figure 3**). This is the key to the improved performance of *Geonames Helper* compared to *Geonames Original*, as it's no longer necessary to allocate String objects for every field in every line.

Note that the AsParallel method used in Line 19 can be used with any IEnumerable object. As I mentioned earlier, **Figure 4** shows the Helper class ExtractIntegerField method. It simply extracts and evaluates the specified field (elevation in this case), avoiding library methods for performance. **Figure 1** shows that this enhancement doubles the performance with DoP 1.

Geonames MMChar and Geonames MMByte

Geonames MMChar represents an unsuccessful attempt to improve performance by memory mapping the input file, using a custom class, FileMmChar. *Geonames MMByte*, however, does yield significant benefits, as the input file bytes aren't expanded to Unicode.

MMChar requires a new class, FileMmChar, which supports the IEnumerable<String> interface. The FileMmByte class is similar and deals with byte[] objects rather than String objects. The only significant code change is to **Figure 3**, Lines 16-17, which are now:

```
var lines = FileMmByte.ReadLines(Path.Combine(
  Environment.CurrentDirectory, inFile));
```

The code for

```
public static IEnumerable<byte[]> ReadLines(String path)
```

that supports the IEnumerable<byte[]> interface in FileMmByte constructs a FileMmByte object and an IEnumerator<byte[]> object that scans the mapped file for individual lines.

Note that the FileMmChar and FileMmByte classes are "unsafe" because they create and use pointers to access the files, and they use C#/native code interoperability. All the pointer usage is, however, isolated in a separate assembly, and the code uses arrays rather than pointer dereferencing. The .NET Framework 4 MemoryMappedFile class doesn't help, because it's necessary to use accessor functions to move data from mapped memory.

Geonames Native

Geonames Native exploits the Windows API, threads and file memory mapping. The basic code patterns are described in Chapter 10 of "Windows System Programming." The program must manage the threads directly and must also carefully map the file to memory. The performance is much better than all PLINQ implementations except *Geonames Index*.

There is, however, an important distinction between the Geonames problem and a simple, stateless file search or transformation. The challenge is to determine the correct method to *partition* the input data so as to assign different partitions to different tasks. There's no obvious way to determine the line boundaries without scanning the entire file, so it isn't feasible to assign a fixed-size partition to each task. However, the solution is straightforward when illustrated with DoP 4:

- Divide the input file into four equal partitions, with the partition start location communicated to each thread as part of the thread function argument.
- Have each thread then process all lines (records) that *start* in the partition. This means that a thread will probably scan into the next partition in order to complete processing of the last line that starts in the partition.

The challenge is to determine the correct method to partition the input data so as to assign different partitions to different tasks.

Geonames Threads

Geonames Threads uses the same logic as *Geonames Native*; in fact some code is the same or nearly the same. However, lambda expressions, extension methods, containers and other C#/.NET features greatly simplify the coding.

As with *MMByte* and *MMChar*, the file memory mapping requires "unsafe" classes and C#/native code interoperability in order to use pointers to the mapped memory. The effort is worthwhile, however, because *Geonames Threads* performance is the same as *Geonames Native* performance with much simpler code.

Geonames Index

The PLINQ results (*Original*, *Helper*, *MMChar* and *MMByte*) are disappointing compared to the *Native* and *.NET Threads* results. Is

Figure 4 **Geonames Helper Class and ExtractIntegerField Method**

```
41   class Helper
42   {
43     public static int ExtractIntegerField(String line, int fieldNumber)
44     {
45       int value = 0, iField = 0;
46       byte digit;
47
48       // Skip to the specified field number and extract the decimal value.
49       foreach (char ch in line)
50       {
51         if (ch == '\t') { iField++; if (iField > fieldNumber) break; }
52         else
53         {
54           if (iField == fieldNumber)
55           {
56             digit = (byte)(ch - 0x30);  // 0x30 is the character '0'
57             if (digit >= 0 && digit <= 9)
                 { value = 10 * value + digit; }
58             else // Character not in [0-9]. Reset the value and quit.
59             { value = 0; break; }
60           }
61         }
62       }
63       return value;
64     }
65   }
```

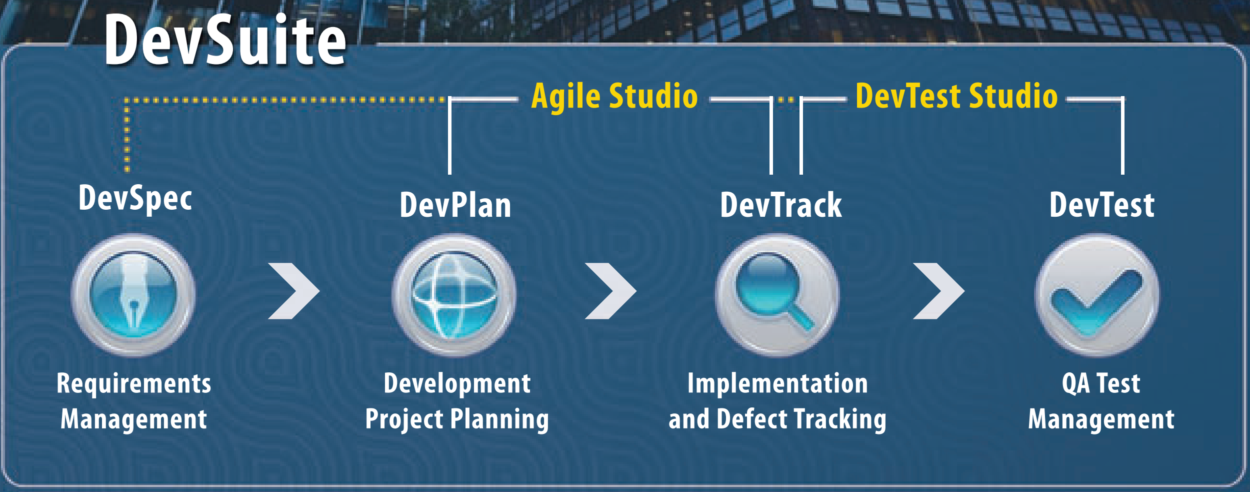
Try our Jump Start Package Today

(FIRST 10 USERS FREE)

Call 1.800.439.7782 or visit www.techexcel.com

TechExcel

Knowledge-Centric ALM for Today's Enterprises



With DevSuite, you can deploy one module, a multi-module solution or the complete fully integrated ALM solution

- DevSpec – Requirements management
- DevPlan – Project planning and tracking
- DevTrack – Defect tracking and task management
- DevTest – Test management
- Agile Studio – Agile planning and implementation
- DevTest Studio – Complete quality management
- DevSuite – Complete Application Lifecycle Management

1.800.439.7782

www.techexcel.com

there a way to exploit the simplicity and elegance of PLINQ without sacrificing performance?

Although it's impossible to determine exactly how PLINQ processes the query (Lines 16-27 in **Figure 3**), it's probable that PLINQ has no good way to partition the input lines for parallel processing by separate tasks. As a working hypothesis, assume that partitioning may be the cause of the PLINQ performance problem.

PLINQ provides an excellent model for processing in-memory data structures.

From Magennis' book (pp. 276-279), the lines String array supports the IEnumerable<String> interface (see also John Sharp's book, "Microsoft Visual C# 2010 Step by Step" [Microsoft Press, 2010], Chapter 19). However, lines isn't indexed, so PLINQ probably uses "chunk partitioning." Furthermore, the IEnumerator.MoveNext methods for the FileMmChar and FileMmByte classes are slow because they need to scan every character until the next new line is located.

What would happen if the lines String array were indexed? Could we improve PLINQ performance, especially when supplemented by memory mapping the input file? *Geonames Index* shows that this technique does improve performance, yielding results comparable to native code. In general, however, either there's a necessary up-front cost to move the lines into an in-memory list or array, which is indexed (the cost can then be amortized over multiple queries), or the file or other data source is already indexed, perhaps during generation in an earlier program step, eliminating the preprocessing cost.

The up-front indexing operation is simple; just access each line, one at a time, and then add the line to a list. Use the list object in Lines 16-17, as illustrated in **Figure 3**, and in this code snippet, which shows the preprocessing:

```
// Preprocess the file to create a list of byte[] lines
List<byte[]> lineListByte = new List<byte[]>();
var lines =
      FileMmByte.ReadLines(Path.Combine(Environment.CurrentDirectory, inFile));
// ... Multiple queries can use lineListByte
// ....
foreach (byte[] line in lines) { lineListByte.Add(line); }
// ....
var q = from line in lineListByte.AsParallel().
      WithDegreeOfParallelism(degreeOfParallelism)
```

Note that it's slightly more efficient to process the data further by converting the list to an array, although this increases the preprocessing time.

A Final Performance Enhancement

Geonames Index performance can be improved still further by indexing the fields in each line so that the ExtractIntegerField method doesn't need to scan all the characters in a line out to the specified field.

The implementation, *Geonames IndexFields*, modifies the ReadLines method so that a returned line is an object containing both a byte[] array and a uint[] array containing the locations of each field. This results in about a 33 percent performance improvement over *Geonames Index* and brings the performance fairly close to the native and C#/.NET solutions. (*Geonames IndexFields* is included with the code download.) Furthermore, it's now much easier to construct more general queries as the individual fields are readily available.

Limitations

The efficient solutions all require memory-resident data, and the performance advantages don't extend to very large data collections. "Very large" in this case refers to data sizes that approach the system physical memory size. In the *Geonames* example, the 3,302MB file (four copies of the original file) could be processed on the 8GB test system. A test with eight concatenated copies of the file, however, was very slow with all the solutions.

As mentioned earlier, performance is also best when the data files are "live," in the sense that they've been accessed recently and are likely to be in memory. Paging in the data file during the initial run can take 10 or more seconds and is comparable to the indexing operation in the earlier code snippet.

In summary, the results in this article apply to memory-resident data structures, and today's memory sizes and prices allow significant data objects, such as a file of 7.25 million place names, to be memory-resident.

Additional Test System Results

Figure 5 shows test results on an additional system (Intel i7 860, 2.80GHz, four cores, eight threads, Windows 7, 4GB RAM). The processor supports hyper-threading, so the tested DoP values are 1, 2, ..., 8. **Figure 1** is based on a test system with six AMD cores; this system doesn't support hyper-threading..

Two additional test configurations produced similar results (and the full data is available on my Web site):

- Intel i7 720, 1.60GHz, four cores, eight threads, Windows 7, 8GB RAM
- Intel i3 530, 2.93GHz, two cores, four threads, Windows XP64, 4GB RAM

Interesting performance characteristics include:

- *Geonames Threads* consistently provides the best perfor-

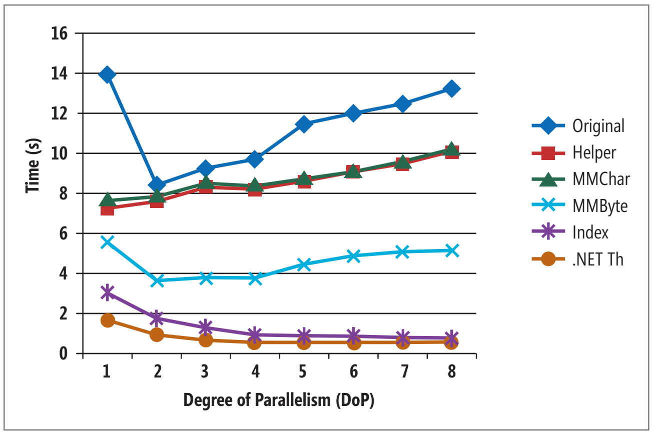


Figure 5 **Intel i7 860, 2.80GHz, Four Cores, Eight Threads, Windows 7, 4GB RAM**

mance, along with *Geonames Native*.

- *Geonames Index* is the fastest PLINQ solution, approaching the *Geonames Threads* performance. *Note: Geonames IndexFields is slightly faster but isn't shown in* **Figure 5**.
- Other than *Geonames Index*, all PLINQ solutions scale *negatively* with the DoP for DoP greater than two; that is, performance decreases as the number of parallel tasks increases. From this example, PLINQ produces good performance only when used with indexed objects.

None of the simple enhancements provide performance close to that of native or multithreaded C#/.NET code.

- The hyper-threading performance contribution is marginal. Therefore, *Geonames Threads* and *Geonames Index* performance doesn't increase significantly for DoP greater than four. This poor HT scalability might be a consequence of scheduling two threads on logical processors on the same core, rather than assuring that they run on distinct cores when possible. However, this explanation doesn't appear to be plausible as Mark E. Russinovich, David A. Solomon and Alex Ionescu say that physical processors are scheduled before logical processors on p. 40 of their book, "Windows Internals, Fifth Edition" (Microsoft Press, 2009). The AMD systems without HT (**Figure 1**) provided about three to four times the performance with DoP greater than four, compared to the sequential DoP one results for *Threads*, *Native* and *Index*. **Figure 1** shows that the best performance occurs when the DoP is the same as the core count, where the multithreaded performance is 4.2 times the DoP one performance.

Results Summary

PLINQ provides an excellent model for processing in-memory data structures, and it's possible to improve existing code performance with some simple changes (for example, *Helper*) or with more advanced techniques, as was shown with *MMByte*. However, none of the simple enhancements provide performance close to that of native or multithreaded C#/.NET code. Furthermore, the enhancement doesn't scale with the core count and DoP.

PLINQ can come close to native and C#/.NET code performance, but it requires the use of indexed data objects.

Using the Code and Data

All code is available on my Web site (jmhartsoftware.com/SequentialFileProcessingSupport.html), following these instructions:

- Go to the page to download a ZIP file containing PLINQ code and *Geonames Threads* code. All PLINQ variations are in the GeonamesPLINQ project (Visual Studio 2010; Visual Studio 2010 Express is sufficient). *Geonames Threads* is in the GeonamesThreads Visual Studio 2010 project. The projects are both configured for 64-bit release builds. The ZIP file also contains a spreadsheet with the raw performance data used in **Figures 1**, **2** and **5**. A simple "Usage" comment at the head of the file explains the command-line option to select the input file, DoP and implementation.
- Go to the Windows System Programming support page (jmhartsoftware.com/comments_updates.html) to download the *Geonames Native* code and projects (a ZIP file), where you'll find the Geonames project. A ReadMe.txt file explains the structure.
- Download the GeoNames database from download.geonames.org/export/dump/allCountries.zip.

Unexplored Questions

This article compared the performance of several alternative techniques to solve the same problem. The approach has been to use standard interfaces as they're described and to assume a simple shared-memory model for the processors and threads. There was, however, no significant effort to probe deeply into the underlying implementations or specific features of the test machines, and numerous questions could be investigated in the future. Here are some examples:

- What's the effect of cache misses, and is there any method to reduce the impact?
- What would be the impact of solid-state disks?
- Is there any way to reduce the performance gap between the PLINQ *Index* solution and the *Threads* and *Native* solutions? Experiments reducing the amount of data copying in the FileMmByte IEnumerator.MoveNext and Current methods didn't show any significant benefit.
- Is the performance close to the theoretical maximum as determined by memory bandwidth, CPU speed and other architectural features?
- Is there a way to get scalable performance on HT systems (see **Figure 5**) that's comparable to systems without HT (**Figure 1**)?
- Can you use profiling and Visual Studio 2010 tools to identify and remove performance bottlenecks?

I hope you're able to investigate further. ■

JOHNSON (JOHN) M. HART *is a consultant specializing in Microsoft Windows and Microsoft .NET Framework application architecture and development, technical training and writing. He has many years of experience as a software engineer, engineering manager and architect at Cilk Arts Inc. (since acquired by Intel Corp.), Sierra Atlantic Inc., Hewlett-Packard Co. and Apollo Computer. He served as computer science professor for many years and has authored four editions of "Windows System Programming" (Addison-Wesley, 2010).*

THANKS to the following technical experts for reviewing this article: *Michael Bruestle, Andrew Greenwald, Troy Magennis and CK Park*

Use Multiple Visual Studio Project Types for Cloud Success

Patrick Foley

As you've probably noticed, there are many different project types in Visual Studio these days. Which one do you choose? All have strengths that help solve problems in different situations. Even within a single business problem, there are often multiple use cases that can best be solved by different Visual Studio project types.

I was confronted with a real-world example of such a problem recently while building out the infrastructure for a cloud-based program I lead that's designed to highlight success stories: Microsoft Solutions Advocates (microsoftsolutionsadvocates.com). I used several different Visual Studio project types to build my solution, and in this article, I'll walk through a simplified example of my project called "Customer Success Stories," or CSS.

This article discusses:

- Creating an Entity Framework data model
- Creating an ASP.NET Dynamic Data project
- Creating a Windows Azure service project
- Creating an ASP.NET MVC project
- Creating a WCF RIA Services project
- Adding the final touches for a complete solution

Technologies discussed:

ASP.NET MVC, WCF RIA Services, ASP.NET Dynamic Data

Code download available at:

code.msdn.microsoft.com/mag201101VSCloud

CSS has three distinct use cases:

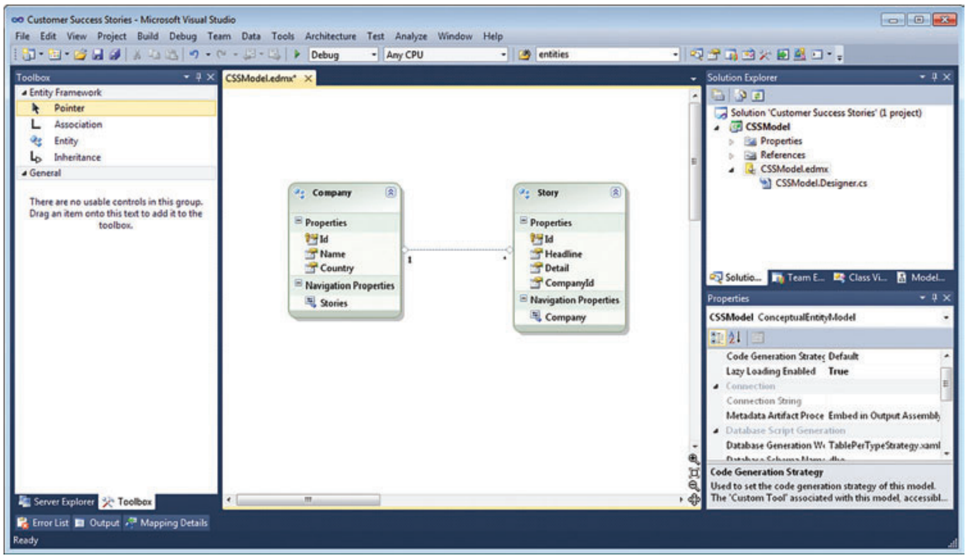
1. Anonymous users read success stories on a public Web site.
2. Users who belong to the program log in to a private Web site to create and edit their own success stories.
3. Administrators (like me) log in to an administrative Web site to manage and edit all the data, including minutia such as lookup tables.

The approach I settled on combined three Microsoft .NET Framework technologies:

1. ASP.NET MVC for the public site
2. WCF RIA Services for the private, customer-edit site
3. ASP.NET Dynamic Data for the administrative site

Any of these technologies could've been used to create the whole solution, but I preferred to take advantage of the best features of each. ASP.NET MVC is an ideal technology for creating a public Web site that will work everywhere, because it emits standard HTML, for one thing. The public site has a marketing purpose, so I'll eventually engage a designer to polish the appearance. Working with a designer adds complexity, but ASP.NET MVC has a straightforward view implementation that makes it easy to incorporate a designer's vision. Making the site read-only and separating it from the other use cases helps isolate the scope of the designer's involvement.

Although ASP.NET MVC could also be used to implement the customer-editing functionality, WCF RIA Services is an even better fit. (Conversely, WCF RIA Services could be used to build a great-looking public Web site, but Silverlight isn't supported on some devices, such as iPhones and iPads, and I wanted the greatest



CSSModel ConceptualEntityModel

Code Generation Strateg	Default
Lazy Loading Enabled	True
Connection	
Connection String	
Metadata Artifact Proce	Embed in Output Assembly
Database Script Generation	
Database Generation W	TablePerTypeStrategy.xaml

Code Generation Strategy
Used to set the code generation strategy of this model. The 'Custom Tool' associated with this model, accessibl...

Figure 1 **Adding Company and Story Entities to the Visual Studio Project**

reach for the public use case.) Silverlight is here to stay, and it's perfect for creating a rich editing experience with very little programming, so long as it's reasonable to expect users to have it or install it, as would be the case with customers collaborating on a success-stories site.

ASP.NET Dynamic Data provides a handy way to build an administrative solution without too much work. The administrative site doesn't need to be fancy; it simply needs to provide a way to manage all of the data in the solution without having to resort to SQL Server Management Studio. As my solution evolves, the ASP.NET Dynamic Data site could conceivably be subsumed by the WCF RIA Services site. Nevertheless, it's useful at the beginning of a data-centric development project such as this one, and it costs almost nothing to build.

Targeting Windows Azure

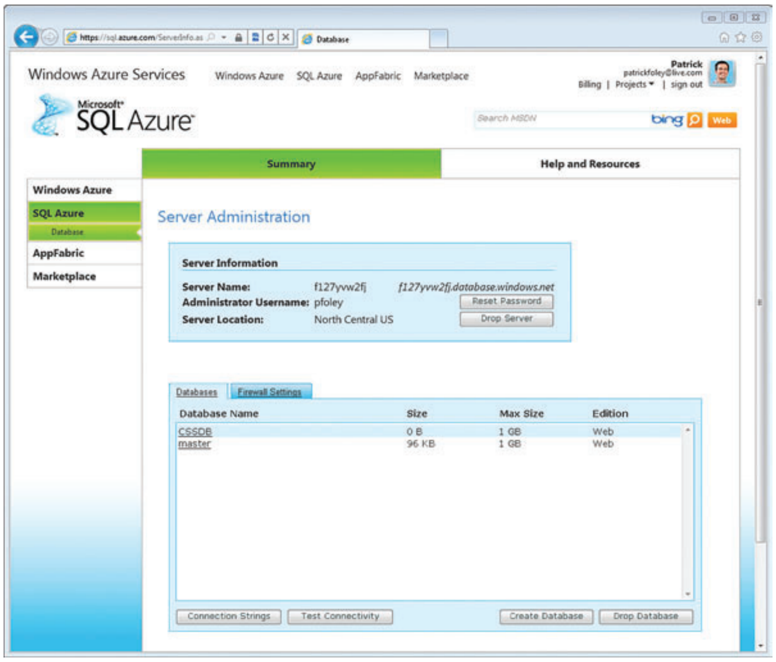
Again, this example is based on a real-world problem, and because the solution requires a public Web site, I'm going to target Windows Azure and SQL Azure. Windows Server and SQL Server might be more familiar, but I need the operational benefits of running in the cloud (no need to maintain the OS, apply patches and so on). I barely have time to *build* the solution—I certainly don't have time to *operate* it, so Windows Azure is a must for me.

To work through this example and try it on Windows Azure, you need an account. There are various options and packages found at microsoft.com/windowsazure/offers. MSDN subscribers and Microsoft partners (including BizSpark startups—visit bizspark.com to learn more) have access to several months of free resources. For prototyping and learning (such as working through this example), you can use the "Introductory Special." It includes three months of a 1GB SQL Azure database and 25 hours of a Windows Azure small compute instance, which should be enough to familiarize yourself with the platform. I built this example using the Introductory Special without incurring any additional charges.

'Customer Success Stories' Overview

This example is presented in the form of a tutorial. Several important aspects of a real-world implementation are out of scope for this article, including user-level security, testing, working with a designer and evolving beyond an extremely simplified model. I'll attempt to address these in future articles or on my blog at pfoley.com.

The steps are presented at a high level and assume some familiarity with Visual Studio and with the technologies involved. Code for the entire solution can be downloaded from code.msdn.microsoft.com/mag201101VSCloud, and click-by-click instructions for each step can be found at pfoley.com/mm2011jan.



Server Information

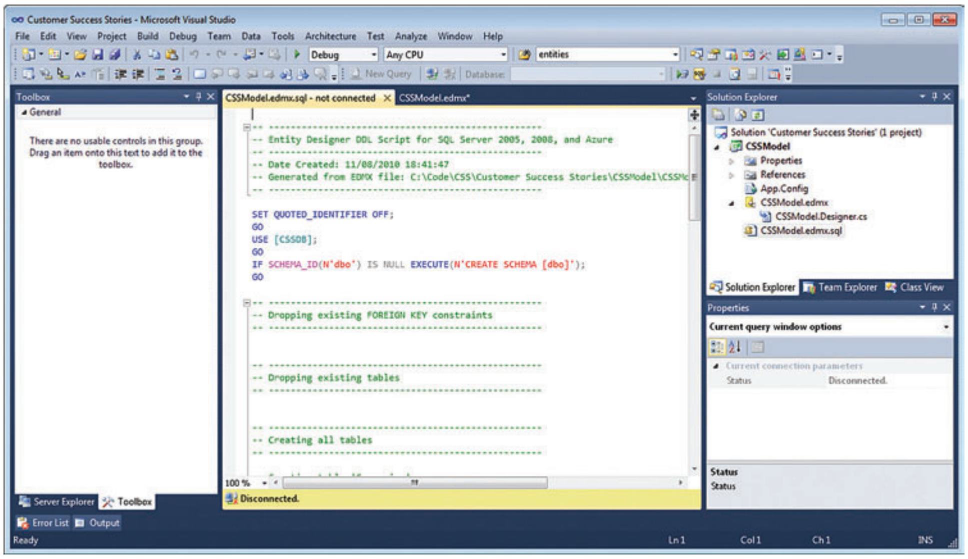
Server Name:	f127yvw2fj	*f127yvw2fj.database.windows.net*
Administrator Username:	pfoley	
Server Location:	North Central US	

Database Name	Size	Max Size	Edition
CSSDB	0 B	1 GB	Web
master	96 KB	1 GB	Web

Figure 2 **Configuring Settings in the SQL Azure Portal**

Step 1: Create a Project for the Entity Framework Data Model

The Web technologies used in this example all use the Entity Framework effectively, so I chose to integrate the three use cases by having them all share a common Entity Framework model. When working with an existing database, you have to generate the model from the database. Whenever possible, I prefer to create the model first and generate the database from it, because I like to think about my design more at the model level than the database level. To keep things simple, this example uses just two entities: Company and Story.



```
-- --------------------------------------------------
-- Entity Designer DDL Script for SQL Server 2005, 2008, and Azure
-- --------------------------------------------------
-- Date Created: 11/08/2010 18:41:47
-- Generated from EDMX file: C:\Code\CSS\Customer Success Stories\CSSModel\CSSMc
-- --------------------------------------------------

SET QUOTED_IDENTIFIER OFF;
GO
USE [CSSDB];
GO
IF SCHEMA_ID(N'dbo') IS NULL EXECUTE(N'CREATE SCHEMA [dbo]');
GO

-- --------------------------------------------------
-- Dropping existing FOREIGN KEY constraints
-- --------------------------------------------------



-- --------------------------------------------------
-- Dropping existing tables
-- --------------------------------------------------



-- --------------------------------------------------
-- Creating all tables
-- --------------------------------------------------
```

Customer Success Stories - Microsoft Visual Studio
File Edit View Project Build Debug Team Data Tools Architecture Test Analyze Window Help
Debug
Any CPU
entities
New Query
Database:
Toolbox
General
There are no usable controls in this group. Drag an item onto this text to add it to the toolbox.
CSSModel.edmx.sql - not connected
CSSModel.edmx*
Solution Explorer
Solution 'Customer Success Stories' (1 project)
CSSModel
Properties
References
App.Config
CSSModel.edmx
CSSModel.Designer.cs
CSSModel.edmx.sql
Solution Explorer
Team Explorer
Class View
Properties
Current query window options
Current connection parameters
Status
Disconnected.
100 %
Disconnected.
Server Explorer
Toolbox
Error List
Output
Ready
Ln 1
Col 1
Ch 1
INS

Figure 3 **Generating the Database Model**

entities with an association between them as in **Figure 1** (when you right-click on Company to add the association, make sure that "Add foreign key properties to 'Person' Entity" is checked on the ensuing dialog—foreign key properties are required in future steps).

The model is now ready to generate database tables, but a SQL Azure database is needed to put them in. When prototyping is completed and the project is evolving, it's useful to add a local SQL Server database for testing purposes, but at this point, it's less complicated to work directly with a SQL Azure database.

From your SQL Azure account portal, create a new database called "CSSDB" and add rules for your current IP address and to "Allow Microsoft Services access to this server" on the Firewall Settings tab. Your SQL Azure account portal should look something like **Figure 2**.

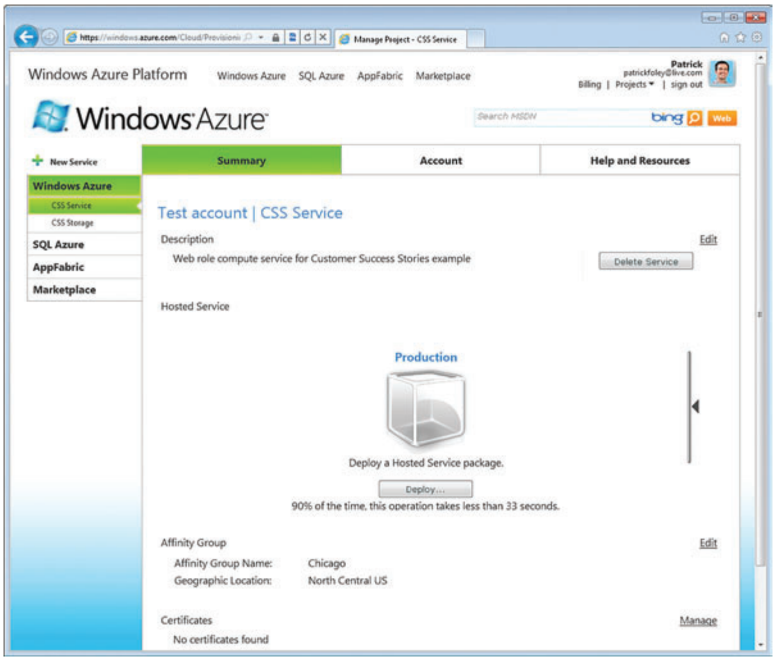
In Visual Studio, right-click on the design surface and select "Generate Database from Model." Add a connection to your new SQL Azure database and complete the wizard, which generates some Data Definition Language (DDL) and opens it in a .sql file, as shown in **Figure 3**.

Before you can execute the SQL, you must connect to the SQL Azure database (click the Connect button on the Transact-SQL Editor toolbar). The "USE" statement is not supported in SQL Azure, so you must choose your new database from the Database dropdown on the toolbar and then execute the SQL. Now you have a SQL Azure database that you can explore in Visual Studio Server Explorer, SQL Server Management Studio or the new management tool, Microsoft Project Code-Named "Houston" (sqlazurelabs.com/houston.aspx). Once you build the solution, you have an Entity Framework project that you can use to access that database programmatically, as well.

The Entity Framework model will be created in its own project and shared across multiple projects (I learned how to do this from Julie Lerman; see pfoley.com/mm2011jan01 for more on that). I call best practices like these "secret handshakes"—figuring it out the first time is a challenge, but once you know the secret, it's simple:

1. Create the Entity Framework model in a "class library" project.
2. Copy the connection string into any projects that share the model.
3. Add a reference to the Entity Framework project and System.Data.Entity in any projects that share the model.

To start, create a Blank Solution in Visual Studio 2010 named "Customer Success Stories" and add a Class Library project named "CSSModel." Delete the class file and add an empty ADO.NET Entity Data Model item named "CSSModel." Add Company and Story



Description
Web role compute service for Customer Success Stories example

Hosted Service

Deploy a Hosted Service package.

90% of the time, this operation takes less than 33 seconds.

Affinity Group
Affinity Group Name: Chicago
Geographic Location: North Central US

Certificates
No certificates found

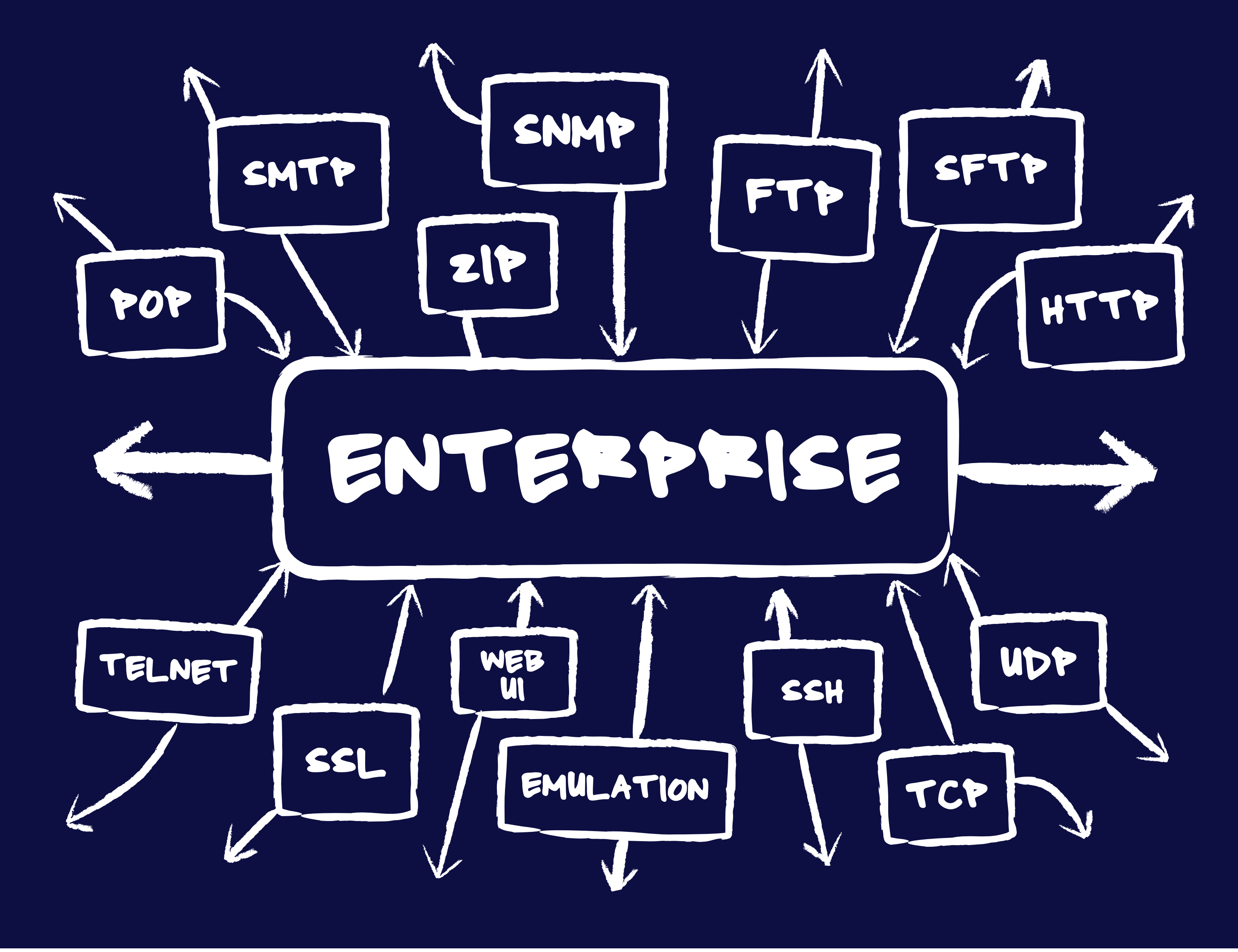
Figure 4 **Creating Services in the Windows Azure Portal**

Step 2: Create the ASP.NET Dynamic Data Project

An ASP.NET Dynamic Data Web site provides a simple way to work with all the data in the database and establishes baseline functionality to ensure the environment is working properly—all with one line of code.

Add a new ASP.NET Dynamic Data Entities Web Application project to the solution and call it "CSSAdmin." To use the data model from the first step, copy the connectionStrings element from App.Config in CSSModel to web.config in CSSAdmin. Set CSSAdmin as the startup project and add references to the CSSModel project and System.Data.Entity.

There are lots of fancy things you can do with ASP.NET Dynamic Data projects, but it's surprisingly useful to



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

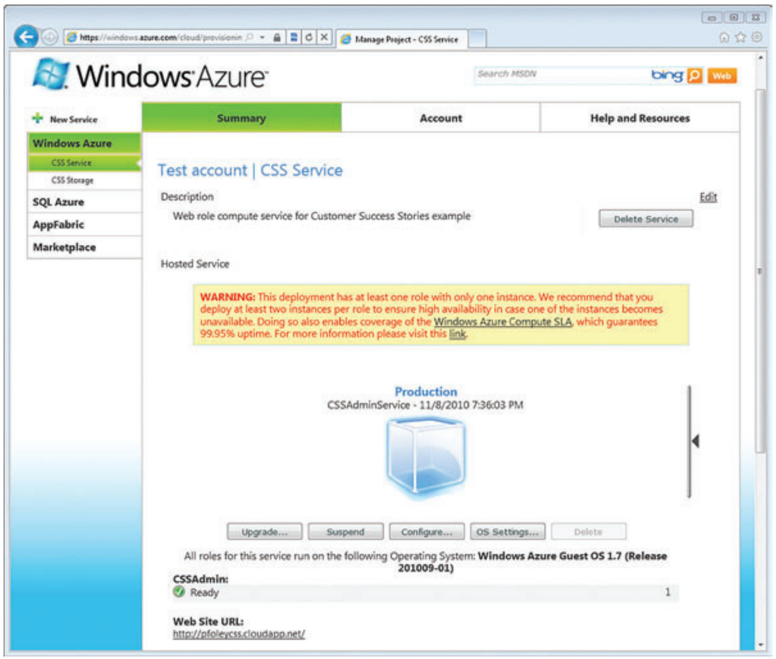
Add high performance Internet connectivity to your ActiveX, .NET and Compact Framework projects. Reduce integration costs with detailed documentation, hundreds of samples and an expert in-house support staff.

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



https://windows.azure.com/cloud/provisionin

Manage Project - CSS Service

Windows Azure

Search MSDN

New Service

Windows Azure

CSS Service

CSS Storage

SQL Azure

AppFabric

Marketplace

Summary | Account | Help and Resources

Test account | CSS Service

Description

Web role compute service for Customer Success Stories example

Edit

Delete Service

Hosted Service

WARNING: This deployment has at least one role with only one instance. We recommend that you deploy at least two instances per role to ensure high availability in case one of the instances becomes unavailable. Doing so also enables coverage of the Windows Azure Compute SLA, which guarantees 99.95% uptime. For more information please visit this link.

Production

CSSAdminService - 11/8/2010 7:36:03 PM

Upgrade... | Suspend | Configure... | OS Settings... | Delete

All roles for this service run on the following Operating System: **Windows Azure Guest OS 1.7 (Release 201009-01)**

CSSAdmin:

Ready 1

Web Site URL:

http://pfoleycss.cloudapp.net/

Figure 5 **A Windows Azure Deployed Service**

implement the default behavior that comes by simply uncommenting the RegisterContext line in Global.asax.cs and changing it to:

```
DefaultModel.RegisterContext(typeof(CSSModel.CSSModelContainer), new
ContextConfiguration() { ScaffoldAllTables = true });
```

Build and run the project, and you have a basic site to manage your data. Add some test data to make sure everything's working.

Step 3: Create the Windows Azure Services Project

The result of the previous step is a local Web site that accesses a database on SQL Azure—the next step is it to get that Web site running on Windows Azure.

From your Windows Azure account portal, create a Storage Service called "CSS Storage" and a Hosted Service called "CSS Service." Your Windows Azure account portal should look similar to **Figure 4**.

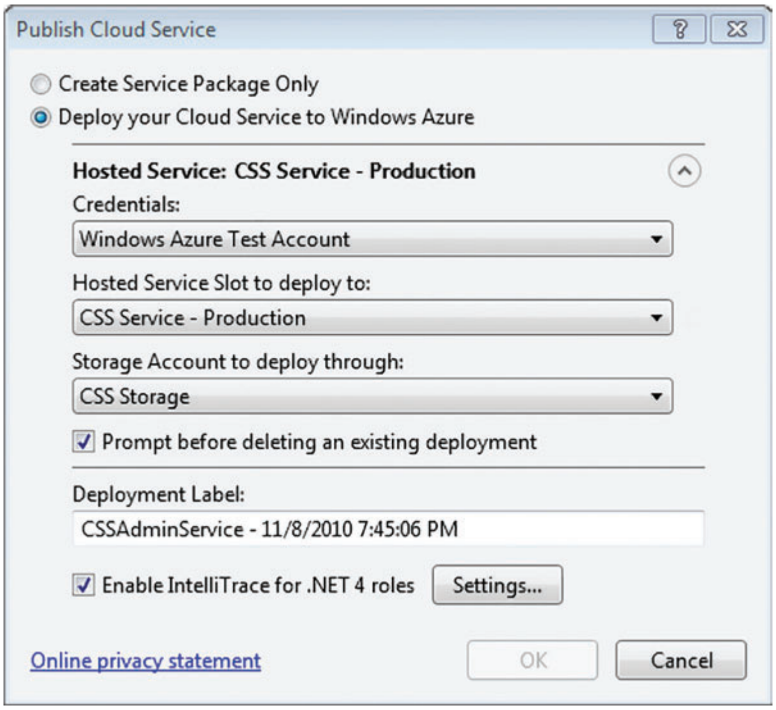
In Visual Studio, add a new Windows Azure Cloud Service project to your solution called "CSSAdmin-Service" (you must have Windows Azure Tools for Visual Studio installed), but don't add additional "cloud service solutions" from the wizard. The cloud service project adds the infrastructure necessary to run your application in a local version of the "cloud fabric" for development and debugging. It also makes it easy to publish to Windows Azure interactively. This is great for prototyping and for simpler Windows Azure solutions, but once you get serious about developing on Windows Azure, you'll probably want to use Windows PowerShell to script deployment, perhaps as part of a continuous integration solution.

Right-click on the Roles folder in CSSAdminService, then select "Add | Web Role Project in solution" to associate the CSSAdmin project with the cloud service project. Now when you compile and run the solution, it runs in the development fabric. At this point, the solution doesn't look any different than it did running on IIS or Cassini, but it's important to run on the dev fabric anyway to catch mistakes such as using unsupported Windows APIs as you evolve a Windows Azure solution.

Deploy to your Windows Azure account by right-clicking on the CSSAdminService project and selecting Publish. The first time you do this, you'll need to add credentials (follow the instructions to copy a certificate to your Windows Azure account). Then select a "Hosted Service Slot" and a "Storage Account" to deploy your solution to. There are two options for the hosted service slot: production and staging. When updating a real-world solution in production, deploy first to staging to make sure everything works and then promote the staging environment into production. While prototyping, I prefer to deploy straight to production because I'm not going to leave the solution running anyway. Click OK to deploy to Windows Azure, which can take several minutes. Once complete, run your Windows Azure application using the Web site URL shown on the service page, which should look similar to **Figure 5**.

After verifying that the service works, suspend *and delete* the deployment to avoid charges (consumption-based plans are billed for any time a service package is deployed, whether or not it's actually running). Don't delete the service itself, because doing so returns the Web site URL back into the pool of available URLs. Obviously, when you're ready to flip the switch on a real production solution, you'll have to budget for running Windows Azure services nonstop.

When a service deployment fails, it doesn't always tell you *exactly* what's wrong. It usually doesn't even *tell* you something is wrong.



Publish Cloud Service

○ Create Service Package Only

● Deploy your Cloud Service to Windows Azure

Hosted Service: CSS Service - Production

Credentials:

Windows Azure Test Account

Hosted Service Slot to deploy to:

CSS Service - Production

Storage Account to deploy through:

CSS Storage

☑ Prompt before deleting an existing deployment

Deployment Label:

CSSAdminService - 11/8/2010 7:45:06 PM

☑ Enable IntelliTrace for .NET 4 roles

Settings...

Online privacy statement

OK

Cancel

Figure 6 **Enabling IntelliTrace While Publishing to Windows Azure**

/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

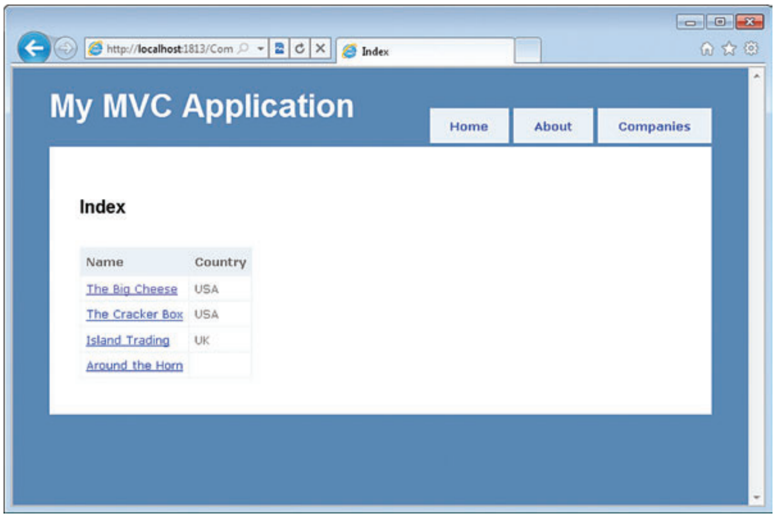
powered by

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



My MVC Application

Home | About | Companies

Index

Name	Country
The Big Cheese	USA
The Cracker Box	USA
Island Trading	UK
Around the Horn	

Figure 7 **A List of Companies in ASP.NET MVC Web Site**

The service status just enters a loop such as "Initializing … Busy … Stopping … Initializing …" When this happens to you—and it will—look for problems such as attempting to access local resources (perhaps a local SQL Server database) or referencing assemblies that don't exist on Windows Azure. Enabling IntelliTrace when you deploy the package (see **Figure 6**) can help you pinpoint problems by identifying the specific exceptions that are being thrown.

Step 4: Create the ASP.NET MVC Project

The solution so far consists of an administrative Web site (albeit with no user-level security) that runs on Windows Azure and accesses a SQL Azure database, all with one line of code. The next step is to create the public Web site.

Add a new ASP.NET MVC 2 Web Application project to the solution named "CSSPublic" (don't create a unit test project while working through this example). If you're already quite experienced with ASP.NET MVC, you might prefer to start with an ASP.NET MVC 2 *Empty Web* Application, but I prefer to start from a Web site structure that already works and modify it bit by bit to make it do what I want.

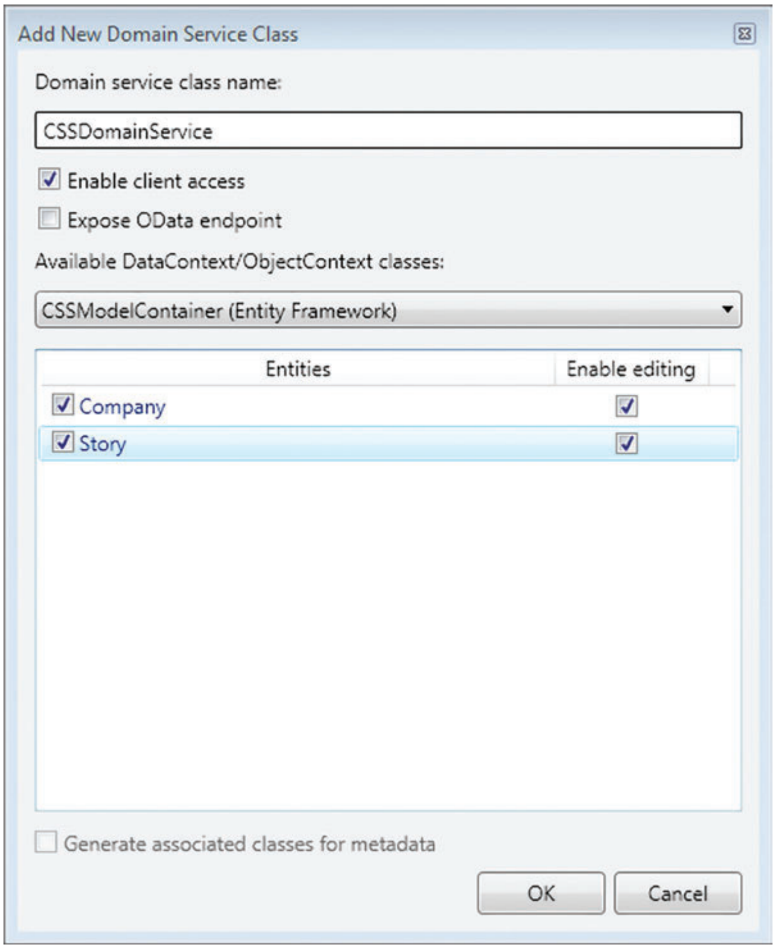
Right-click on CSSPublic to make it your startup project, and then run it to see what you're starting with. The public site for CSS is read-only and anonymous, so remove all login and account functionality with these steps:

1. Delete the "logindisplay" div from Site.Master.
2. Delete the ApplicationServices connection string and authentication element from the main Web.config.
3. Delete AccountController.cs, AccountModels.cs, LogOnUserControl.ascx and the entire Account folder under Views.
4. Run it again to make sure it still works.
5. Copy the connection string from the CSSModel App.Config into the CSSPublic Web.config and add references to CSSModel and System.Data.Entity as before.
6. Select all the references for CSSPublic and set the Copy Local property to true.

I think it makes sense to add separate controllers (with associated views) for Companies and Stories, while keeping the Home controller as a landing page. These are important decisions; a designer can make the site look good, but the information architecture—the site structure—has to be right first.

Naming is relevant in a Model View Controller (MVC) project. Use plural controller names (Companies, not Company) and matching view folders. Use the standard conventions of Index, Details and so on for controller methods and view names. You can tweak these conventions if you really want to, but that adds complexity.

Right-click on the Controllers folder to add a new controller named "CompaniesController." This site won't implement any behavior—it's read-only—so there's no need for an explicit model class. Treat the Entity Framework model container itself as the model. In CompaniesController.cs, add "using CSSModel" and change the Index method to return a list of companies as follows:



Add New Domain Service Class

Domain service class name: CSSDomainService

[✓] Enable client access

[ ] Expose OData endpoint

Available DataContext/ObjectContext classes: CSSModelContainer (Entity Framework)

Entities	Enable editing
[✓] Company	✓
[✓] Story	✓

[ ] Generate associated classes for metadata

Figure 8 **Adding a Domain Service Class**

```
CSSModelContainer db = new CSSModelContainer();
return View(db.Companies.ToList());
```

To create the view, create an empty Companies folder under Views and right-click on it to add a view called "Index." Make it strongly typed with a View data class of CSSModel.Company and View content of List.

In Site.Master, add a list item in the menu to reference the new Controller:

```
<li><%: Html.ActionLink("Companies", "Index", "Companies")%></li>
```

Run the app and click the menu to see the list of Companies. The default view is a good starting point, but remove the unnecessary "Id" field. Because this site is intended to be read-only, delete the ActionLink entries for "Edit," "Delete" and "Create." Finally, make the company name itself a link to the Details view:

```
<%: Html.ActionLink(item.Name, "Details", new { id=item.Id })%>
```

Your Companies list should now look similar to what is shown in **Figure 7**.

To implement Details, add a method to CompaniesController:

```
public ActionResult Details(int id)
{
  CSSModelContainer db = new CSSModelContainer();
  return View(db.Companies.Single(c => c.Id.Equals(id)));
}
```

The id parameter represents the integer following Companies\Details\ in the URL. That integer is used to find the appropriate company using a simple LINQ expression.

Add the Details view underneath the Companies folder as before, but this time select "Details" as the View content and name the view "Details."

Run the project, navigate to Companies, and then click one of the company names to see the default Details view.

Adding a controller and views for Stories is similar. The views still require a fair amount of tweaking before being ready for a designer, but this is a good start, and it's easy to evolve.

To verify this project will work on Windows Azure, create a cloud service project called "CSSPublicService" and add the CSSPublic role to it. Run the service locally in the dev fabric and then publish the site to Windows Azure and run it from the public URL. Don't forget to suspend and delete the deployment when you're done to avoid being billed.

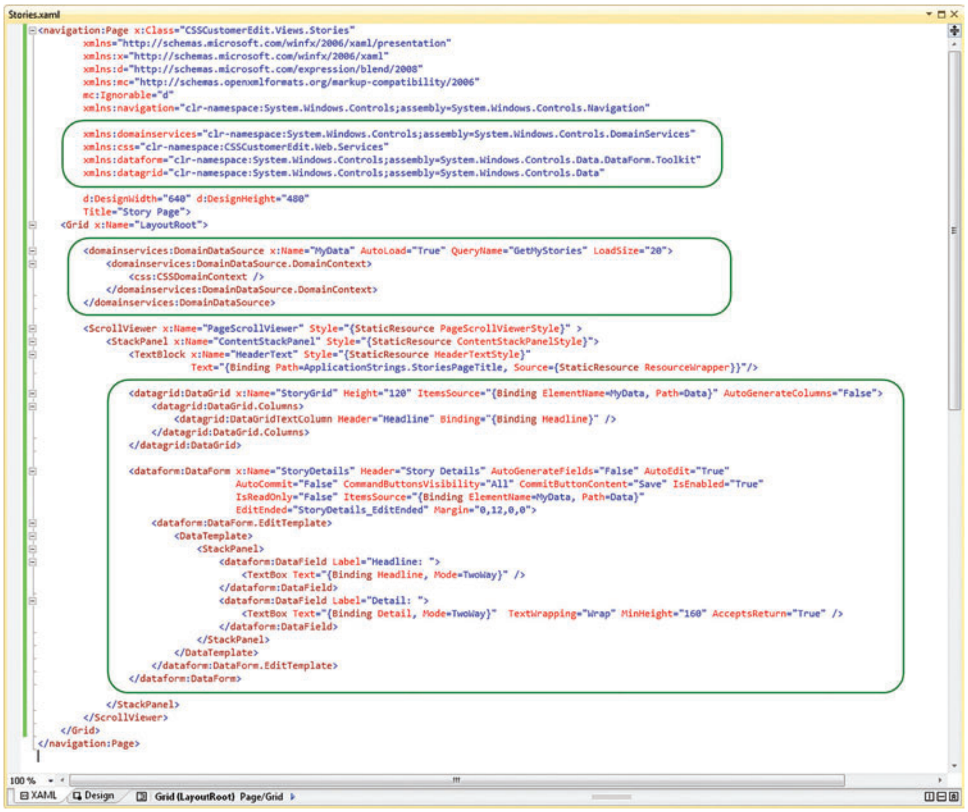
Step 5: Create the WCF RIA Services Project

The solution at this point contains ASP.NET MVC and ASP.NET Dynamic Data Web sites running (or at least runnable) on Windows Azure, using a shared Entity Framework model to access a SQL Azure database. Very little manual plumbing code has been required to obtain a decent amount of functionality. Adding a WCF RIA Services Web site adds another dimension: a rich editing experience.

Add a Silverlight Business Application project named "CSSCustomerEdit" (no spaces) and Visual Studio adds two projects to your solution: a Silverlight client (CSSCustomerEdit) and a Web service (CSSCustomerEdit.Web). Run the solution to see what you're starting with. Open ApplicationStrings.resx in CSSCustomerEdit project and change the value for ApplicationName to "Customer Success Stories" to make it look nice.

While prototyping, I prefer to deploy straight to production because I'm not going to leave the solution running anyway.

In CSSCustomerEdit.Web, copy connectionStrings from CSSModel into Web.config, add references to CSSModel and System.Data.Entity and set Copy Local to true for all the references. Then right-click on the Services folder and add a new Domain Service Class item called "CSSDomainService." Make sure the name of this class ends in Service—without a number—to gain the full benefit of



```
<navigation:Page x:Class="CSSCustomerEdit.Views.Stories"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
      mc:Ignorable="d"
      xmlns:navigation="clr-namespace:System.Windows.Controls;assembly=System.Windows.Controls.Navigation"

      xmlns:domainservices="clr-namespace:System.Windows.Controls;assembly=System.Windows.Controls.DomainServices"
      xmlns:css="clr-namespace:CSSCustomerEdit.Web.Services"
      xmlns:dataform="clr-namespace:System.Windows.Controls;assembly=System.Windows.Controls.Data.DataForm.Toolkit"
      xmlns:datagrid="clr-namespace:System.Windows.Controls;assembly=System.Windows.Controls.Data"

      d:DesignWidth="640" d:DesignHeight="480"
      Title="Story Page">
  <Grid x:Name="LayoutRoot">

      <domainservices:DomainDataSource x:Name="MyData" AutoLoad="True" QueryName="GetMyStories" LoadSize="20">
          <domainservices:DomainDataSource.DomainContext>
              <css:CSSDomainContext />
          </domainservices:DomainDataSource.DomainContext>
      </domainservices:DomainDataSource>

      <ScrollViewer x:Name="PageScrollViewer" Style="{StaticResource PageScrollViewerStyle}" >
          <StackPanel x:Name="ContentStackPanel" Style="{StaticResource ContentStackPanelStyle}">
              <TextBlock x:Name="HeaderText" Style="{StaticResource HeaderTextStyle}"
                         Text="{Binding Path=ApplicationStrings.StoriesPageTitle, Source={StaticResource ResourceWrapper}}"/>

              <datagrid:DataGrid x:Name="StoryGrid" Height="120" ItemsSource="{Binding ElementName=MyData, Path=Data}" AutoGenerateColumns="False">
                  <datagrid:DataGrid.Columns>
                      <datagrid:DataGridTextColumn Header="Headline" Binding="{Binding Headline}" />
                  </datagrid:DataGrid.Columns>
              </datagrid:DataGrid>

              <dataform:DataForm x:Name="StoryDetails" Header="Story Details" AutoGenerateFields="False" AutoEdit="True"
                                 AutoCommit="False" CommandButtonsVisibility="All" CommitButtonContent="Save" IsEnabled="True"
                                 IsReadOnly="False" ItemsSource="{Binding ElementName=MyData, Path=Data}"
                                 EditEnded="StoryDetails_EditEnded" Margin="0,12,0,0">
                  <dataform:DataForm.EditTemplate>
                      <DataTemplate>
                          <StackPanel>
                              <dataform:DataField Label="Headline: ">
                                  <TextBox Text="{Binding Headline, Mode=TwoWay}" />
                              </dataform:DataField>
                              <dataform:DataField Label="Detail: ">
                                  <TextBox Text="{Binding Detail, Mode=TwoWay}"  TextWrapping="Wrap" MinHeight="160" AcceptsReturn="True" />
                              </dataform:DataField>
                          </StackPanel>
                      </DataTemplate>
                  </dataform:DataForm.EditTemplate>
              </dataform:DataForm>

          </StackPanel>
      </ScrollViewer>
  </Grid>
</navigation:Page>
```

Figure 9 **XAML for the Stories View**

the tooling between the two projects (another "secret handshake"). Click OK to bring up the Add New Domain Service Class dialog and check all the entities along with Enable editing for each (**Figure 8**).

Notice that "Generate associated classes for metadata" is grayed out. This illustrates a tradeoff with the approach I'm espousing here. In a Silverlight Business Application, metadata classes can be used to add additional validation logic such as ranges and display defaults. However, when the Entity Framework model is in a separate project from CSSCustomerEdit.Web, the toolkit doesn't let you add these metadata classes. If this feature is important to you or if you know you're going to invest most of your energy in the Silverlight Business Application part of your solution, you might want to create your Entity Framework model directly in the ".Web" project instead of a separate project. You could still reference CSSCustomerEdit.Web to share the Entity Framework model in another project.

In Silverlight development, most of the magic involves editing XAML.

As mentioned, authentication and authorization are out of scope for this article, but it's possible to punt and still be precise. In the CSSDomainService class, add a placeholder property named "myCompany" to represent the company the user is authorized to edit. For now, hardcode it to 1, but eventually the login process will set it to the right company for the authenticated user.

Edit the CSSDomainService class to reflect the specific use case for the project: the user can update companies but not insert or delete them (an administrator does that in the ASP.NET Dynamic Data Web site), so remove those service methods. Also, the user can only edit the single company they work for, not a list of companies, so change GetCompanies to GetMyCompany. Similarly, change GetStories to GetMyStories and ensure that the user creates stories whose CompanyId is equal to myCompany:

```
private int myCompany = 1; // TODO: set myCompany during authentication
public Company GetMyCompany()
{
  return this.ObjectContext.Companies.Single(c=>c.Id.Equals(myCompany));
}
...
public IQueryable<Story> GetMyStories()
{
  return this.ObjectContext.Stories.Where(s=>s.CompanyId.Equals(myCompany));
}

public void InsertStory(Story story)
{
  story.CompanyId = myCompany;
  story.Id = -1; // database will replace with next auto-increment value
  ...
}
```

WCF RIA Services shines in the creation of editable, field-oriented interfaces, but it's important to start simply and add functionality slowly. The DataGrid and DataForm controls are powerful, but whenever I work too fast or try to add too much functionality at once, I end up messing up and having to backtrack. It's better to work incrementally and add one UI improvement at a time.

To implement a baseline UI for this example, add references to System.Windows.Controls.Data and System.Windows.Controls.DomainServices in CSSCustomerEdit. Create new views (Silverlight Page items) for Company (singular) and Stories, then mimic the XAML from the existing Home and About views. Edit MainPage.xaml to add new dividers and link buttons (alternatively, just co-opt the existing Home and About views to use for Company and Stories).

In Silverlight development, most of the magic involves editing XAML. In CSSCustomerEdit, add namespace entries, a DomainDataSource and a DataForm for the Company view. In addition, add a DataGrid for the Stories view. In both DataForms, handle the EditEnded event to call MyData.SubmitChanges. Stories.xaml, which should look similar to **Figure 9**.

Build it … run it … it works! A rich editing experience that's ready to evolve (see **Figure 10**).

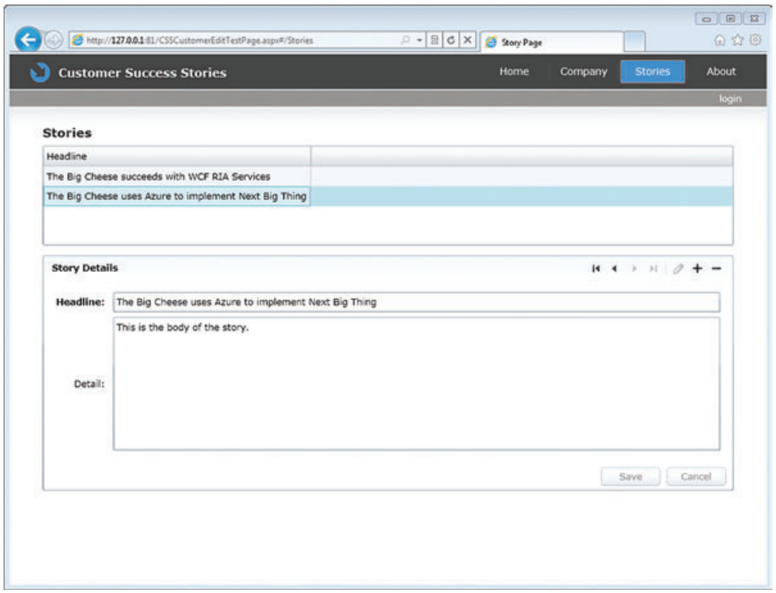
As before, create a new cloud service project, publish it and test it on Windows Azure. Copy CSSCustomerEditTestPage.aspx to Default.aspx for a cleaner experience, and you're done.

No 'One True Way'

Visual Studio and the .NET Framework provide myriad choices for creating solutions that can run on Windows Azure. While it's tempting to search for the "one true way" to create the next killer app for the cloud, it's more rewarding to leverage capabilities from multiple technologies to solve complex problems. It's easier to code, easier to evolve and—thanks to Windows Azure—easier to operate. ■

PATRICK FOLEY *is an ISV architect evangelist for Microsoft, which means he helps software companies succeed in building on the Microsoft platform. Read his blog at pfoley.com.*

THANKS to the following technical expert for reviewing this article: *Hanu Kommalapati*



Stories

Headline
The Big Cheese succeeds with WCF RIA Services
The Big Cheese uses Azure to implement Next Big Thing

Story Details

Headline: The Big Cheese uses Azure to implement Next Big Thing

Detail: This is the body of the story.

Figure 10 **The Stories View in Action**

Powerful Tools for Developers

High-Performance PDF Printer Driver

- Create accurate PDF documents in a fraction of the time needed with other tools
- WHQL tested for all Windows 32 and 64-bit platforms
- Produce fully compliant PDF/A documents
- Standard PDF features included with a number of unique features
- Interface with any .NET or ActiveX programming language

PDF Editor for .NET, now Webform Enabled

- Edit, process and print PDF 1.7 documents programmatically
- Fast and lightweight 32 and 64-bit managed code assemblies for Windows, WPF and Web applications
- Support for dynamic objects such as edit-fields and sticky-notes
- Save image files directly to PDF, with optional OCR
- Multiple image compression formats such as PNG, JBIG2 and TIFF

PDF Integration into Silverlight Applications

- Server-side PDF component based on the robust Amyuni PDF Creator ActiveX or .NET components
- Client-side C# Silverlight 3 control provided with source-code
- Optimization of PDF documents prior to converting them into XAML
- Conversion of PDF edit-boxes into Silverlight TextBox objects
- Support for other document formats such as TIFF and XPS

New Touchscreen Tablet for Mobile Development!

The DevTouch Pro is a new color touchscreen tablet designed to provide mobile application developers with a customizable development, testing and deployment platform.

Windows CE · android · Linux

- Fully open customizable tablet
- Develop with .NET, Java or C++
- Unrestricted development and flexible quantities
- Fully supported in North America

Wi Fi · Bluetooth · MADE IN CANADA FABRIQUÉ AU CANADA

Learn more at www.devtouchpro.com

Windows 7 · CERTIFIED FOR Windows Server 2008

More Development Tools Available at:

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

Build a Data-Driven Enterprise Web Site in 5 Minutes

James Henry

For years, developers have wrestled with the tedious tasks of building data layers with Create, Read, Update and Delete (CRUD) functionality to be consumed by the UI. I can personally remember a project from more than 10 years ago that required me to write code that automatically generated business and data layers from a Rational Rose object model. It was a lot of work.

A few years later Microsoft introduced the Microsoft .NET Framework and the DataSet object. With the DataSet designer, you could connect a strongly typed DataSet to a back-end database such as SQL Server and then work with the DataSet throughout the application. This minimized the need to work directly with SQL, but it still required the manual creation of Web pages to display data from the database.

Fast-forward a few more years and Microsoft introduced the Entity Framework. This Object Relational Mapping (ORM) framework uses LINQ to abstract a database schema and presents it to an application as a conceptual schema. Like the DataSet, this technology also minimized the need to work directly with SQL. However, it still required the manual creation of Web pages to display data.

Now Microsoft has introduced ASP.NET Dynamic Data—a combination of the Entity Framework and ASP.NET Routing—which allows an application to respond to URLs that do not physically exist. With these features you can create a production-ready, data-driven Web site in just a few minutes. In this article I'll show you how.

This article discusses:

- Starting a Dynamic Data project
- Using ASP.NET Routing
- Supporting metadata
- Customizing Dynamic Data templates

Technologies discussed:

ASP.NET, SQL Server

Getting Started

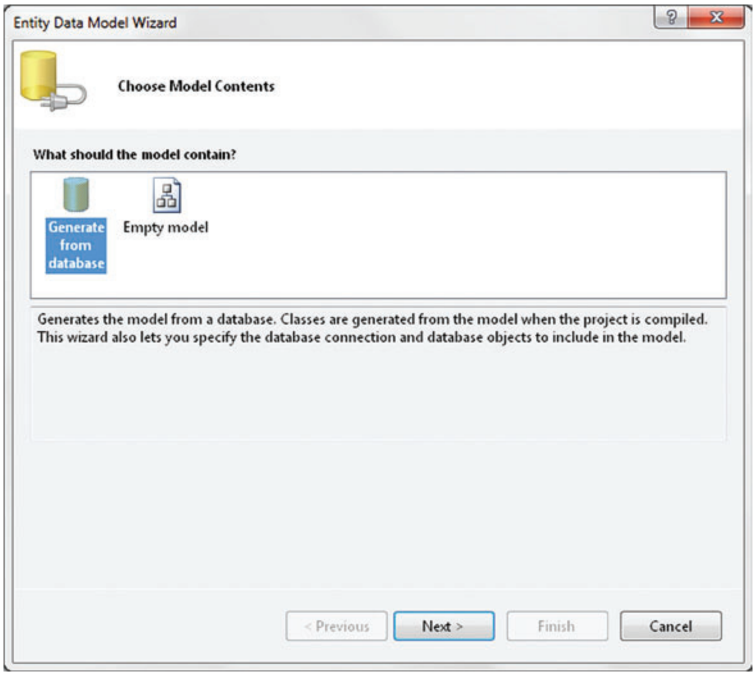
Just to set up the scenario, let's say I'm building an intranet Web site for a fictional company called Adventure Works. This Web site allows the company to manage employee information.

The company's data is stored in a Microsoft SQL Server 2008 database. (You can download and install the database from msftdbprodsamples.codeplex.com.)

Now, open Microsoft Visual Studio 2010 and create a new ASP.NET Dynamic Data Entities Web Site C# project.

The ASP.NET Dynamic Data Entities Web Site project type takes advantage of ASP.NET Routing and the Entity Framework to enable you to quickly create data-driven Web sites. To get this functionality, you need to add an Entity Framework data model to the project. To do this, choose Add New Item from the Web site menu. In the Add New Item dialog box, choose ADO.NET Entity Data Model. Name it HumanResources.edmx.

Visual Studio will then prompt you to add the model to the App_Code folder. Choose Yes and allow it to make this change.


Entity Data Model Wizard

Choose Model Contents

What should the model contain?

Generate from database

Empty model

Generates the model from a database. Classes are generated from the model when the project is compiled. This wizard also lets you specify the database connection and database objects to include in the model.

Figure 1 **Starting the Entity Data Model Wizard**

When you create Web site projects, Visual Studio dynamically compiles all code that is placed in the App_Code folder. In the case of the Entity Data Model, Visual Studio automatically generates a partial class for the data context and partial classes for the entities. In this example, it places the code in a file named HumanResources.Designer.cs.

Next, the Entity Data Model Wizard appears, as shown in **Figure 1**. Choose Generate from Database and click Next.

Now choose a connection to the Adventure Works database. If a connection doesn't already exist, you need to create a new one. **Figure 2** shows a connection to AdventureWorks on a SQL Server instance named Dev\BlueVision.

On the next page you can choose all tables in the Human Resources schema. The name of the schema appears in parenthesis. **Figure 3** shows some of the tables that are selected.

You can create a production-ready, data-driven Web site in just a few minutes.

When you click Finish, Visual Studio automatically generates entities from the tables that you chose for your project. **Figure 4** shows seven entities that Visual Studio generated from the database schema. Visual Studio used the foreign key constraints in the database to create relationships between the entities. For example, the Employee entity participates in a one-to-many relationship with the JobCandidate entity. This means that an employee can be a candidate for multiple jobs within the company.

Also note that the EmployeeDepartmentHistory entity joins the Employee and Department entities. Had the EmployeeDepartmentHistory table contained only the fields that were necessary for joining the Employee and Department tables, Visual Studio would have simply omitted the EmployeeDepartmentHistory entity. This would have allowed direct navigation between the Employee and Department entities.

Using ASP.NET Routing

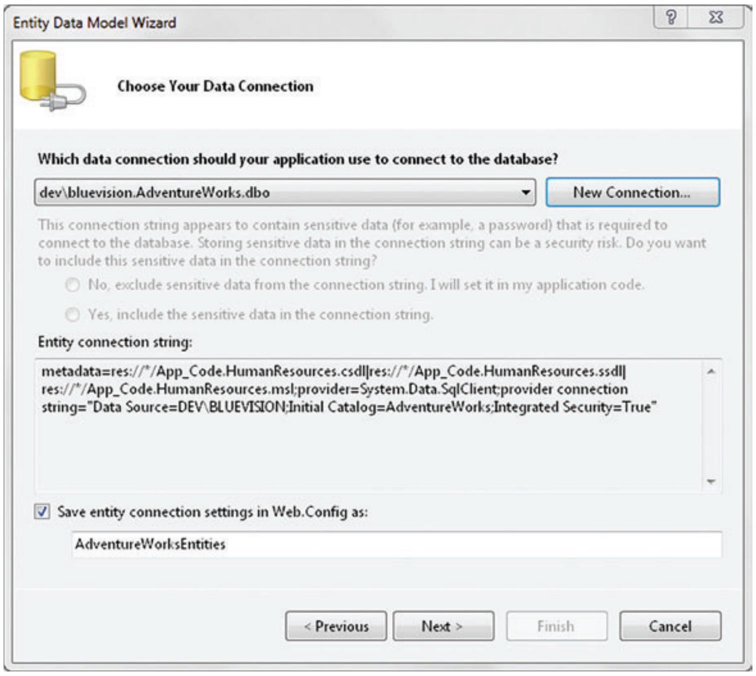
ASP.NET Routing allows an application to respond to URLs that do not physically exist. For example, two URLs—http://mysite/Employee and http://mysite/Department—could be directed to a page at http://mysite/Template.aspx. The page itself could then extract information from the URL to determine whether to display a list of employees or a list of departments, with both views using the same display template.

A route is simply a URL pattern that's mapped to an ASP.NET HTTP handler. It's the handler's responsibility to determine how the actual URL maps to the interpreted pattern. ASP.NET Dynamic Data uses a route handler named DynamicDataRouteHandler that interprets placeholders named {table} and {action} in URL patterns. For example, a handler could use this URL pattern:

```
http://mysite/{table}/{action}.aspx
```

That pattern could then be used to interpret the URL http://mysite/Employee/List.aspx.

Employee is processed as the {table} placeholder and List is processed as the {action} placeholder. The handler could then display


Entity Data Model Wizard

Choose Your Data Connection

Which data connection should your application use to connect to the database?

dev\bluevision.AdventureWorks.dbo

This connection string appears to contain sensitive data (for example, a password) that is required to connect to the database. Storing sensitive data in the connection string can be a security risk. Do you want to include this sensitive data in the connection string?

No, exclude sensitive data from the connection string. I will set it in my application code.

Yes, include the sensitive data in the connection string.

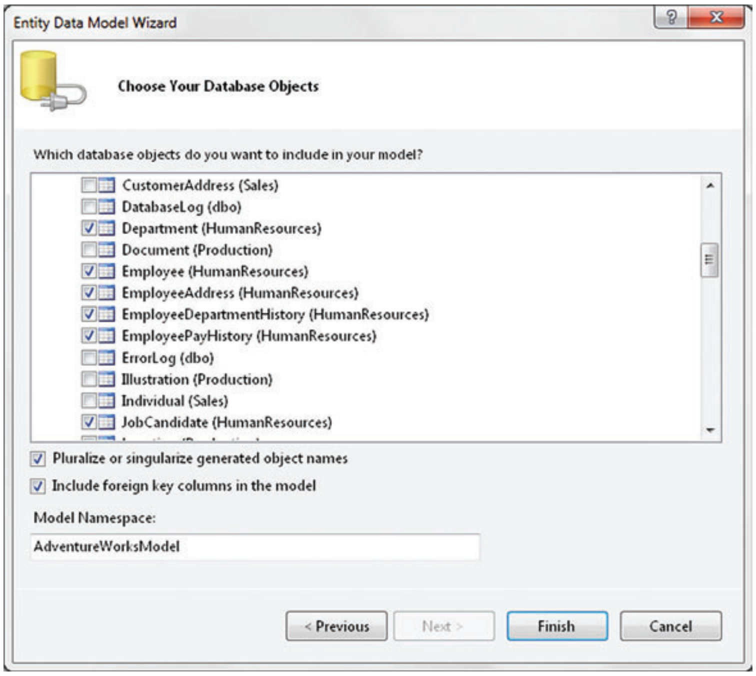
Entity connection string:

```
metadata=res://*/App_Code.HumanResources.csdl|res://*/App_Code.HumanResources.ssdl|
res://*/App_Code.HumanResources.msl;provider=System.Data.SqlClient;provider connection
string="Data Source=DEV\BLUEVISION;Initial Catalog=AdventureWorks;Integrated Security=True"
```

[✓] Save entity connection settings in Web.Config as:

AdventureWorksEntities

Figure 2 **Configuring the Data Connection**



Entity Data Model Wizard

Choose Your Database Objects

Which database objects do you want to include in your model?

- [ ] CustomerAddress (Sales)
- [ ] DatabaseLog (dbo)
- [✓] Department (HumanResources)
- [ ] Document (Production)
- [✓] Employee (HumanResources)
- [✓] EmployeeAddress (HumanResources)
- [✓] EmployeeDepartmentHistory (HumanResources)
- [✓] EmployeePayHistory (HumanResources)
- [ ] ErrorLog (dbo)
- [ ] Illustration (Production)
- [ ] Individual (Sales)
- [✓] JobCandidate (HumanResources)

[✓] Pluralize or singularize generated object names

[✓] Include foreign key columns in the model

Model Namespace:

AdventureWorksModel

< Previous | Next > | Finish | Cancel

Figure 3 **Selecting Tables for the Human Resources Schema in Visual Studio**

a list of Employee entity instances. DynamicDataRouteHandler uses the {table} placeholder to determine the name of the entity to display, and it uses the {action} parameter to determine the page template used to display the entity.

The Global.asax file contains a static method named RegisterRoutes that gets called when the application first starts, as shown here:

```
public static void RegisterRoutes(RouteCollection routes) {
  // DefaultModel.RegisterContext(typeof(YourDataContextType),
  //   new ContextConfiguration() { ScaffoldAllTables = false });
  routes.Add(new DynamicDataRoute("{table}/{action}.aspx") {
    Constraints = new RouteValueDictionary(
    new { action = "List|Details|Edit|Insert" }),
    Model = DefaultModel
  });
}
```

The first thing you need to do is uncomment the code that calls the RegisterContext method of the DefaultModel instance. This method registers the Entity Framework data context with ASP.NET Dynamic Data. You need to modify the line of code as follows:

```
DefaultModel.RegisterContext(
  typeof(AdventureWorksModel.AdventureWorksEntities),
  new ContextConfiguration() { ScaffoldAllTables = true });
```

The first parameter specifies the type of the data context, which is AdventureWorksEntities in this example. By setting the ScaffoldAllTables property to true, you allow all entities to be viewed on the site. Had you set this property to false, you'd need to apply the ScaffoldTable(true) attribute to each entity that should be viewed on the site. (In practice, you should set the ScaffoldAllTables property to false to prevent the accidental exposure of the entire database to end users.)

The Add method of the RouteCollection class adds a new Route instance to the route table. With ASP.NET Dynamic Data, the Route instance is actually a DynamicDataRoute instance. The DynamicDataRoute class internally uses DynamicDataRouteHandler as its handler. The parameter passed to the constructor of DynamicDataRoute represents the pattern that the handler should use to process physical URLs. A constraint is also set to limit the {action} values to List, Details, Edit or Insert.

This is theoretically all you need to do to use ASP.NET Dynamic Data. If you build and run the application, you should see the page shown in **Figure 5**.

Supporting Metadata

One thing you should notice is that the entity names are identical to the table names that they represent. In code and in the database this might be fine, however, usually this is not the case in the UI.

ASP.NET Dynamic Data makes it easy to change the name of a displayed entity by applying the Display attribute to the class that represents that entity. Because the entity classes are contained in a file that's automatically regenerated by Visual Studio, you should make any code changes to partial classes in a separate code file. Therefore, add a new code file named Metadata.cs to the App_Code folder. Then add the following code to the file:

```
using System;
using System.Web;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;

namespace AdventureWorksModel {
  [Display(Name = "Employee Addresses")]
  public partial class EmployeeAddress { }
}
```

Then rebuild and run the application. You should notice that EmployeeAddresses is now Employee Addresses.

Similarly, you can change the names of EmployeeDepartmentHistories, EmployeePayHistories and JobCandidates to more appropriate names.

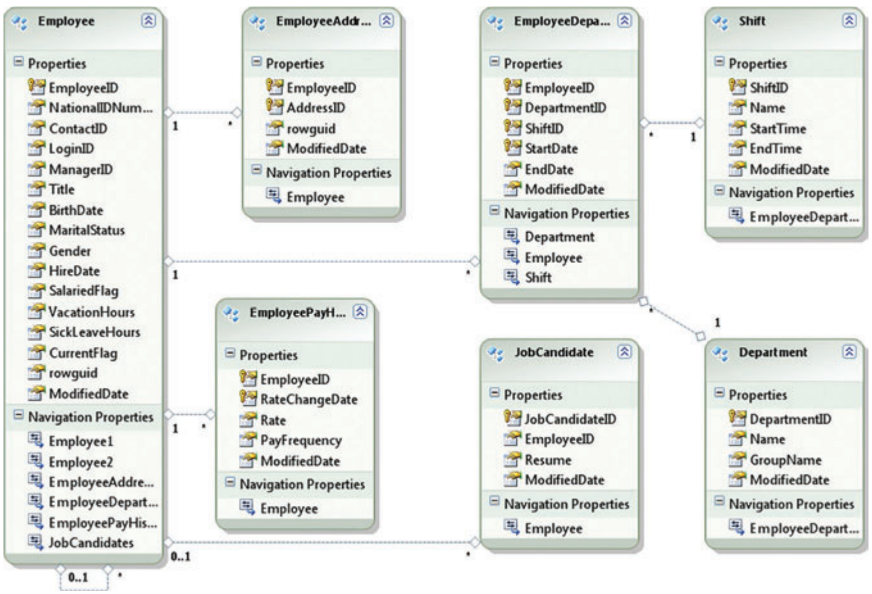
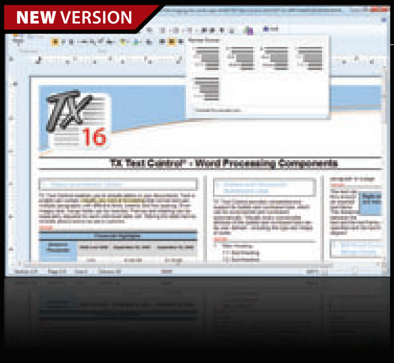


Figure 4 **Entities Generated from Database Tables in Visual Studio**

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



TX Text Control .NET for Windows Forms/WPF | 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 and WPF rich text box
- True WYSIWYG, nested tables, text frames, headers and footers, images, bullets, structured numbered lists, zoom, dialog boxes, section breaks, page columns
- Load, save and edit DOCX, DOC, PDF, PDF/A, RTF, HTML, TXT and XML

FusionCharts | from $195.02

InfoSoft Global empowering human thoughts

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

- Liven up your web applications using animated & data-driven charts
- Create AJAX-enabled charts with drill-down capabilities in minutes
- Export charts as images/PDF and data as CSV from charts itself
- Create gauges, dashboards, financial charts and over 550 maps
- Trusted by over 17,000 customers and 330,000 users in 110 countries

Spread for Windows Forms | from $959.04

GrapeCity

A comprehensive Excel compatible spreadsheet component for Windows Forms applications.

- Speed development with Spreadsheet Designers, Quick-start Wizard and Chart Designers
- Automatic completion - provide "type ahead" from user input to cell
- Features built-in tool to design charts with 85 different styles
- Preserves Excel files and restores unsupported functionality
- Includes sleek new predefined skins and the ability to create custom skins

ActiveReports 6 | from $685.02

GrapeCity

Latest release of the best selling royalty free .NET report writer.

- Now supports Windows Server 2008, 64Bit & IE8.0
- First Flash based Report Viewer for end users
- Supports PDF Digital Signatures, RSS Bar Codes and external stylesheets
- Features an easy-to-use Visual Studio report designer and a powerful API
- Offers seamless run-time deployment, royalty free

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

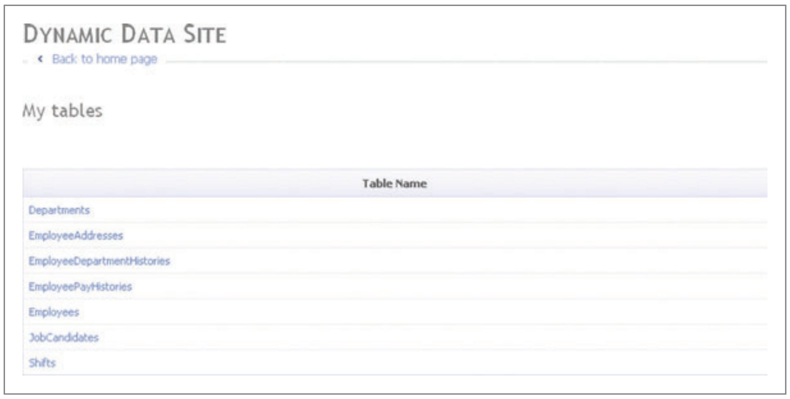
US Headquarters
ComponentSource
650 Claremore Prof Way
Suite 100
Woodstock
GA 30188-5188
USA

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

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

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

VISA DISCOVER
GSA Schedule Contract GS-35F-0168R



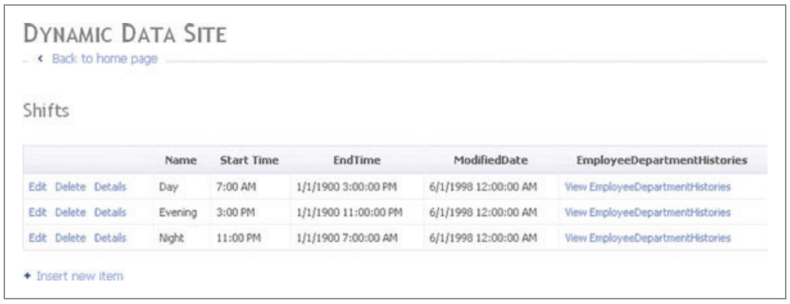
DYNAMIC DATA SITE

‹ Back to home page

My tables

Table Name
Departments
EmployeeAddresses
EmployeeDepartmentHistories
EmployeePayHistories
Employees
JobCandidates
Shifts

Figure 5 **A Basic ASP.NET Dynamic Data Site**



DYNAMIC DATA SITE

‹ Back to home page

Shifts

	Name	Start Time	EndTime	ModifiedDate	EmployeeDepartmentHistories
Edit Delete Details	Day	7:00 AM	1/1/1900 3:00:00 PM	6/1/1998 12:00:00 AM	View EmployeeDepartmentHistories
Edit Delete Details	Evening	3:00 PM	1/1/1900 11:00:00 PM	6/1/1998 12:00:00 AM	View EmployeeDepartmentHistories
Edit Delete Details	Night	11:00 PM	1/1/1900 7:00:00 AM	6/1/1998 12:00:00 AM	View EmployeeDepartmentHistories

+ Insert new item

Figure 6 **Revised Shifts Page Using MetadataType Definitions**

Next click the Shifts link. This displays a list of employee shifts. I'll change the names StartTime and EndTime to Start Time and End Time, respectively.

Another issue is that the Start Time and End Time values show both the date and the time. In this context, the user really only needs to see the time, so I'll format the Start Time and End Time values so that they display times only. An attribute named DataType allows you to specify a more specific data type for a field, such as EmailAddress or Time. You apply the DataType attribute to the field to which it applies.

First, open the Metadata.cs file and add the following class definitions:

```
[MetadataType(typeof(ShiftMetadata))]
public partial class Shift { }
public partial class ShiftMetadata { }
```

Notice that an attribute named MetadataType is applied to the Shift class. This attribute allows you to designate a separate class that contains additional metadata for an entity's fields. You can't apply the additional metadata directly to members of the Shift class because you can't add the same class members to more than one partial class. For example, the Shift class defined in HumanResources.Designer.cs already has a field named StartTime. So you can't add the same field to the Shift class defined in Metadata.cs. Also, you shouldn't manually modify HumanResources.Designer.cs because that file gets regenerated by Visual Studio.

The MetadataType attribute allows you to specify an entirely different class so that you can apply metadata to a field. Add the following code to the ShiftMetadata class:

```
[DataType(DataType.Time)]
[Display(Name = "Start Time")]
public DateTime StartTime { get; set; }
```

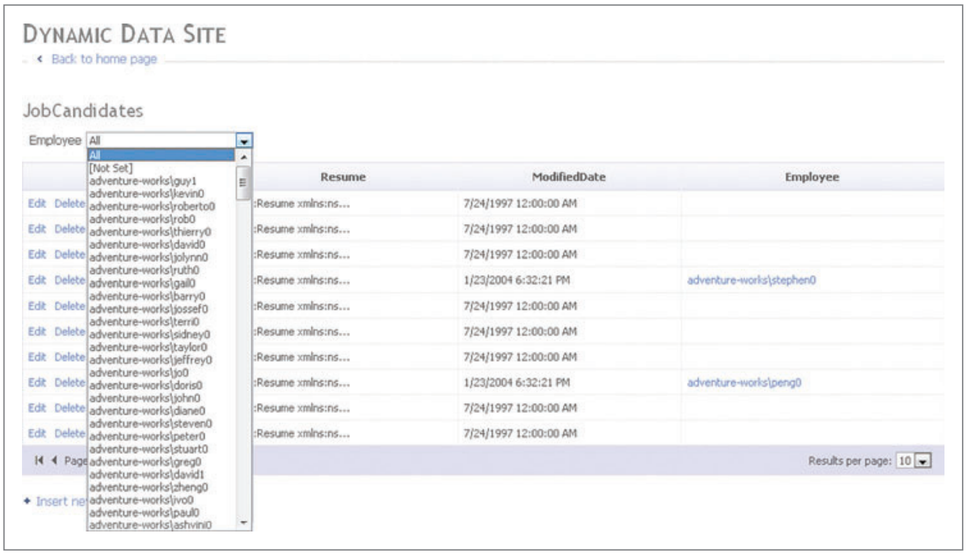
Visual Studio automatically generates entities from the tables that you chose.

The DataType attribute specifies the Time enumerated value to indicate that the StartTime field should be formatted as a time value. Other enumerated values include PhoneNumber, Password, Currency and EmailAddress, to name a few. The Display attribute specifies the text that should be displayed as the field's column when the field is displayed in a list, as well as the field's label when it's displayed in edit or read-only mode.

Now rebuild and run the application. **Figure 6** shows the result of clicking the Shifts link.

You can add similar code to change the appearance of the EndTime field.

Now let's take a look at the JobCandidates link. The Employee column displays values for the NationalIDNumber field. This might not be useful. Although the Employee database table doesn't have a field for an employee's name, it does have a field for an



DYNAMIC DATA SITE

‹ Back to home page

JobCandidates

Employee: All

```
All
[Not Set]
adventure-works\guy1
adventure-works\kevin0
adventure-works\roberto0
adventure-works\rob0
adventure-works\thierry0
adventure-works\david0
adventure-works\jolynn0
adventure-works\ruth0
adventure-works\gail0
adventure-works\barry0
adventure-works\jossef0
adventure-works\terri0
adventure-works\sidney0
adventure-works\taylor0
adventure-works\jeffrey0
adventure-works\jo0
adventure-works\doris0
adventure-works\john0
adventure-works\diane0
adventure-works\steven0
adventure-works\peter0
adventure-works\stuart0
adventure-works\greg0
adventure-works\david1
adventure-works\zheng0
adventure-works\ivo0
adventure-works\paul0
adventure-works\ashvini0
```

	Resume	ModifiedDate	Employee
Edit Delete	:Resume xmlns:ns...	7/24/1997 12:00:00 AM	
Edit Delete	:Resume xmlns:ns...	7/24/1997 12:00:00 AM	
Edit Delete	:Resume xmlns:ns...	7/24/1997 12:00:00 AM	
Edit Delete	:Resume xmlns:ns...	1/23/2004 6:32:21 PM	adventure-works\stephen0
Edit Delete	:Resume xmlns:ns...	7/24/1997 12:00:00 AM	
Edit Delete	:Resume xmlns:ns...	7/24/1997 12:00:00 AM	
Edit Delete	:Resume xmlns:ns...	7/24/1997 12:00:00 AM	
Edit Delete	:Resume xmlns:ns...	1/23/2004 6:32:21 PM	adventure-works\peng0
Edit Delete	:Resume xmlns:ns...	7/24/1997 12:00:00 AM	
Edit Delete	:Resume xmlns:ns...	7/24/1997 12:00:00 AM	

Page

Results per page: 10

+ Insert ne

Figure 7 **Identifying Employees with LoginIDs**

WINDOWS FORMS / WPF / ASP.NET / ACTIVEX

WORD PROCESSING COMPONENTS

MILES BEYOND RICH TEXT

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

TX Text Control Version 16.0 released!

TX TEXTCONTROL®
word processing components

Microsoft® Visual Studio® PARTNER

Word Processing Components for Windows Forms & ASP.NET

WWW.TEXTCONTROL.COM

TX Text Control Sales:
US +1 877-462-4772 (toll-free)
EU +49 421-4270671-0

Attribute Changes in the .NET Framework 4

Starting with the Microsoft .NET Framework 4, we recommend that you use the Display attribute instead of the DisplayName attribute from the .NET Framework 3.5. DisplayName is obviously still in the framework, but we recommend against it whenever the Display attribute can be used.

There are two reasons for preferring Display instead of DisplayName. First, the Display attribute supports localization while the DisplayName attribute does not.

Second, the Display attribute allows you to control all kinds of things. For example, you can control the text for the various ways a field can be displayed (prompt, header and so on), whether a field shows up as a filter, or whether the field is shown in the scaffold (AutoGenerate=false turns it off).

So, while the code shown in the examples in this article is completely valid, we recommend that you replace DisplayName and ScaffoldColumn with the Display attribute. You still need to use ScaffoldTable and the DisplayName attribute at the class level.

Following these recommendations is best because other teams at Microsoft support the DataAnnotations namespace (WCF RIA Services), and following these practices will make sure code works with them.

—Scott Hunter, Senior Program Manager Lead, ASP.NET

Figure 8 **Custom Date_EditField Class**

```
public partial class Date_EditField : System.Web.DynamicData.
FieldTemplateUserControl {
  protected void Page_Load(object sender, EventArgs e) {
    DateCalendar.ToolTip = Column.Description;
  }

  protected override void OnDataBinding(EventArgs e) {
    base.OnDataBinding(e);

    if (Mode == DataBoundControlMode.Edit &&
        FieldValue != null) {
      DateTime date = DateTime.MinValue;
      DateTime.TryParse(FieldValue.ToString(), out date);
      DateCalendar.SelectedDate =
        DateCalendar.VisibleDate = date;
    }
  }

  protected override void ExtractValues(
    IOrderedDictionary dictionary) {
    dictionary[Column.Name] = ConvertEditedValue(
      DateCalendar.SelectedDate.ToShortDateString());
  }

  public override Control DataControl {
    get {
      return DateCalendar;
    }
  }
}
```

employee's login, which is LoginID. This field might provide more useful information to a user of this site.

To do this, I'll again modify the metadata code so that all Employee columns display the value of the LoginID field. Open the Metadata.cs file and add the following class definition:

```
[DisplayColumn("LoginID")]
public partial class Employee { }
```

ASP.NET Routing allows an application to respond to URLs that do not physically exist.

The DisplayColumn attribute specifies the name of the field from an entity that should be used to represent instances of that entity. **Figure 7** shows the new list with LoginIDs.

The ContactID field of the Employee entity actually refers to a row in the Contact table, which is part of the Person schema in the database. Because I didn't add any tables from the Person schema, Visual Studio allows direct editing of the ContactID field. To enforce relational integrity, I'll prevent editing of this field while still allowing it to be displayed for reference. Open the Metadata.cs file and modify the Employee class by applying the following MetadataType attribute:

```
[DisplayColumn("LoginID")]
[MetadataType(typeof(EmployeeMetadata))]
public partial class Employee { }
```

Then define the EmployeeMetadata class as follows:

```
public partial class EmployeeMetadata {
  [Editable(false)]
  public int ContactID { get; set; }
}
```

The Editable attribute specifies whether or not a field is editable in the UI.

Next, I'll add metadata to the EmployeePayHistory entity to display the Rate field as Hourly Rate, as well as format the field's value as currency. Add the following class definitions to the Metadata.cs file:

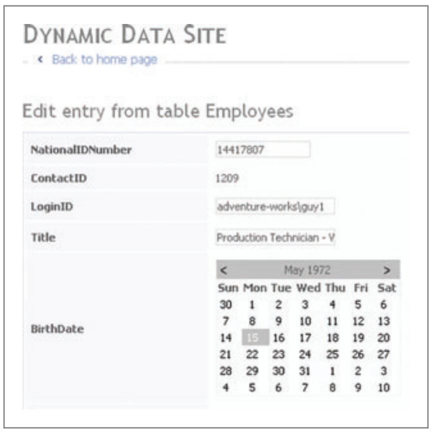
```
[MetadataType(typeof(EmployeePayHistoryMetadata))]
public partial class EmployeePayHistory { }
public partial class EmployeePayHistoryMetadata {
  [DisplayFormat(DataFormatString="{0:c}")]
  [Display(Name = "Hourly Rate")]
  public decimal Rate { get; set; }
}
```

Customizing Templates

The Visual Studio project contains a folder named FieldTemplates. This folder contains user controls for editing fields of various data types. By default, ASP.NET Dynamic Data associates a field with a user control that has the same name as the field's associated data type. For example, the Boolean.ascx user control contains the UI for displaying Boolean fields, while the Boolean_Edit.ascx user control contains the UI for editing Boolean fields.

Alternatively, you can apply the UIHint attribute to a field to ensure that a different user control is used. I'll add a custom field template to display a Calendar control for editing of the Employee entity's BirthDate field.

In Visual Studio, add a new Dynamic Data Field item to the project and name it Date.ascx. Visual Studio automatically adds a second file named Date_Edit.ascx to the



DYNAMIC DATA SITE

‹ Back to home page

Edit entry from table Employees

NationalIDNumber	14417807
ContactID	1209
LoginID	adventure-works\guy1
Title	Production Technician - V
BirthDate	May 1972

```
<        May 1972        >
Sun Mon Tue Wed Thu Fri Sat
30   1   2   3   4   5   6
 7   8   9  10  11  12  13
14  15  16  17  18  19  20
21  22  23  24  25  26  27
28  29  30  31   1   2   3
 4   5   6   7   8   9  10
```

Figure 9 **Customized Employee Edit Form**

```
40, 0); AddEAN13Sup5(page.Elements, 480, 0); AddUPCVersionA(page.Elements, 0, 135); AddUPCVersionASup2(page.Elements, 240, 135); AddUPCVersionASup5(page.Elements, 480, 135); AddEAN8(page.Element
ements, 480, 270); AddUPCVersionE(page.Elements, 0, 405); AddUPCVersionESup2(page.Elements, 240, 405); AddUPCVersionESup5(page.Elements, 480, 405); // Add the page to the document document.Pages.Add
tionAndRectangle(elements, "EAN/JAN 13 Bar Code", x, y, 204, 99); BarCode barCode = new Ean13("123456789012", x, y + 21); barCode.X += (204 - barCode.GetSymbolWidth()) / 2; elements.Add(barCode); } privat
ctangle(elements, "EAN/JAN 13 Bar Code, 2 digit supplement", x, y, 204, 99); BarCode barCode = new Ean13Sup2("12345678901212", x, y + 21); barCode.X += (204 - barCode.GetSymbolWidth()) / 2; elements.Add(
loat x, float y) { AddCaptionAndRectangle(elements, "EAN/JAN 13 Bar Code, 5 digit supplement", x, y, 204, 99); BarCode barCode = new Ean13Sup5("12345678901212345", x, y + 21); barCode.X += (204 - barCode.
PCVersionA(Group elements, float x, float y) { AddCaptionAndRectangle(elements, "UPC Version A Bar Code", x, y, 204, 99); BarCode barCode = new UpcVersionA("12345678901", x, y + 21); barCode.X += (204 - ba
PCVersionASup2(Group elements, float x, float y) { AddCaptionAndRectangle(elements, "UPC Version E Bar Code, 2 digit supplement", x, y, 204, 99); BarCode barCode = new UpcVersionASup2("1234567890112", x,
dd(barCode); } private void AddUPCVersionASup5(Group elements, float x, float y) { AddCaptionAndRectangle(elements, "UPC Version E Bar Code, 5 digit supplement", x, y, 204, 99); BarCode barCode = new UpcVe
GetSymbolWidth()) / 2; elements.Add(barCode); } private void AddEAN8(Group elements, float x, float y) { AddCaptionAndRectangle(elements, "EAN/JAN 8 Bar Code", x, y, 204, 99); BarCode barCode = new Ean8("12
fileDialog.Title = "Open File Dialog";  fileDialog.Filter = "Adobe PDF files (*.pdf)|*.pdf|All Files (*.*)|*.*";  if (fileDialog.ShowDialog() == DialogResult.OK)  { pdfViewer.OpenFile(fileDialog.FileName, "");  }  SaveFileDialog sa
e Dialog";  saveFileDialog.Filter = "Adobe PDF files (*.pdf)|*.pdf|All Files (*.*)|*.*";  if (saveFileDialog.ShowDialog()==DialogResult public ViewerForm().OK)  { pdfViewer.SaveAs(saveFileDialog.FileName);  }  if (pdfViewer.P
Dialog();  }  else  {  MessageBox.Show("Please open a file to print");  }  OpenFileDialog fileDialog = new OpenFileDialog();  file{ Dialog.Title = "Open File Dialog";  fileDialog.InitialDirectory = @"c:\";  fileDialog.Filter = "
 DialogResult.OK)  {  DynamicPDFViewerClass test = new DynamicPDFViewerClass();  PDFPrinter printer = test.OpenFileForPrinter// Convert RTF file to PDF (fileDialog.FileName);  printer.PrintQuiet();  }  byte[] content
rvices;  GCHandle gch = GCHandle.Alloc(contents, GCHandleType.Pinned);  IntPtr contentsIntPtr =gch.AddrOfPinnedObject();pdf Converter.Convert(@"C:\Input.rtf", @"C:\Ouptut.pdf"); Viewer.OpenBuffer(contentsInt
ark Page Element:", x, y); pageElements.Add(new Bookmark("Bookmarked Text", x + 5, y + 20, parentOutline)); pageElements.Add  // Output PDF to Custom Viewer (new Label("This text is bookmarked.", x + 5, y + 2
Elements, float x, float y) { // Adds a circlt to the pageElements AddCaptionAndRectangle(pageElements, "Circle Page Element:", x, pdfViewer.OpenFile(@"C:\Ouptut.pdf", ""); y); pageElements.Add(new Circle(x + 112
arge)); } private void AddFormattedTextArea(Group pageElements, float x, float y) { // Adds a formatted text area to thepageEle}ments string formattedHtml = "<p><i>Dynamic</i><b>PDF</b>&tm; Generator v6.0 for
tting support for text that appears in the document. You have " + "complete control over 8 paragraph properties: spacing before, spacing after, first line " + "indentation, left indentation, right indentation, alignment, all
nt face='Times'>font face, </font><font pointSize='6'>font " + "size, </font><font color='FF0000'>color, </font><b>bold, </b><i>italic and </i><u>underline</u>; " + "and 2 line properties: leading, and leading type.
rea = new FormattedTextArea(formattedHtml, x + 5, y + 20, 215, 60, FontFamily.Helvetica, 9, false); // Sets the indent property formattedTextArea.Style.Paragraph.Indent = 18; AddCaptionAndRectangle(pageElement
Elements, "FormattedTextArea Overflow Text:", x + 279, y); pageElements.Add(formattedTextArea); // Create an overflow formatted text area for the overflow text FormattedTextArea overflowFormattedTextArea = form
 284, y + 20); pageElements.Add(overflowFormattedTextArea); } private void AddImage(Group pageElements, float x, float y) { // Adds an image to the pageElements AddCaptionAndRectangle(pageElements, "Image
DPDFLogo.png"), x + 112.5f, y + 50f, 0.24f); // Image is sized and centered in the rectangle image.SetBounds(215, 60); image.VAlign = VAlign.Center; image.Align = Align.Center; pageElements.Add(image); } private vo
eElements AddCaptionAndRectangle(pageElements, "Label & PageNumberingLabel Page Elements:", x, y); string labelText = "Labels can be rotated"; string numberText = "PageNumberingLabels contain page numbe
x + 5, y + 12, 220, 80, Font.TimesRoman, 12, TextAlign.Center); label.Angle = 8; PageNumberingLabel pageNumLabel = new PageNumberingLabel(numberText, x + 5, y + 55, 220, 80, Font.TimesRoman, 12, TextAlign
ts.Add(label); } private void AddLine(Group pageElements, float x, float y) { // Adds a line to the pageElements AddCaptionAndRectangle(pageElements, "Line Page Element:", x, y); pageElements.Add(new Line(x + 5,
ne(x + 220, y + 20, x + 5, y + 80, 3, RgbColor.Green)); } private void AddLink(Group pageElements, float x, float y) { // Adds a link to the pageElements Font font = Font.TimesRoman; string text = "This is a link to Dyna
:", x, y); Label label = new Label(text, x + 5, y + 20, 215, 80, font, 12, RgbColor.Blue); label.Underline = true; Link link = new Link(x + 5, y + 20, font.GetTextWidth(text, 12), 12 - font.GetDescender(12), new UrlAction("
ments.Add(link); } private void AddPath(Group pageElements, float x, float y) { // Adds a path to the pageElements ceTe.DynamicPDF.PageElements.Path path = new ceTe.DynamicPDF.PageElements.Path(x + 5, y + 2
hs.Add(new LineSubPath(x + 215, y + 40)); path.SubPaths.Add(new CurveToSubPath(x + 108, y + 80, x + 160, y + 80)); path.SubPaths.Add(new CurveSubPath(x + 5, y + 40, x + 65, y + 80, x + 5, y + 60)); AddCaption
(path); } private void AddRectangle(Group pageElements, float x, float y) orderedList = orderedList.GetOverFlowList(x + 5, y + 20); AddCaptionAndRectangle(page.Elements, "Ordered List Page Element Overflow:", x,
/ Create an unordered list UnorderedList unorderedList = new UnorderedList(x + 5, y + 20, 400, 90, Font.Helvetica, 10); unorderedList.Items.Add("Fruits"); unorderedList.Items.Add("Vegetables"); UnorderedSubList un
unorderedSubList.Items.Add(" Citrus"); unorderedSubList.Items.Add(" Non-Citrus"); AddCaptionAndRectangle(page.Elements, "Unordered List Page Element:", x, y, 225, 110); UnorderedSubList unorderedSubList2 =
SubList2.Iten                                                                              s[0].SubLists.AddUnorderedSubList(); subUnorderedSubList.Items.Add("Lime
subUnorderec                                                                               norderedSubList2.Items.Add("Banana"); UnorderedSubList subUnorderedSub
subUnorderec                                                                               UnorderedSubList(); subUnorderedSubList4.Items.Add("String Bean"); subUn
"Kidney Bean                                                                               AndRectangle(page.Elements, "Unordered List Page Element Overflow:", x, y,
AddTextField(                                                                              his is a Scrollable TextField"; txt.BorderColor = RgbColor.Black; txt.Backgroun
t); TextField tx                                                                           gth = 9; txt1.BorderColor = RgbColor.Black; txt1.BackgroundColor = RgbColo
es(); pieSeries.DataLabel = da; plotArea.Series.Add(pieSeries); pieSeries.Elements.Add(27, "Website A"); pieSeries.Elements.Add(19, "Website B"); pieSeries.Elements.Add(21, "Website C"); pieSeries.Elements[0].Colo
lements[2].Co                                                                    ectangle(pageElements, "TextField Form Page Element:", x, y, 504, 85); } private void AddCo
nboBox("cmb                                                                      cb.Font = Font.Helvetica; cb.FontSize = 12; cb.Items.Add("Item 1"); cb.Items.Add("Item 2")
ble"); cb.Item                                                                   ComboBox cb1 = new ComboBox("cmb1name", x + 303, y + 40, 150, 20); cb1.BorderColor
nt.Helvetica;                                                                    "Item 4"); cb1.Items.Add("Non-Editable"); cb1.Items["Non-Editable"].Selected = true; cb1.E
Add(cb1); Cor                                                                    put.pdf");System.Diagnostics.Process.Start("Output.pdf"); AsyncConverter aConverter = ne
Converter_Co                                                                     nverter.Convert(@"C:\temp\DocumentA.rtf", @"C:\temp\OutputA.pdf"); aConverter.Convert(
(@"C:\temp\DocumentC.rtf", @"C:\temp\OutputC.pdf"); aConverter.Convert("http://www.yahoo.com", @"C:\Temp\yahoo.pdf"); ConversionOptions options = new ConversionOptions(720, 720, 72, true); ceTe.Dynamic
p\\output.pdf", options); ceTe.DynamicPDF.Conversion.Converter.Convert("C:\\temp\\Document2.docx", "C:\\temp\\output.pdf", options); string sampleHtml = "<html><body><p>This is a very simple HTML string inc
le border=\"1\"><tr><td>100</td><td>200</td>" + "<td>300</td></tr><tr><td>400</td><td>500</td><td>600</td></tr></table></body></html>";Conversion.Converter.ConvertHtmlString(sampleHtml, "C:\\temp\\
ame", Path.Combine(GetPath(), "LetterPortrait.pdf")); printJob.DocumentName = "Letter Portrait"; if (printJob.Printer.Color) printJob.PrintOptions.Color = true; if (printJob.Printer.Collate) printJob.PrintOptions.Collate =
```

DEVELOPED FOR INTUITIVE USE

DynamicPDF—Comprehensive PDF Solutions for .NET Developers

ceTe Software's DynamicPDF products provide real-time PDF generation, manipulation, conversion, printing, viewing, and much more. Providing the best of both worlds, the object models are extremely flexible but still supply the rich features you need as a developer. Reliable and efficient, the high-performance software is easy to learn and use. If you do encounter a question with any of our components, simply contact ceTe Software's readily available, industry-leading support team.

DynamicPDF®

WWW.DYNAMICPDF.COM

TRY OUR PDF SOLUTIONS FREE TODAY!

www.DynamicPDF.com/eval or call 800.681.5008 | +1 410.772.8620

ceTesoftware

FieldTemplates folder. I'll first replace the content of the Date_Edit.ascx page with the following markup:

```
<asp:Calendar ID="DateCalendar"
runat="server"></asp:Calendar>
```

I'll then modify the Date_Edit.ascx.cs file with the complete class definition shown in **Figure 8**.

I override the OnDataBinding method to set the SelectedDate and VisibleDate properties of the Calendar control to the value of the FieldValue field. The FieldValue field is inherited from FieldTemplateUserControl, and it represents the value of the data field being rendered. I also modify the ExtractValues overridden method to save any changes to the SelectedDate property to a dictionary of field name-value pairs. ASP.NET Dynamic Data uses the values in this dictionary to update the underlying data source.

Next, I need to inform ASP.NET Dynamic Data to use the Date.ascx and Date_Edit.ascx field templates for the BirthDate field. I can accomplish this in one of two ways. First, I can apply the UIHint attribute as follows:

```
[UIHint("Date")]
public DateTime BirthDate { get; set; }
```

Alternatively, I can apply the DateType attribute as follows:

```
[DataType(DataType.Date)]
public DateTime BirthDate { get; set; }
```

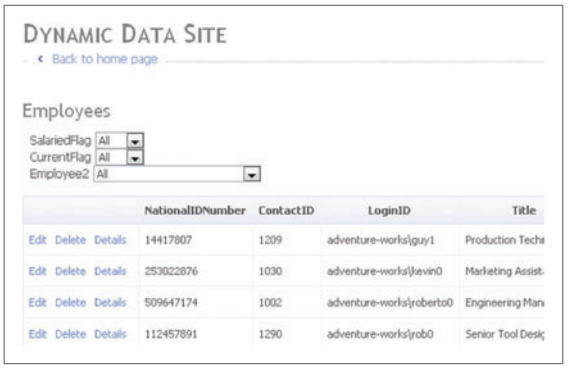
The DataType attribute provides automatic mapping by matching the data type name to the name of the user control. The UIHint attribute gives you greater control in situations where the field type doesn't match the name of the user control. **Figure 9** shows the result of editing an employee.

If you change the birth date of the selected employee and click Update, the new data will be saved to the database.

The PageTemplates folder contains page templates that render the appropriate views for entities. By default, five page templates are supported: List.aspx, Edit.aspx, Details.aspx, Insert.aspx and ListDetails.aspx. The List.aspx page template renders a UI for displaying entities as tabular data. The Details.aspx page template renders a read-only view of an entity, while the Edit.aspx page template displays an editable view of an entity. The Insert.aspx page renders an editable view with default field values. The ListDetails.aspx page template allows you to view a list of entities as well as details for the selected entity on a single page.

As I mentioned earlier in the article, ASP.NET Dynamic Data automatically routes URL requests to the appropriate page by examining the value of the {action} parameter defined for the route. For example, if ASP.NET Dynamic Data evaluates an {action} parameter as List, it uses the List.aspx page template to display a list of entities. You can alter the existing page templates or add a new one to the folder. If you add a new one, you must ensure that you add it to the route table in the Global.asax file.

The EntityTemplates folder contains templates for displaying entity instances in read-only, edit and insert modes. By default, this folder contains three templates named Default.ascx, Default_Edit.ascx and Default_Insert.ascx, which display entity instances in read-only, edit and insert mode, respectively. To create a template that applies to only a specific entity, simply add a new user control to the folder and give it the name of the entity set. For example, if you add a new user control named Shifts.ascx to the folder, ASP.NET Dynamic Data uses this user control to render the read-only mode for a Shift entity (Shifts entity set). Similarly, Shifts_Edit.ascx and Shifts_Insert.ascx render the edit and insert modes of the Shift entity, respectively.


DYNAMIC DATA SITE

< Back to home page

Employees

SalariedFlag All
CurrentFlag All
Employee2 All

	NationalIDNumber	ContactID	LoginID	Title
Edit Delete Details	14417807	1209	adventure-works\guy1	Production Tech
Edit Delete Details	253022876	1030	adventure-works\kevin0	Marketing Assist
Edit Delete Details	509647174	1002	adventure-works\roberto0	Engineering Man
Edit Delete Details	112457891	1290	adventure-works\rob0	Senior Tool Desig

Figure 10 **Including Data Filters on the Page**

For each list of entities, ASP.NET Dynamic Data uses the entity's foreign key fields, Boolean fields and enumeration fields to build a filter list. The filters are added as DropDown controls to the list page, as shown in **Figure 10**.

For Boolean filters, the DropDownList control simply contains three values: All, True and False. For enumeration filters, the DropDownList control contains all enumerated values. For foreign key filters, the DropDownList control contains all distinct foreign key values. The filters are defined as user controls in the Filters folder. By default, only three user controls exist: Boolean.ascx, Enumeration.ascx and ForeignKey.ascx.

You can apply the UIHint attribute to a field to ensure that a different user control is used.

Ship It!

Although this was a fictional scenario, you've seen that it's entirely possible to create a fully operational Human Resources Web site in just a few minutes. I then proceeded to enhance the UI by adding metadata and a custom field template.

ASP.NET Dynamic Data provides out-of-the-box functionality that allows you to get a site up and running quickly. However, it's also entirely customizable, allowing it to meet the needs of individual developers and organizations. ASP.NET Dynamic Data support for ASP.NET Routing allows you to reuse page templates for CRUD operations. If you've become frustrated with having to continually perform the tedious tasks of implementing CRUD pages on every Web application project, then ASP.NET Dynamic Data should make your life a lot easier. ■

JAMES HENRY *is an independent software developer for BlueVision LLC, a company that specializes in Microsoft technology consulting. He's the author of "Developing Business Intelligence Solutions Using Information Bridge and Visual Studio .NET" (Blue Vision, 2005) and "Developing .NET Custom Controls and Designers Using C#" (Blue Vision, 2003). He can be reached at msdnmag@bluevisionsoftware.com.*

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

Does your Team do more than just track bugs?

Free Trial and Single User FreePack™ available at www.alexcorp.com

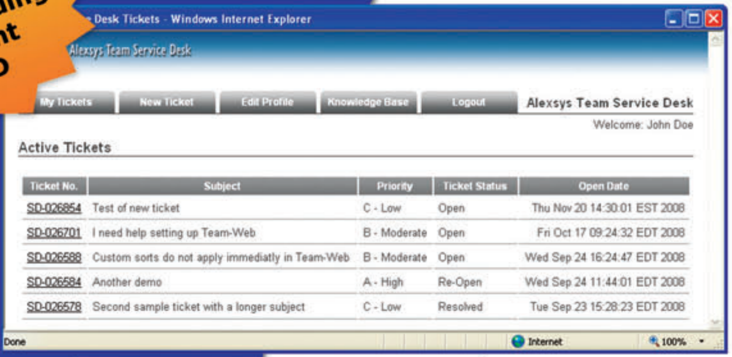
Alexsys Team® does! Alexsys Team 2 is a multi-user Team management system that provides a powerful yet easy way to manage all the members of your team and their tasks - including defect tracking. Use Team right out of the box or tailor it to your needs.

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

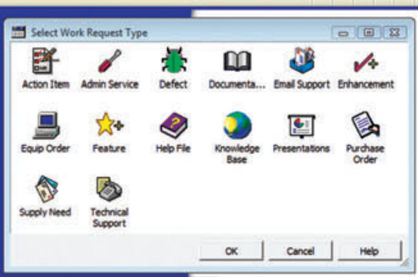

Desk Tickets - Windows Internet Explorer
Alexsys Team Service Desk
My Tickets
New Ticket
Edit Profile
Knowledge Base
Logout
Alexsys Team Service Desk
Welcome: John Doe
Done
Internet
100%

Active Tickets

Ticket No.	Subject	Priority	Ticket Status	Open Date
SD-026854	Test of new ticket	C - Low	Open	Thu Nov 20 14:30:01 EST 2008
SD-026701	I need help setting up Team-Web	B - Moderate	Open	Fri Oct 17 09:24:32 EDT 2008
SD-026588	Custom sorts do not apply immediatly in Team-Web	B - Moderate	Open	Wed Sep 24 16:24:47 EDT 2008
SD-026584	Another demo	A - High	Re-Open	Wed Sep 24 11:44:01 EDT 2008
SD-026578	Second sample ticket with a longer subject	C - Low	Resolved	Tue Sep 23 15:28:23 EDT 2008

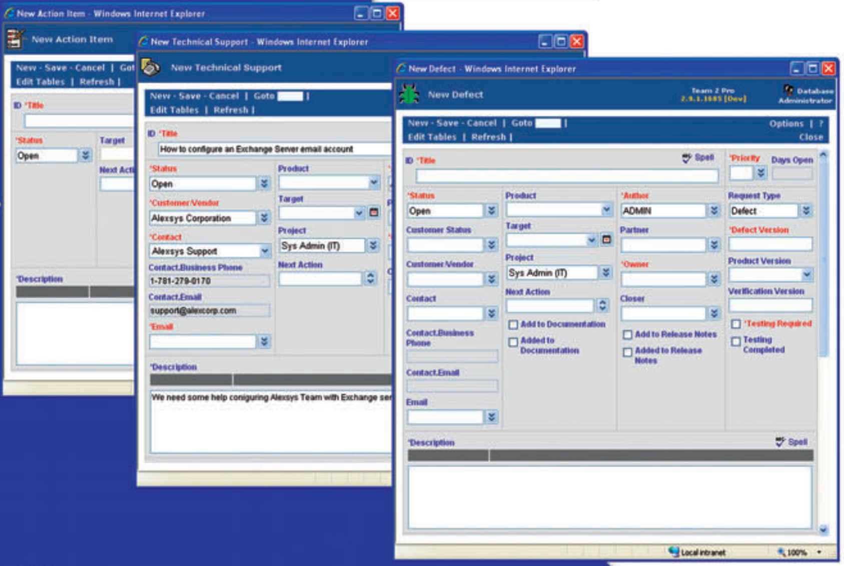
New in Team 2.11

- Full Windows 7 Support
- Windows Single Sign-on
- System Audit Log
- Trend Analysis
- Alternate Display Fields for Data Normalization
- Lookup Table Filters
- XML Export
- Network Optimized for Enterprise Deployment



Service Desk Features

- Fully Secure
- Unlimited Users Self Registered or Active Directory
- Integrated into Your Web Site
- Fast/AJAX Dynamic Content
- Unlimited Service Desks
- Visual Service Desk Builder



Team 2 Features

- Windows and Web Clients
- Multiple Work Request Forms
- Customizable Database
- Point and Click Workflows
- Role Based Security
- Clear Text Database
- Project Trees
- Time Recording
- Notifications and Escalations
- Outlook Integration

ZDNet Editors' Pick

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.

Multiparadigmatic .NET, Part 5: Automatic Metaprogramming

In last month's piece, objects came under the microscope, and in particular we looked at the "axis" of commonality/variability analysis that inheritance offers us. While inheritance isn't the only form of commonality/variability available within a modern object-oriented (OO) language such as C# or Visual Basic, it certainly stands at the center of the OO paradigm. And, as also discussed, it doesn't always provide the best solution to all problems.

To recap, what we've discovered so far is that C# and Visual Basic are both procedural and OO languages—but clearly the story doesn't stop there. Both are also *metaprogrammatic* languages—each offers the Microsoft .NET Framework developer the opportunity to build programs out of programs, in a variety of different ways: automatic, reflective and generative.

Automatic Metaprogramming

The core idea behind metaprogramming is simple: the traditional constructs of procedural or OO programming haven't solved quite all of our software design issues, at least not in a way that we find satisfying. For example, to cite a basic flaw, developers frequently found a need for a data structure that maintained an ordered list of some particular type, such that we could insert items into the list in a particular slot and see the items in that exact order. For performance reasons, sometimes the list wanted to be in a linked list of nodes. In other words, we wanted an ordered linked list, but strongly typed to the type being stored within it.

Developers who came to the .NET Framework from the C++ world know one solution to this problem—that of parameterized types, also known more informally as generics. But, as developers who came to .NET through Java's early days know, another solution emerged long before templates (which did, eventually, make it into the Java platform). That solution was to simply write each needed list implementation as necessary, as shown in **Figure 1**.

Now, obviously, this fails the Don't Repeat Yourself (DRY) test—every time the design calls for a new list of this kind, it will need to be written "by hand," which will clearly be a problem as time progresses. Although not complicated, it's still going to be awkward and time-consuming to write each of these, particularly if more features become necessary or desirable.

Of course, nobody ever said developers had to be the ones writing such code. Which brings us around neatly to the solution of code generation, or as it's sometimes called, *automatic metaprogramming*. Another program can easily do it, such as a program designed to kick out classes that are customized to each type needed, as shown in **Figure 2**.

Then, once the class in question has been created, it needs only to be compiled and either added to the project or else compiled into its own assembly for reuse as a binary.

Of course, the language being generated doesn't have to be the language in which the code generator is written—in fact, it will often help immensely if it isn't, because then it will be easier to keep the two more clearly distinct in the developer's head during debugging.

Commonality, Variability and Pros and Cons

In the commonality/variability analysis, automatic metaprogramming occupies an interesting place. In the **Figure 2** example, it places structure and behavior (the outline of the class above) into commonality, allowing for variability along data/type lines, that of the type being stored in the generated class. Clearly, we can swap in any type desired into the ListOf type.

Figure 1 **An Example of Writing List Implementations as Necessary**

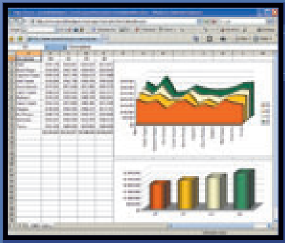
```
Class ListOfInt32

  Class Node
    Public Sub New(ByVal dt As Int32)
      data = dt
    End Sub
    Public data As Int32
    Public nextNode As Node = Nothing
  End Class

  Private head As Node = Nothing

  Public Sub Insert(ByVal newParam As Int32)
    If IsNothing(head) Then
      head = New Node(newParam)
    Else
      Dim current As Node = head
      While (Not IsNothing(current.nextNode))
        current = current.nextNode
      End While
        current.nextNode = New Node(newParam)
    End If
  End Sub

  Public Function Retrieve(ByVal index As Int32)
    Dim current As Node = head
    Dim counter = 0
    While (Not IsNothing(current.nextNode) And counter < index)
      current = current.nextNode
      counter = counter + 1
    End While

    If (IsNothing(current)) Then
      Throw New Exception("Bad index")
    Else
      Retrieve = current.data
    End If
  End Function
End Class
```

Microsoft Chose SpreadsheetGear...

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

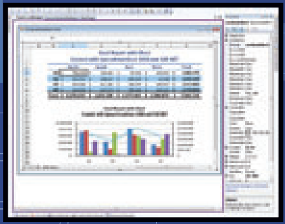
Chris Donohue, MSN Money Program Manager

SpreadsheetGear 2010



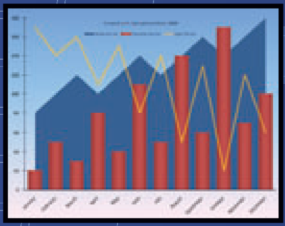
ASP.NET Excel Reporting

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



Excel Compatible Windows Forms Control

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



Create Dashboards from Excel Charts and Ranges

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

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

www.SpreadsheetGear.com.

SpreadsheetGear

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

But automatic metaprogramming can reverse that, too, if necessary. Using a rich templating language, such as the Text Template Transformation Toolkit (T4) that ships with Visual Studio, the code generation templates can do source-time decision making, which then allows the template to provide commonality along data/structural lines, and vary by structural and behavioral lines. In fact, if the code template is sufficiently complex (and this isn't necessarily a good angle to pursue), it's even possible to eliminate commonality altogether and vary everything (data, structure, behavior and so on). Doing so typically becomes unmanageable quite quickly, however, and in general should be avoided. That leads to one of the key realizations about automatic metaprogramming: Because it lacks any sort of inherent structural restrictions, choose the commonality and variability explicitly, lest the source code template grow out of control trying to be too flexible. For example, given the ListOf example in **Figure 2**, the commonality is in the structure and behavior, and the variability is in the data type being stored—any attempt to introduce variability in the structure or behavior should be considered to be a red flag and a potential slippery slope to chaos.

Obviously, code generation carries with it some significant risks, particularly around areas of maintenance: Should a bug be discovered (such as the concurrency one in the ListOf… example in **Figure 2**), fixing it isn't a simple matter. The template can obviously be fixed, but that doesn't do anything for the code already generated—each of those source artifacts needs to be regenerated, in turn, and this is something that's hard to track and ensure automatically. And, what's more, any handmade changes to those generated files will be intrinsically lost, unless the template-generated code has allowed for customizations. This risk of overwrite can be mitigated by use of partial classes, allowing developers to fill in the "other half" (or not) of the class being generated, and by extension methods, giving developers an opportunity to "add" methods to an existing family of types without having to edit the types. But partial classes must be in place from the beginning within the templates, and extension methods carry some restrictions that prevent them from replacing existing behavior—again leaving neither approach as a good mechanism to carry negative variability.

Figure 2 **An Example of Automatic Metaprogramming**

```
Sub Main(ByVal args As String())
  Dim CRLF As String = Chr(13).ToString + Chr(10).ToString()
  Dim template As String =
   "Class ListOf{0}" + CRLF +
   "  Class Node" + CRLF +
   "    Public Sub New(ByVal dt As {0})" + CRLF +
   "      data = dt" + CRLF +
   "    End Sub" + CRLF +
   "    Public data As {0}" + CRLF +
   "    Public nextNode As Node = Nothing" + CRLF +
   "  End Class" + CRLF +
   "  Private head As Node = Nothing" + CRLF +
   "  Public Sub Insert(ByVal newParam As {0})" + CRLF +
   "    If IsNothing(head) Then" + CRLF +
   "      head = New Node(newParam)" + CRLF +
   "    Else" + CRLF +
   "      Dim current As Node = head" + CRLF +
   "      While (Not IsNothing(current.nextNode))" + CRLF +
   "        current = current.nextNode" + CRLF +
   "      End While" + CRLF +
   "      current.nextNode = New Node(newParam)" + CRLF +
   "    End If" + CRLF +
   "  End Sub" + CRLF +
   "  Public Function Retrieve(ByVal index As Int32)" + CRLF +
   "    Dim current As Node = head" + CRLF +
   "    Dim counter = 0" + CRLF +
   "    While (Not IsNothing(current.nextNode) And counter < index)"+ CRLF +
   "      current = current.nextNode" + CRLF +
   "      counter = counter + 1" + CRLF +
   "    End While" + CRLF +
   "    If (IsNothing(current)) Then" + CRLF +
   "      Throw New Exception()" + CRLF +
   "    Else" + CRLF +
   "      Retrieve = current.data" + CRLF +
   "    End If" + CRLF +
   "  End Sub" + CRLF +
   "End Class"

    If args.Length = 0 Then
      Console.WriteLine("Usage: VBAuto <listType>")
      Console.WriteLine("   where <listType> is a fully-qualified CLR typename")
    Else
      Console.WriteLine("Producing ListOf" + args(0))

      Dim outType As System.Type =
        System.Reflection.Assembly.Load("mscorlib").GetType(args(0))
      Using out As New StreamWriter(New FileStream("ListOf" + outType.Name + ".vb",
                                        FileMode.Create))
        out.WriteLine(template, outType.Name)
      End Using
    End If
```

Code Generation

Code generation—or automatic metaprogramming—is a technique that's been a part of programming for many years, ranging all the way from the C preprocessor macro through to the C# T4 engine, and will likely continue to carry forward, owing to the conceptual simplicity of the idea. However, its principal flaws are the lack of compiler structure and checking during the expansion process (unless, of course, that checking is done by the code generator itself, a task that's harder than it sounds) and the inability to capture negative variability in any meaningful way. The .NET Framework offers some mechanisms to make code generation easier—in many cases, those mechanisms were introduced to save other Microsoft developers some grief—but they won't eliminate all the potential pitfalls in code generation, not by a long shot.

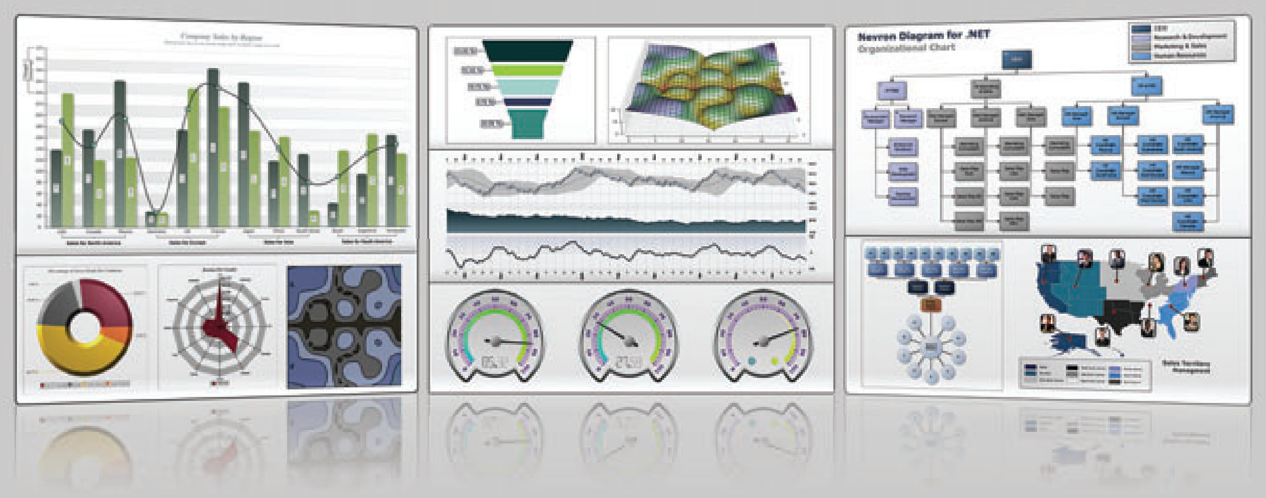
And yet, automatic metaprogramming remains one of the more widely used forms of metaprogramming. C# has a macro preprocessor, as do C++ and C. (Using macros to create "tiny templates" was common before C++ got templates.) On top of that, using metaprogramming as part of a larger framework or library is common, particularly for inter-process communication scenarios (such as the client and server stubs generated by Windows Communication Foundation). Other toolkits use automatic metaprogramming to provide "scaffolding" to ease the early stages of an application (such as what we see in ASP.NET MVC). In fact, arguably every Visual Studio project begins with automatic metaprogramming, in the form of the "project templates" and "item templates" that most of us use to create new projects or add files to projects. And so on. Like so many other things in computer science, automatic metaprogramming remains a useful and handy tool to have in the designer toolbox, despite its obvious flaws and pitfalls. Fortunately, it's far from the only meta-tool in the programmer's toolbox. ■

TED NEWARD *is a principal with Neward & Associates, an independent firm specializing in enterprise .NET Framework and Java platform systems. He's written more than 100 articles, is a C# MVP and INETA speaker, and has authored and coauthored a dozen books, including "Professional F# 2.0" (Wrox, 2010). He also consults and mentors regularly. Reach him at ted@tedneward.com and read his blog at blogs.tedneward.com.*

visualize your success

Let us help you visualize your success

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



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

Developers

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

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

IT Professionals

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

- Chart for SSRS
- Gauge for SSRS

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

- Chart for SharePoint
- Gauge for SharePoint

MAKE SURE THAT YOUR DATA IS MAKING THE VISUAL STATEMENT IT DESERVES BY DOWNLOADING YOUR FREE EVALUATION COPY FROM WWW.NEVRON.COM TODAY.

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

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

CHARLES PETZOLD

A Color Scroll for XNA

One of the first Windows programs I wrote for publication was called COLORSCR ("color scroll"), and it appeared in the May 1987 issue of *Microsoft Systems Journal*, the predecessor to this magazine.

Over the years, I've frequently found it instructive to rewrite this program for other APIs and frameworks. Although the program is simple—you manipulate three scrollbars or sliders for red, green and blue values to create a custom color—it involves crucial tasks such as layout and event handling. In functionality, as well, the program is not strictly a simple, meaningless demo. If you ever found yourself creating a color-selection dialog, this program would be a good place to start.

In this article, I want to show you how to write a color-scroll program using the XNA Framework as implemented in Windows Phone 7. The result is shown in **Figure 1**.

XNA is Microsoft .NET Framework-based managed code for games programming, but if you start searching through the Microsoft.Xna.Framework namespaces, you won't find any scrollbars or sliders or even buttons among the classes! XNA is simply not that kind of framework. If you want a slider in your program, it must be specially created for that job.

Showing you how to write an XNA color-scroll program isn't just an academic exercise, however. My intention is actually much broader.

As you know, Windows Phone 7 supports both Silverlight and XNA, and generally for any particular application the choice between the two is obvious. However, you might find that the graphical demands of your application are simply too much for Silverlight, but at the same time you might be hesitant about using XNA instead because it's missing familiar amenities, such as buttons and sliders.

My response is: Do not be afraid. Writing simple controls in XNA is easier than you probably think.

You can also consider this article as an introduction to XNA programming for Windows Presentation Foundation (WPF) and Silverlight programmers. I found myself applying certain concepts from WPF and Silverlight to this job, so the XNA color-scroll program is a celebration of synergy as well.

All the downloadable source code is in one Visual Studio solution called XnaColorScroll. This solution includes an XnaColorScroll project for Windows Phone 7 and a library called Petzold.Xna.Controls containing the four XNA "controls" that I created for this program. (Obviously, loading the solution into Visual Studio requires that you have the Windows Phone Development Tools installed.) Feel free to steal these controls and supplement them with your own for your own XNA controls library.

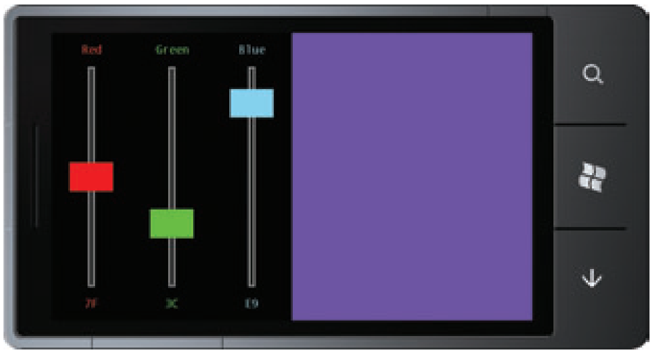


Figure 1 **The Xnacolorscroll Program**

Games and Components

Just as the main class of a WPF, Silverlight or Windows Forms program is a class that derives from Window, Page or Form, the main class of an XNA program derives from Game. An XNA program uses this Game derivative to draw bitmaps, text and 3D graphics on the screen.

To help you organize your program into discrete functional entities, XNA supports the concept of a *component*, which is perhaps the closest that XNA comes to a control. Components come in two varieties: GameComponent derivatives and DrawableGameComponent derivatives. DrawableGameComponent itself derives from GameComponent, and the names suggest the difference. I'll be using both base classes in this exercise.

The Game class defines a property named Components of type GameComponentCollection. Any components that a game creates must be added to this collection. Doing so ensures that the component gets the same calls to various overridden methods that the game itself gets. A DrawableGameComponent draws on top of whatever the Game draws.

Let's begin the analysis of the XnaColorScroll program—not with the Game derivative, but with the GameComponent derivatives. **Figure 2** shows the source code for a component I call TextBlock, and it's used in much the same way as a WPF or Silverlight TextBlock.

Notice the public properties following the constructor in **Figure 2**. These indicate the text itself, and the font, color and position of the text relative to the screen. Two additional properties, HorizontalAlignment and VerticalAlignment, allow that position to reference

Code download available at code.msdn.microsoft.com/mag201101UIFrontiers.

the side or center of the text string. In XnaColorScroll, these properties are handy for centering the text relative to the sliders.

The class also includes a get-only Size property, which returns the size of the rendered text with that particular font. (The XNA Vector2 structure is often used for representing two-dimensional positions and sizes as well as vectors.)

The TextBlock component overrides both the Update and Draw methods, which is normal for a drawable component and normal for Game derivatives as well. These two methods constitute the program's "game loop," and they're called at the same rate as the video display—30 times per second for Windows Phone 7. You can think of the Draw method like a WM_PAINT message or an OnPaint override, except that it's called for every screen refresh. The Update method is called before each Draw call and is intended to be used for performing all the necessary calculations and preparation for the Draw method.

TextBlock as written has lots of room for performance improvements. For example, it could cache the value of the Size property or only recalculate the textOrigin field whenever any property that contributes to it changes. These enhancements might take priority if you ever discover that the program has become sluggish because all the Update and Draw calls in the program are taking longer than 0.03 seconds to execute.

If you want a slider in your program, it must be specially created for that job.

Components and Child Components

As you can see in **Figure 1**, XnaColorScroll has six TextBlock components but also three sliders, which are probably going to be a little more challenging.

Very many years ago, I might have tried coding the entire slider in a single class. With my experience in WPF and Silverlight control templates, however, I realize that the Slider control might best be constructed by assembling a Thumb control and RepeatButton controls. It seems reasonable to try to create an XNA Slider component from similar child components.

A Game class adds child components to itself by adding them to its Components property of type GameComponentCollection. The big question now becomes: Does GameComponent itself have a Components property for its own child components?

Unfortunately, the answer is no. However, a component has access to its parent Game class, and if the component needs to instantiate its own child components, it can add those additional components to the same Components collection of the Game class. Do this in the right order and you won't even need to mess around with the DrawOrder property—defined by DrawableGameComponent—that can help with the order in which components are rendered. By default, the Draw method in the Game class is called first, then all the Draw methods in the components as they appear in the Components collection.

So let's begin. Like TextBlock, the Thumb class derives from DrawableGameComponent. The entire Thumb class except for the touch processing is shown in **Figure 3**. Thumb defines the same three events as the WPF and Silverlight Thumb controls.

Figure 2 **The TextBlock Game Component**

```
public class TextBlock : DrawableGameComponent
{
  SpriteBatch spriteBatch;
  Vector2 textOrigin;

  public TextBlock(Game game) : base (game)
  {
    // Initialize properties
    this.Color = Color.Black;
  }

  public string Text { set; get; }
  public SpriteFont Font { set; get; }
  public Color Color { set; get; }
  public Vector2 Position { set; get; }
  public HorizontalAlignment HorizontalAlignment { set; get; }
  public VerticalAlignment VerticalAlignment { set; get; }

  public Vector2 Size
  {
    get
    {
      if (String.IsNullOrEmpty(this.Text) || this.Font == null)
          return Vector2.Zero;

      return this.Font.MeasureString(this.Text);
    }
  }

  protected override void LoadContent()
  {
    spriteBatch = new SpriteBatch(this.GraphicsDevice);
    base.LoadContent();
  }

  public override void Update(GameTime gameTime)
  {
    if (!String.IsNullOrEmpty(Text) && Font != null)
    {
      Vector2 textSize = this.Size;
      float xOrigin = (int)this.HorizontalAlignment * textSize.X / 2;
      float yOrigin = (int)this.VerticalAlignment * textSize.Y / 2;
      textOrigin = new Vector2((int)xOrigin, (int)yOrigin);
    }
    base.Update(gameTime);
  }

  public override void Draw(GameTime gameTime)
  {
    if (!String.IsNullOrEmpty(Text) && Font != null)
    {
      spriteBatch.Begin();
      spriteBatch.DrawString(this.Font, this.Text, this.Position, this.Color,
                             0, textOrigin, 1, SpriteEffects.None, 0);
      spriteBatch.End();
    }
    base.Draw(gameTime);
  }
}
```

In the TextBlock component in **Figure 2**, the Position property is a point of type Vector2; in Thumb, the Position property is a Rectangle object, and defines not only the location of the Thumb, but also its rectangular size. The Thumb visuals consist entirely of a white 1 x 1 pixel bitmap that's displayed at the location and size indicated by the Position property. (Using single-pixel bitmaps stretched to a particular size is one of my favorite XNA programming techniques.)

The Thumb processes touch input but doesn't move itself. The Slider component is responsible for handling DragDelta events from the Thumb and setting a new Position property.

Besides the Thumb, my Slider also contains two RepeatButton components, one on each side of the Thumb. Like the WPF and Silverlight RepeatButton controls, these components generate a series of Click events if you hold your finger on them.

In my XNA Slider, the RepeatButton components occupy a particular area of the screen but are actually invisible. (The thin vertical track down the center is drawn by Slider itself.) This means that RepeatButton can derive from GameComponent rather than DrawableGameComponent. **Figure 4** shows the whole RepeatButton control except for the touch processing.

At this point, we're ready to build the Slider component. To keep it reasonably simple, I restricted myself to a vertical orientation. Slider has the usual public properties—Minimum, Maximum and Value, among others—and also defines a ValueChanged event. In its Initialize override, Slider creates the three child components (and saves them as fields), sets the event handlers and adds them to the Components collection of its parent Game class accessible through its Game property:

```
public override void Initialize()
{
  thumb = new Thumb(this.Game);
  thumb.DragDelta += OnThumbDragDelta;
  this.Game.Components.Add(thumb);

  btnDecrease = new RepeatButton(this.Game);
  btnDecrease.Click += OnRepeatButtonClick;
  this.Game.Components.Add(btnDecrease);

  btnIncrease = new RepeatButton(this.Game);
  btnIncrease.Click += OnRepeatButtonClick;
  this.Game.Components.Add(btnIncrease);

  base.Initialize();
}
```

XNA Game derivatives and GameComponent derivatives have three occasions to perform initialization—the class constructor, an override of the Initialize method and an override of the LoadContent method—and it might be a little confusing when to use which. As the name suggests, LoadContent is great for loading content (such as fonts or bitmaps) and should be used for all other initialization that depends on the existence of the GraphicsDevice object.

I prefer creating components and adding them to the Components collection during the Initialize override, as shown in the previous code snippet. In XnaColorScroll, the Game derivative class also creates six TextBlock components and three Slider components in its Initialize override. When the Initialize method in the base class is called at the end, then all the Initialize methods for the child components are called. The Initialize method in each Slider then creates the Thumb and two RepeatButton components. This scheme ensures that all these components are added to the Components collection in a proper order for rendering.

Processing Touch Input

The primary means of user input in most Windows Phone 7 programs is multi-touch, and XNA is no exception. (Keyboard input is also available to XNA programs, and programs can also use the phone's accelerometer as an input device.)

An XNA program obtains touch input during the Update override. There are no touch events. All touch input is polled and is accessible through the TouchPanel class. The static TouchPanel.GetState method provides low-level touch input that consists of TouchLocation objects indicating Pressed, Moved and Released activity with position information and integer ID numbers to distinguish between multiple fingers. The TouchPanel.ReadGesture method (which I don't use in this program) provides higher-level gesture support.

When I first started working with game components, I thought that each component could obtain touch input on its own, take what it needed and ignore the rest. This didn't seem to work very well; it was as if the game class and the components were all competing for touch input rather than sharing it like polite children.

Writing simple controls in XNA is easier than you probably think.

I decided to work out another system for processing touch input. Only the Game class calls TouchPanel.GetState. Then, each TouchLocation object is passed to child components through a method I call ProcessTouch. This allows the components to get first dibs on the input. A component that makes use of a TouchLocation object returns true from the ProcessTouch method, and further processing for that particular TouchLocation object ends. (Returning true from ProcessTouch is like setting the Handled property of RoutedEventArgs to true in WPF or Silverlight.)

Figure 3 **Most of the Thumb Class**

```
public class Thumb : DrawableGameComponent
{
  SpriteBatch spriteBatch;
  Texture2D tinyTexture;
  int? touchId;

  public event EventHandler<DragEventArgs> DragStarted;
  public event EventHandler<DragEventArgs> DragDelta;
  public event EventHandler<DragEventArgs> DragCompleted;

  public Thumb(Game game) : base(game)
  {
    // Initialize properties
    this.Color = Color.White;
  }

  public Rectangle Position { set; get; }
  public Color Color { set; get; }

  protected override void LoadContent()
  {
    spriteBatch = new SpriteBatch(this.GraphicsDevice);
    tinyTexture = new Texture2D(this.GraphicsDevice, 1, 1);
    tinyTexture.SetData<Color>(new Color[] { Color.White });

    base.LoadContent();
  }

  ...

  public override void Draw(GameTime gameTime)
  {
    spriteBatch.Begin();
    spriteBatch.Draw(tinyTexture, this.Position, this.Color);
    spriteBatch.End();

    base.Draw(gameTime);
  }
}
```

YOU ARE NOT A CODER: YOU ARE A CREATOR.

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

YOU TAKE AN IDEA AND MAKE IT A REALITY. You turn the "impossible" into "possible." You bring into being the applications that your company relies on everyday.

Attend Visual Studio Live! for real-world education for creators like you. Full-day intensive workshops. Practical how-to sessions. Insightful keynotes from Microsoft MVPs. From Silverlight to Sharepoint, Data Management to Visual Studio 2010, you'll get the training that will take your code to the next level.

APRIL 18–22, 2011

LAS VEGAS, NEVADA

RIO ALL-SUITE HOTEL & CASINO

REGISTER TODAY
SAVE $300!
USE CODE ADJAN

WWW.VSLIVE.COM/LV

SUPPORTED BY: Microsoft

PRODUCED BY: 1105 MEDIA

In XnaColorScroll, the Update method in the main Game class calls TouchPanel.GetState to get all the touch input and then calls the ProcessTouch method in each Slider component as shown here:

```
TouchCollection touches = TouchPanel.GetState();

foreach (TouchLocation touch in touches)
{
  for (int primary = 0; primary < 3; primary++)
    if (sliders[primary].ProcessTouch(gameTime, touch))
      break;
}
```

Notice how further processing of each TouchLocation object stops whenever ProcessTouch returns true.

The ProcessTouch method in each Slider control then calls the ProcessTouch methods in the two RepeatButton components and the Thumb component:

```
public bool ProcessTouch(GameTime gameTime, TouchLocation touch)
{
  if (btnIncrease.ProcessTouch(gameTime, touch))
    return true;

  if (btnDecrease.ProcessTouch(gameTime, touch))
    return true;

  if (thumb.ProcessTouch(gameTime, touch))
    return true;

  return false;
}
```

If any of these components (or the Game class itself) wished to perform its own touch processing, it would first call ProcessTouch in each child. Any TouchLocation object that remained unprocessed by the children could then be examined for the class's own use. In effect, this scheme allows the visually topmost children to have first access to touch input, which is often exactly what you want. It's much like the routed event handling implemented in WPF and Silverlight.

The primary means of user input in most Windows Phone 7 programs is multi-touch, and XNA is no exception.

Both the RepeatButton and the Thumb are interested in touch input if a finger is first touched within the area indicated by the component's Position rectangle. The component then "captures" that finger by saving the touch ID. All other touch input with that ID belongs to that particular component until the finger is released.

Propagating Events

The touch processing of the Thumb and RepeatButton components is really the crux of the Slider control, but that's something you can study on your own if you're interested. When the Thumb component detects finger movement on its surface, it generates DragDelta events; when the RepeatButton component detects taps or sustained presses, it fires Click events.

Both these events are handled by the parent Slider component, which adjusts its Value property accordingly, and this fires a ValueChanged event from the Slider. The Slider is also responsible for setting the Position properties of the Thumb and two RepeatButton components based on the new Value.

Figure 4 Most of the RepeatButton Class

```
public class RepeatButton : GameComponent
{
  static readonly TimeSpan REPEAT_DELAY = TimeSpan.FromMilliseconds(500);
  static readonly TimeSpan REPEAT_TIME = TimeSpan.FromMilliseconds(100);

  int? touchId;
  bool delayExceeded;
  TimeSpan clickTime;

  public event EventHandler Click;

  public RepeatButton(Game game) : base(game)
  {
  }

  public Rectangle Position { set; get; }

  ...

}
```

The Game class handles the ValueChanged events from the three Slider components in a single event handler, so the method simply sets new values of the three TextBlock components and a new color:

```
void OnSliderValueChanged(object sender, EventArgs args)
{
  txtblkValues[0].Text = sliders[0].Value.ToString("X2");
  txtblkValues[1].Text = sliders[1].Value.ToString("X2");
  txtblkValues[2].Text = sliders[2].Value.ToString("X2");

  selectedColor = new Color(sliders[0].Value,
                            sliders[1].Value,
                            sliders[2].Value);
}
```

That leaves the Draw override in the Game class with just two jobs: Draw the black rectangle that serves as background behind the components, and draw the rectangle with a new selectedColor on the right. Both jobs involve a 1 x 1 pixel bitmap stretched to fill half the screen.

Better than Silverlight?

I recently also wrote a Silverlight version of the color-scroll program for Windows Phone 7, and I discovered that the program only allowed me to manipulate one Slider at a time. However, if you deploy the XnaColorScroll program to a phone, you can manipulate all three Slider controls independently. I find that difference very interesting. It illustrates once again how high-level interfaces such as Silverlight may make our lives much simpler, but often something else needs to be sacrificed. I believe the concept is called TANSTAAFL: There ain't no such thing as a free lunch.

XNA is so low-level in some respects that it might remind us of the Wild West. But with a little judicious use of components (including components built from other components) and mimicking routed event handling, even XNA can be tamed and made to seem more like Silverlight—perhaps even better. ■

CHARLES PETZOLD *is a longtime contributing editor to* MSDN Magazine. *His new book, "Programming Windows Phone 7" (Microsoft Press, 2010), is available as a free download at bit.ly/cpebookpdf.*

THANKS to the following technical expert for reviewing this article: *Shawn Hargreaves*

Know it all.

We've got all the answers.

- Thousands of code samples
- Over 30,000 articles
- Practical answers in active forums
- MFC/Microsoft® Visual C++®, Visual C#®, Visual Basic® .NET, ASP.NET, SQL Server®, Windows® Phone, and more!
- QuickAnswers
- 7.5 million members
- Start your FREE membership TODAY!

THE CODE PROJECT™
WWW.CODEPROJECT.COM

Turn! Turn! Turn!

I was listening to Pandora.com the other day, and its algorithms decided to play me the Byrds classic (1965) folk-rock song "Turn! Turn! Turn!" I know you'll recognize some of its lyrics: "To everything, there is a season, and a time for every purpose under heaven. A time to be born, a time to die; a time to plant, a time to reap; a time to kill, a time to heal; a time to laugh, a time to weep."

The words, of course, come from the biblical book of Ecclesiastes, specifically the third chapter. Their authorship is popularly attributed to Shlomo ben David, Solomon the Wise (although see the debate at bit.ly/g8Q5I6). If true, an ancient Israelite king would own credit for the lyrics to a Billboard No. 1 hit (though I suspect the copyright has expired, so he wouldn't get much money for it). Pete Seeger put the words to music in 1959, rearranging some to make the song more singable ("poetic license," another triumph of usability over strict veracity) and adding six words of his own.

Geek that I am, I couldn't help but realize the pattern that unites the verses of this song. They all describe opposite actions and state that a time exists for each. To put it into canonical geek language, there is a time for A and a time for !A (or ~A, if you're going to get picky).

To put it into canonical geek language, there is a time for A and a time for !A (or ~A, if you're going to get picky).

Andy Rooney extended this pattern in his book, "The Most of Andy Rooney" (Galahad Books, 1991), when he suggested changes to the list: "There is a time for putting chocolate sauce on vanilla ice cream, and a time for eating vanilla ice cream without chocolate sauce. … A time for exercise and a time for lying down and taking a little nap. … A time to play basketball, which shall be in the wintertime, and a time to knock it off with the basketball, which shall be as the wintertime comes to an end, and not in June."

Rooney is one of my longtime influences. We both graduated from Colgate University, he in 1942 and I in 1979. I've sometimes described my goals for this column as: "Think of it as a cross between Andy Rooney, Dennis Miller and George Will. Or maybe it's better if you don't."

I've blatantly plagiarized Sol's and Pete's and Andy's idea and made the following observations about the contrasting times in a geek's life. I hope you'll use the e-mail link below to send me your own favorites. My friends, there is:

A time to slam down a huge jolt of caffeine, and a time to sip it slowly over the course of an hour.

A time to save state in object instances, and a time to make them stateless.

A time for bullet-proofing your code and a time for taking short cuts.

A time to listen to Clippy ("I see you're writing a crude forgery") and a time to shoot him in the head.

A time to play Solitaire at meetings, and a time to pay attention to the speaker (rare, unless that speaker is me).

A time for bullet points and a time to ditch PowerPoint and actually talk to your audience.

A time to chase the cat off your laptop keyboard, and a time to use your desktop and let her sleep.

A time to hold up delivery until you can fix a particular feature, a time to chop out the entire feature because you can't fix it and need to make the ship date, and a time to just ship the damn thing.

A time to put your user first and not stroke your own ego. There isn't an opposite to this one.

A time to work until two in the morning, and a time to go home and appreciate still having a family.

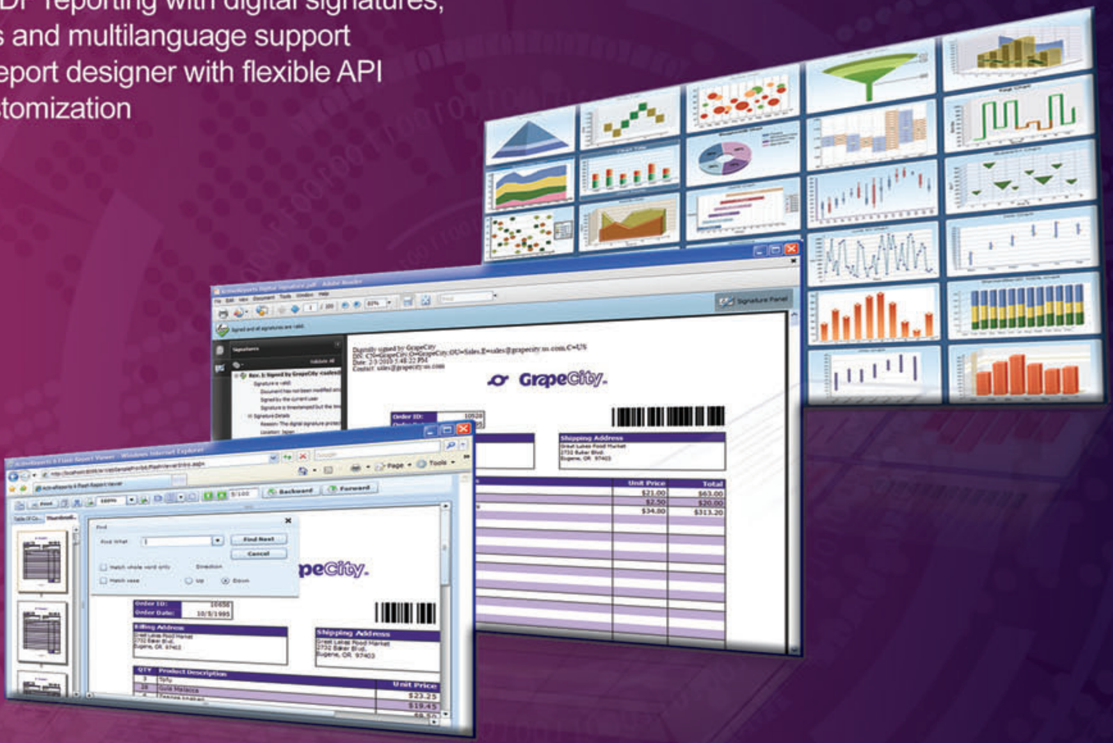
A time to read brilliant columns, and a time to quit goofing off and get back to work already. That's now. ■

DAVID S. PLATT *teaches Programming .NET at Harvard University Extension School and at companies all over the world. He's the author of 11 programming books, including "Why Software Sucks" (Addison-Wesley Professional, 2006) and "Introducing Microsoft .NET" (Microsoft Press, 2002). Microsoft named him a Software Legend in 2002. He wonders whether he should tape down two of his daughter's fingers so she learns how to count in octal. You can contact him at rollthunder.com.*

WE ARE REPORTING

The defacto standard reporting tool for Microsoft Visual Studio.NET

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



ACTIVEREPORTS

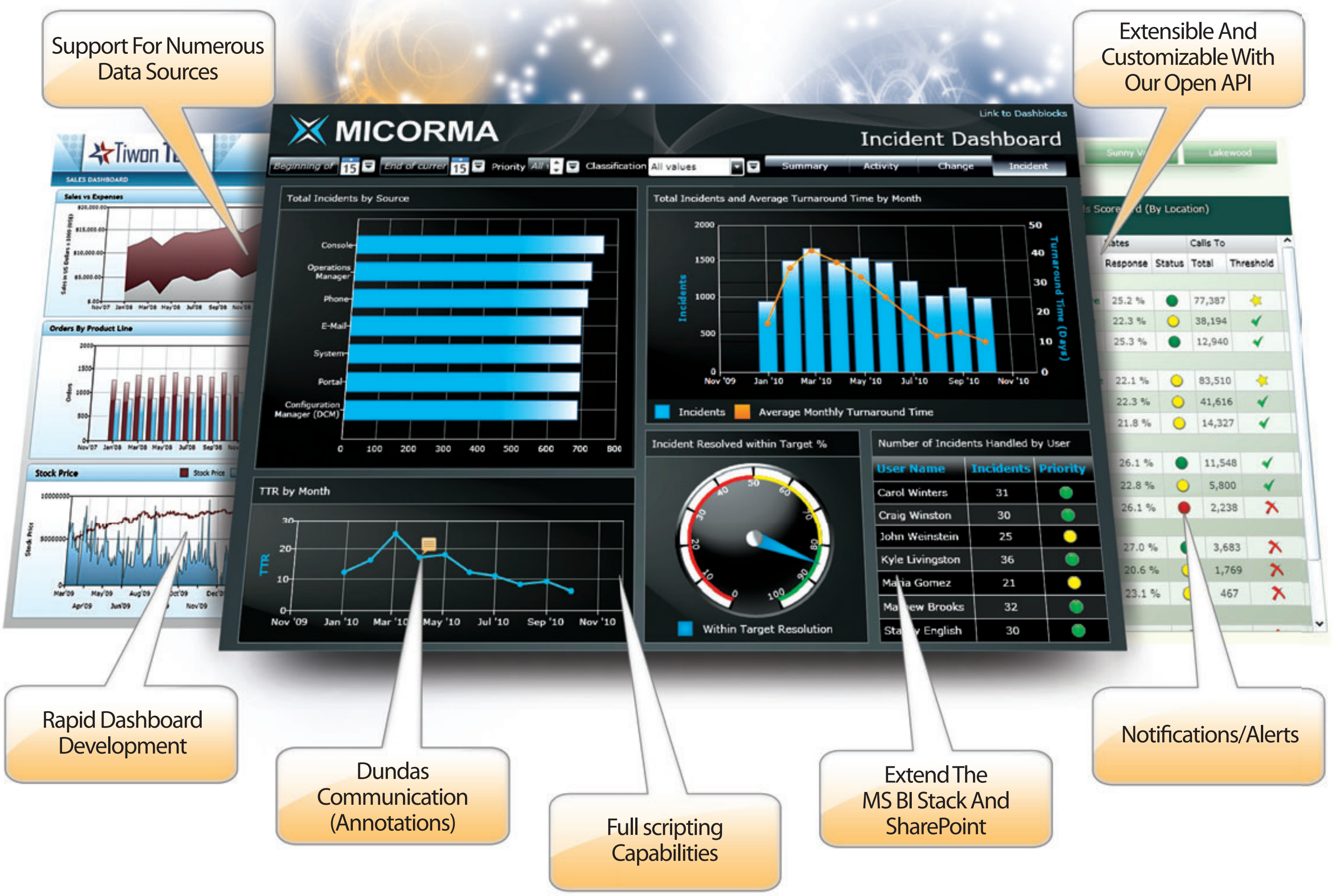
ACTIVEREPORTS SPREAD DATA DYNAMICS REPORTS ACTIVEANALYSIS

GrapeCity PowerTools
Report Analyze Excel

www.GCPowerTools.com

Dundas Dashboard

The Dashboard Framework for Developers like you


Support For Numerous Data Sources
Extensible And Customizable With Our Open API
MICORMA
Incident Dashboard
Total Incidents by Source
Total Incidents and Average Turnaround Time by Month
TTR by Month
Incident Resolved within Target %
Rapid Dashboard Development
Dundas Communication (Annotations)
Full scripting Capabilities
Extend The MS BI Stack And SharePoint
Notifications/Alerts

Number of Incidents Handled by User

User Name	Incidents	Priority
Carol Winters	31	🟢
Craig Winston	30	🟢
John Weinstein	25	🟡
Kyle Livingston	36	🟢
Ma ia Gomez	21	🟡
Ma ew Brooks	32	🟢
Sta y English	30	🟢

V2.5 Now Available

Dundas Dashboard was built with developers and IT staff in mind. Whether it's our open API, simple web integration or powerful scripting capabilities, technologists have all the tools and options they need for getting their dashboard projects up and running quickly and easily.

Powered by Microsoft® Silverlight®

Dundas Data Visualization

www.dundas.com
(416) 467-5100 • (800) 463-1492