



magazine
msdn[®]

DATA IN THE CLOUD

Introducing DataMarket

Elisa Flasko 26

Getting Started with SQL Azure Development

Lynn Langit 30

Synchronizing Multiple Nodes in Windows Azure

Joshua Twist 40

Connecting SharePoint to Windows Azure with Silverlight Web Parts

Steve Fox 50

PLUS:

Scalable Multithreaded Programming with Tasks

Ron Fosner 58

A Coder's Guide to Writing API Documentation

Peter Gruenbaum 70

COLUMNS

CUTTING EDGE

Dynamic Action Filters in ASP.NET MVC
Dino Esposito page 6

FORECAST: CLOUDY

SQL Azure and Windows Azure Table Storage
Joseph Fultz page 12

DATA POINTS

Using the Entity Framework to Reduce Latency to SQL Azure
Julie Lerman page 23

TEST RUN

Web UI Test Automation with the WebBrowser Control
James McCaffrey page 78

THE WORKING PROGRAMMER

Multiparadigmatic .NET, Part 3: Procedural Programming
Ted Neward page 82

SECURITY BRIEFS

Web Application Configuration Security Revisited
Bryan Sullivan page 86

UI FRONTIERS

The Intricacies of Touch Controls
Charles Petzold page 92

DON'T GET ME STARTED

A Real Pain in the Neck
David Platt page 96

Microsoft[®]

DESIGN DEVELOP EXPERIENCE



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 6785 1111 • @infragistics

dtSearch®

Instantly Search Terabytes of Text



◆ 25+ full-text and fielded data search options

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

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

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

◆ Fully-functional evaluations available

Content extraction only licenses also available

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

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

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

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

1-800-IT-FINDS • www.dtSearch.com

The Smart Choice for Text Retrieval® since 1991



NOVEMBER 2010 VOLUME 25 NUMBER 11

LUCINDA ROWLEY Director

DIEGO DAGUM Editorial Director/mmeditor@microsoft.com

KERI GRASSL Site Manager

KEITH WARD Editor in Chief/mmeditor@microsoft.com

TERRENCE DORSEY Technical Editor

DAVID RAMEL Features Editor

WENDY GONCHAR Managing Editor

KATRINA CARRASCO Associate Managing Editor

SCOTT SHULTZ Creative Director

JOSHUA GOULD Art Director

ALAN TAO Senior Graphic Designer

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

Redmond Media Group

Henry Allain President, Redmond Media Group

Matt Morollo Vice President, Publishing

Doug Barney Vice President, Editorial Director

Michele Imgrund Director, Marketing

Tracy Cook Online Marketing Director

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

Matt Morollo VP Publishing

Chris Kourtoglou Regional Sales Manager

William Smith National Accounts Director

Danna Vedder Microsoft Account Manager

Jenny Hernandez-Asandas Director Print Production

Serena Barnes Production Coordinator/msdnadproduction@1105media.com

1105 MEDIA

Neal Vitale President & Chief Executive Officer

Richard Vitale Senior Vice President & Chief Financial Officer

Michael J. Valenti Executive Vice President

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

Christopher M. Coates Vice President, Finance & Administration

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

Carmel McDonagh Vice President, Attendee Marketing

David F. Myers Vice President, Event Operations

Jeffrey S. Klein Chairman of the Board

MSDN Magazine (ISSN 1528-4859) is published monthly by 1105 Media, Inc., 9201 Oakdale Avenue, Ste. 101, Chatsworth, CA 91311-9998, 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. 101, 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



Printed in the USA

Your best source for software development tools!

programmer's paradise®



LEADTOOLS Document Imaging v17.0: by LEAD Technologies

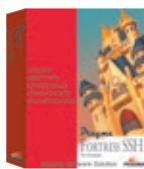
LEADTOOLS Document Imaging has every component you need to develop powerful image-enabled business applications including specialized bi-tonal image display and processing, document clean up, annotation, high-speed scanning, advanced compression (CCITT G3/G4,JBIG2, MRC, ABC) and more.

- Multi-threaded OCR/ICR/OMR/MICR/Barcodes (1D/2D)
- Forms recognition/processing
- PDF and PDF/A
- Win32/x64 binaries for C/C++, .NET, Silverlight, WPF, WCF, and WF

programmers.com/LEAD

Paradise #
L05 03301A01

\$2,007.99



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

by Pragma Systems

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

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

programmers.com/pragma

Paradise #
P35 04201A01

\$550.99



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



FREE WEBINAR SERIES: MAXIMIZING DATA QUALITY FOR VALUE AND ROI

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

REGISTER TODAY! programmers.com/CA

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

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

programmers.com/ca

866-719-1528

Prices subject to change. Not responsible for typographical errors.

STOP OVERBUYING SOFTWARE TODAY!

Eliminate Wasteful Software License Spend:

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



FREE 30-DAY
PROOF OF CONCEPT

Learn more:

programmers.com/eliminate-wasteful-license-spend

InstallShield Professional for Windows 2011

by Flexera Software

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

NEW RELEASE!



FLEXERA SOFTWARE®

InstallShield®



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

\$1,384.99

programmers.com/flexera

VMware vSphere Essentials Kit Bundle

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

- VMware ESXi and VMware ESX (deployment-time choice)
- VMware vStorage VMS
- 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 #
V5585101C02

\$446.99

programmers.com/vSphere



BUILD ON VMWARE ESXI AND VSPPHERE

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

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

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



Microsoft Visual Studio Professional 2010

by Microsoft

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

with MSDN
Paradise #
M47 40201A02

\$1,060.99

programmers.com/microsoft

TX Text Control 15.1

Word Processing Components

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

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

New Service Pack!



Professional Edition
Paradise #
T79 02101A02

\$1,220.99

[Download a demo today.](http://programmers.com/theimagingsource)

programmers.com/theimagingsource

NEW! Intel® Parallel Studio 2011

by Intel

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

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



Single User DVD
Paradise #
I23 63101E03

\$753.99

programmers.com/intel

programmersparadise.com



EDITOR'S NOTE

KEITH WARD

Proactive Education

Talk about striking a nerve: I've gotten more feedback on my columns about how well colleges are preparing students for a career in IT than I've ever gotten on any topic, at any magazine. I'm encouraged that so many people are interested in the next generation of developers and IT pros.

What I'm even more encouraged about is that some folks out there are doing something about it. Among the deluge of e-mails I received was one from Steven Pencle, an applications manager with the Florida Department of Transportation (FDOT), District 4. Pencle told me about a program he and some others started to mentor high school students interested in the IT field, and help nurture that interest into productive careers. It's a story worth sharing.

The program actually got started about five years ago, when the agency hired students from the nearby high school to work on some summer projects as paid interns. These were students in a special school program called the Academy of Information Technology, which partnered with local businesses to mentor students and provide internships. Although the program was successful, the budget eventually evaporated.

The focus of the Mentoring Team is on practical application of IT concepts, not sterile classroom lectures.

Not willing to give up, Pencle and some coworkers took matters into their own hands in 2009 and formed the Information Technology Mentoring Team for local high schools. The Team, currently comprised of four technologists and two human resources specialists, gives presentations and tutoring for interested students. And they are interested, Pencle says. "The enthusiasm from the students and the teachers makes us want to continue. They (the teachers) want us back."

The focus of the Mentoring Team is on practical application of IT concepts, not sterile classroom lectures. For example, Workstation Support Supervisor Radame Gutierrez presented the students with a real challenge. "I disabled a PC with bad memory, [broken] clock, stuff like that, and had them come up and see if they could spot the issues and rebuild the machine," he says.

That kind of hands-on is what sparks students' interest, says Network Security Coordinator Hank Sanchez. During one session, he remembers, "Students were so interested that my presentation took on a life of its own ... they were very involved, and asked great questions. I met one-on-one with them after."

That enthusiasm is often carrying over into college and beyond. Many of the students they worked with, from the past interns to the more recent high school students, are now studying computer science and engineering at colleges and universities such as nearby Florida Atlantic University, or working productively in the field.

One of them, in fact, is now working for FDOT, where she once interned. Melissa Fuentes, senior clerk at the Broward Operations Center, developed her love of IT partially through her paid internship in 2007. She enjoyed it enough that she would've done it for free. "For me, I honestly would go without being paid," she says. "They let me do my own coding; I did Visual Basic and Visual C#. It was great."

The Mentoring Team goes beyond just the technical aspects of work. The HR members advise students on the process of getting hired. This includes advice on how to fill out applications, write resumes and develop interviewing skills.

In his initial e-mail to me, Pencle laid out a vision for the future: "One of my dreams is to see high school IT students and their technology teachers attend a four-day conference similar to Microsoft Tech-Ed where they attend breakout sessions, participate in labs, meet IT mentors and preview upcoming technologies. Like thousands of Tech-Ed attendees, such an experience will forever change their lives and possibly the technology landscape of our nation."

With folks like him and the Mentoring Team working on it, that change is truly possible.

Keith Ward

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

© 2010 Microsoft Corporation. All rights reserved.

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

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

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



OnTime Now!

Cloud-based Scrum Tool for Developers

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

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

axosoft

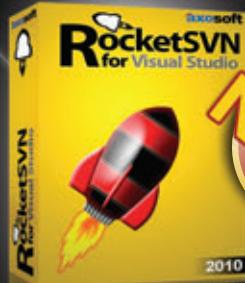
Software for Software Development



Local Scrum Tool for Developers

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

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



Subversion for Visual Studio

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

Just \$49 / user

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



Dynamic Action Filters in ASP.NET MVC

Last month I discussed the role and implementation of action filters in an ASP.NET MVC application. To review a bit: Action filters are attributes you use to decorate controller methods and classes with the purpose of having them perform some optional actions. As an example, you could write a Compress attribute and have it transparently filter any response generated by the method through a compressed gzip stream. The major benefit is that the compression code remains isolated in a distinct and easily reusable class, which contributes to keeping the responsibilities of the method as low as possible.

Attributes, however, are a static thing. To enjoy the benefits of their inherent flexibility, you need to go through an additional compile step. Changing the additional aspects of your controller class is easy, but it comes at the cost of modifying the source code. Generally speaking, this is not a big drawback. Most of the code maintenance work passes through physical changes made to the source code. The more you can make these changes effectively and with no risk of introducing regression, the better.

For Web sites (mostly Web portals) with highly volatile content and features, and for highly customizable software as a service (SaaS) applications, any solution that saves you from touching the source code is more than welcome. So the question is, is there anything you can do to load action filters dynamically? As the rest of this article demonstrates, the answer is a resounding yes.

Inside ASP.NET MVC

The ASP.NET MVC framework exposes a number of interfaces and overridable methods that you can customize nearly every aspect of. In brief, the entire collection of action filters for a controller method

Figure 1 The ControllerDescriptor Class

```
public abstract class ControllerDescriptor :  
    ICustomAttributeProvider {  
  
    // Properties  
    public virtual string ControllerName { get; }  
    public abstract Type ControllerType { get; }  
  
    // Method  
    public abstract ActionDescriptor[] GetCanonicalActions();  
    public virtual object[] GetCustomAttributes(bool inherit);  
    public abstract ActionDescriptor FindAction(  
        ControllerContext controllerContext,  
        string actionName);  
    public virtual object[] GetCustomAttributes(  
        Type attributeType, bool inherit);  
    public virtual bool IsDefined(  
        Type attributeType, bool inherit);  
}
```

is loaded and kept in an in-memory list. As a developer, you're allowed to access and inspect this list. With some more work, you can modify the list of action filters and even populate it dynamically.

Let's take a closer look at how this works with an overview of the steps performed by the framework to perform an action. Along the way, you'll meet the central component whose manipulation allows for dynamic filters: the action invoker.

The action invoker is ultimately responsible for the execution of any action methods on a controller class. The action invoker implements the internal lifecycle of each ASP.NET MVC request. The invoker is an instance of a class that implements the `IActionInvoker` interface. Each controller class has its own invoker object exposed to the world through a plain get/set property named `ActionInvoker`. The property is defined on the base `System.Web.Mvc.Controller` type as follows:

```
public IActionInvoker ActionInvoker {  
    get {  
        if (this._actionInvoker == null) {  
            this._actionInvoker = this.CreateActionInvoker();  
        }  
        return this._actionInvoker;  
    }  
    set {  
        this._actionInvoker = value;  
    }  
}
```

The `CreateActionInvoker` method is a protected overridable method of the `Controller` type. Here's its implementation:

```
protected virtual IActionInvoker CreateActionInvoker() {  
    // Creates an instance of the built-in invoker  
    return new ControllerActionInvoker();  
}
```

It turns out that the action invoker can be changed at will for any controller. However, because the invoker is involved at quite an early stage of the request lifecycle, you probably need a controller factory to exchange your own invoker for the default invoker. Coupled with an Inversion of Control (IoC) framework like Unity, this approach would let you change the invoker logic directly from the (offline) settings of the IoC container.

As an alternative, you can define a custom controller base class for your own application and override the `CreateActionInvoker` method to make it return just the invoker object you need. This is the approach that the ASP.NET MVC framework employs to support the asynchronous execution of controller actions.

The action invoker is built around the `IActionInvoker` interface, which is fairly simple as it exposes just one method:

```
public interface IActionInvoker {  
    bool InvokeAction(  
        ControllerContext controllerContext,  
        String actionName);  
}
```

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

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

Keeping an eye on the default action invoker, let's review the main tasks for which an action invoker is responsible. The invoker first gets information about the controller behind the request and the specific action to perform. Information comes through an ad hoc descriptor object. The descriptor includes the name and type of the controller, plus the list of attributes and actions. For performance reasons, the invoker builds its own cache of action and controller descriptors.

It's interesting to take a quick look at the prototype of the ControllerDescriptor class in **Figure 1**. The class represents just the base class for any real descriptor.

The ASP.NET MVC framework employs two concrete descriptor classes that heavily use the Microsoft .NET Framework reflection internally. One is named ReflectedControllerDescriptor; the other is only used for asynchronous controllers and is named ReflectedAsyncControllerDescriptor.

I can hardly think of a realistic scenario where you might need to create your own descriptor. However, for those who are curious, let's take a look at how it's done.

Imagine you create a derived descriptor class and override the method GetCanonicalActions to read the list of supported actions from a configuration file or a database table. In this way, you can remove valid action methods from the list based on some configurable content. To make this work, however, you need to bring in your own action invoker and write its GetControllerDescriptor method accordingly to return an instance of your custom descriptor:

```
protected virtual ControllerDescriptor
    GetControllerDescriptor(
        ControllerContext controllerContext);
```

Getting information about the controller and action method is only the first step accomplished by the action invoker. Next, and more interestingly for the purposes of this article, the action invoker gets the list of action filters for the method being processed. In addition, the action invoker checks the authorization permissions of the user, validates the request against potentially dangerous posted data and then invokes the method.

Getting the List of Action Filters

Even though the action invoker is identified with the IActionInvoker interface, the ASP.NET MVC framework uses the services of the built-in class ControllerActionInvoker. This class supports a lot of additional methods and features, including the aforementioned descriptors and action filters.

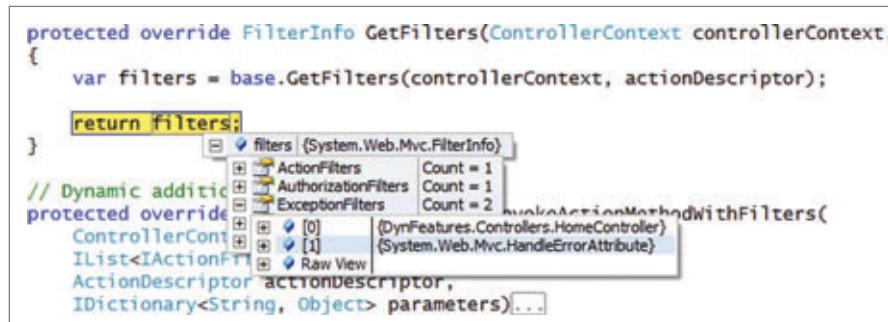


Figure 2 Intercepting the Content of the Filters Collection

The ControllerActionInvoker class offers two main points of intervention for manipulating action filters. One is the GetFilters method:

```
protected virtual ActionExecutedContext
    InvokeActionMethodWithFilters(
        ControllerContext controllerContext,
        IList<IActionFilter> filters,
        ActionDescriptor actionDescriptor,
        IDictionary<string, object> parameters);
```

The other is the InvokeActionMethodWithFilters method:

```
protected virtual FilterInfo GetFilters(
    ControllerContext controllerContext,
    ActionDescriptor actionDescriptor)
```

Both methods are marked protected and virtual, as you can see.

The invoker calls GetFilters when it needs to access the list of filters defined for a given action. As you may guess, this occurs quite early in the lifecycle of a request and earlier than any invocation of the method InvokeActionMethodWithFilters.

You should note that after calling GetFilters, the invoker holds available the entire list of filters for each possible category, including exception filters, result filters, authorization filters and, of course, action filters. Consider the following controller class:

```
[HandleError]
public class HomeController : Controller {
    public ActionResult About() {
        return View();
    }
}
```

The entire class is decorated with the HandleError attribute, which is an exception filter, and no other attribute is visible.

Now let's add a custom invoker, override the method GetFilters and place a breakpoint on the final line of the code, like so:

```
protected override FilterInfo GetFilters(
    ControllerContext controllerContext,
    ActionDescriptor actionDescriptor) {

    var filters = base.GetFilters(
        controllerContext, actionDescriptor);
    return filters;
}
```

Figure 2 shows the actual content of the variable filters.

The FilterInfo class—a public class in System.Web.Mvc—offers specific collections of filters for each category:

```
public class FilterInfo {
    public IList<IActionFilter> ActionFilters { get; }
    public IList<IAuthorizationFilter> AuthorizationFilters { get; }
    public IList<IEffectiveFilter> ExceptionFilters { get; }
    public IList<IRuleFilter> ResultFilters { get; }
    ...
}
```

As in **Figure 2**, for the trivial class shown earlier you count one action filter, one authorization filter, one result filter and two exception filters. Who defined the action, result and authorization filters? The controller class itself is an action filter. In fact, the base Controller class implements all related filter interfaces:

```
public abstract class Controller :
    ControllerBase, IDisposable,
    IActionFilter, IAuthorizationFilter,
    IEffectorFilter, IResultFilter {
    ...
}
```

The base implementation of GetFilters reflects attributes from the controller class using reflection in the .NET Framework. In your implementation of the method GetFilters, you can add as

Full-Featured **Reporting Controls**

Optimized for .NET



Award-Winning Development Tools by DevExpress



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

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

Figure 3 Adding an Action Filter Before Executing the Action

```
protected override ActionExecutedContext
    InvokeActionMethodWithFilters(
        ControllerContext controllerContext,
        IList<IActionFilter> filters,
        ActionDescriptor actionDescriptor,
        IDictionary<String, Object> parameters) {
    if (
        actionDescriptor.ControllerDescriptor.ControllerName == "Home"
        && actionDescriptor.ActionName == "About") {
        var compressFilter = new CompressAttribute();
        filters.Add(compressFilter);
    }
    return base.InvokeActionMethodWithFilters(
        controllerContext,
        filters, actionDescriptor, parameters);
}
```

many filters as you want, reading them from any sort of location. All you need is a piece of code such as this:

```
protected override FilterInfo GetFilters(
    ControllerContext controllerContext,
    ActionDescriptor actionDescriptor) {
    var filters = base.GetFilters(
        controllerContext, actionDescriptor);

    // Load additional filters
    var extraFilters = ReadFiltersFromConfig();
    filters.Add(extraFilters);

    return filters;
}
```

This approach gives you the greatest flexibility and works for any goal you want to achieve or with whatever type of filter you want to add.

Invoking an Action

InvokeActionMethodWithFilters is invoked during the process that takes the performance of the action method. In this case, the method receives the list of action filters to take into account. However, you're still allowed to add extra filters at this time. **Figure 3** shows a sample implementation of InvokeActionMethodWithFilters

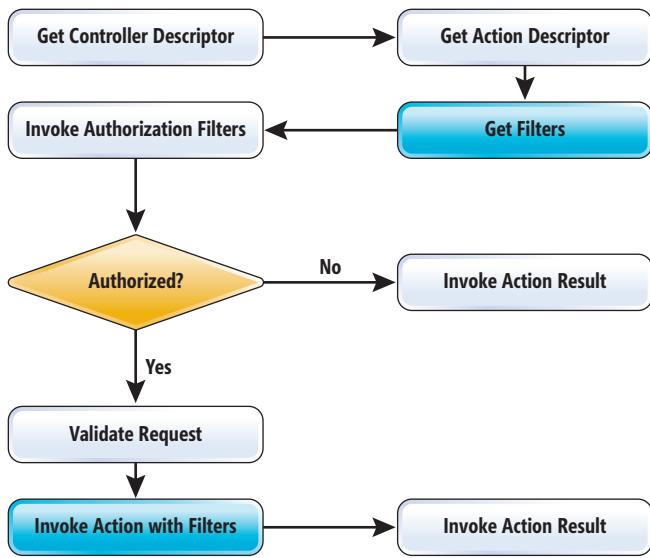


Figure 4 The Lifecycle of an Action Method

that dynamically adds an action filter for compressing the output. The code in **Figure 3** first checks whether the method being invoked is a particular one and then it instantiates and adds a new filter. It goes without saying that you can determine the filters to load in any way that suits you, including reading from a configuration file, a database or whatever else. When overriding the InvokeActionMethodWithFilters method, all you do is check the method being executed, attach additional action filters and invoke the base method so that the invoker can proceed as usual. To retrieve information about the method being executed, you resort to the controller context and the action descriptor.

So you have two possible approaches to add filters dynamically to a controller instance: overriding GetFilters and overriding InvokeActionMethodWithFilters. But, is there any difference?

The Action Lifecycle

Going through GetFilters or InvokeActionMethodWithFilters is pretty much the same thing. Some differences do exist, even though it's no big deal. To understand the difference, let's find out more about the steps taken by the default action invoker when it comes to executing an action method. **Figure 4** summarizes the lifecycle.

After getting descriptors, the invoker gets the list of filters and enters into the authorization phase. At this time, the invoker deals with any authorization filters you may have registered. If the authorization fails, any action result is executed completely ignoring any filters.

Next, the invoker checks whether the request requires validation of posted data and then moves on to executing the action method loading all currently registered filters.

In the end, if you intend to dynamically add any authorization filters, then it will only work if you do it through the GetFilters method. If your goal is only adding action filters, result filters or exception filters, then using either method just produces the same result.

Dynamic Filters

Dynamic loading of filters is an optional feature that mostly serves the purpose of applications with an extremely high-level feature volatility. A filter, specifically an action filter, enables aspect-oriented capabilities in an ASP.NET MVC controller class as it lets developers turn on and off behavior in a declarative manner.

When you write the source code of a controller class, you can choose to add action attributes to the class or the method level. When you read about action filters from an external data source, how to organize information so that it's clear which filters apply to which methods may not be obvious. In a database scenario, you can create a table where you use method and controller name as the key. In a configuration scenario, you probably need to work out a custom configuration section that delivers just the information you need. In any case, the ASP.NET MVC framework is flexible enough to let you decide which filters are to be applied on a per-method and even on a per-call basis. ■

DINO ESPOSITO is the author of “Programming ASP.NET MVC” from Microsoft Press (2010). Based in Italy, Esposito is a frequent speaker at industry events worldwide. You can join his blog at weblogs.asp.net/despos.

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

Powerhouse Analytics Controls

Optimized for .NET



Award-Winning Development Tools by DevExpress



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

DevExpress™
Download • Compare • Decide!

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

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



SQL Azure and Windows Azure Table Storage

A common scenario that plays out in my family is that we decide to take it easy for the evening and enjoy a night out eating together. Everyone likes this and enjoys the time to eat and relax. We have plenty of restaurant choices in our area so, as it turns out, unless someone has a particularly strong preference, we get stuck in the limbo land of deciding where to eat.

It's this same problem of choosing between seemingly equally good choices that I find many of my customers and colleagues experience when deciding what storage mechanism to use in the cloud. Often, the point of confusion is understanding the differences between Windows Azure Table Storage and SQL Azure.

I can't tell anyone which technology choice to make, but I will provide some guidelines for making the decision when evaluating

Data Processing

SQL Azure and other relational databases usually provide data-processing capabilities on top of a storage system. Generally, RDBMS users are more interested in data processing than the raw storage and retrieval aspects of a database.

For example, if you want to find out the total revenue for the company in a given period, you might have to scan hundreds of megabytes of sales data and calculate a SUM. In a database, you can send a single query (a few bytes) to the database that will cause the database to retrieve the data (possibly many gigabytes) from disk into memory, filter the data based on the appropriate time range (down to several hundred megabytes in common scenarios), calculate the sum of the sales figures and return the number to the client application (a few bytes).

To do this with a pure storage system requires the machine running the application code to retrieve all of the raw data over the network from the storage system, and then the developer has to write the code to execute a SUM on the data. Moving a lot of data from storage to the app for data processing tends to be very expensive and slow.

SQL Azure provides data-processing capabilities through queries, transactions and stored procedures that are executed on the server side, and only the results are returned to the app. If you have an application that requires data processing over large data sets, then SQL Azure is a good choice. If you have an app that stores and retrieves (scans/filters) large datasets but does not require data processing, then Windows Azure Table Storage is a superior choice.

—Tony Petrossian, Principal Program Manager, Windows Azure

the needs of the solution and the solution team against the features and the constraints for both Windows Azure Table Storage and SQL Azure. Additionally, I'll add in a sprinkle of code so you can get the developer feel for working with each.

Reviewing the Options

Expanding the scope briefly to include the other storage mechanisms in order to convey a bit of the big picture, at a high level it's easy to separate storage options into these big buckets:

- Relational data access: SQL Azure
- File and object access: Windows Azure Storage
- Disk-based local cache: role local storage

However, to further qualify the choices, you can start asking some simple questions such as:

- How do I make files available to all roles commonly?
- How can I make files available and easily update them?
- How can I provide structured access semantics, but also provide sufficient storage and performance?
- Which provides the best performance or best scalability?
- What are the training requirements?
- What is the management story?

SQL Azure provides
the base functionality of a
relational database.

The path to a clear decision starts to muddy and it's easy to get lost in a feature benefit-versus-constraint comparison. Focusing back on SQL Azure and Windows Azure Table Storage, I'm going to describe some ideal usage patterns and give some code examples using each.

SQL Azure Basics

SQL Azure provides the base functionality of a relational database for use by applications. If an application has data that needs to be hosted in a relational database management system (RDBMS), then this is the way to go. It provides all of the common seman-

Code download available at code.msdn.microsoft.com/mag201011Cloudy.

Elegant Scheduling Controls

Optimized for .NET



Award-Winning Development Tools by DevExpress



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

DevExpress™
Download • Compare • Decide!

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

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

The screenshot shows the Microsoft Visual Studio interface with the title bar "AzureStructuredStorageAccessExample - Microsoft Visual Studio (Administrator)". The main window displays the code editor for "pubsDataSet.Designer.cs" under the "Data Sources" node. The code is for a typed dataset named "pubsDataSet" containing a table named "titleview". The code includes annotations for serialization and schema handling. The Solution Explorer on the right shows the project structure with files like "Program.cs", "pubsDataSet.xsd", and "pubsDataSet.Designer.cs". The Properties window is open for "pubsDataSet.Designer.cs", showing build action as "Compile" and file name as "pubsDataSet.Designer.cs".

Figure 1 Automatically Generated Code for Accessing SQL Azure

tics for data access via SQL statements. In addition, SQL Server Management Studio (SSMS) can hook directly up to SQL Azure, which provides for an easy-to-use and well-known means of working with the database outside of the code.

For example, setting up a new database happens in a few steps that need both the SQL Azure Web Management Console and SSMS. Those steps are:

1. Create database via Web
2. Create rule in order to access database from local computer
3. Connect to Web database via local SSMS
4. Run DDL within context of database container

If an application currently uses SQL Server or a similar RDBMS back end, then SQL Azure will be the easiest path in moving your data to the cloud.

SQL Azure is also the best choice for providing cloud-based access to structured data. This is true whether the app is hosted in Windows Azure or not. If you have a mobile app or even a desktop app, SQL Azure is the way to put the data in the cloud and access it from those applications.

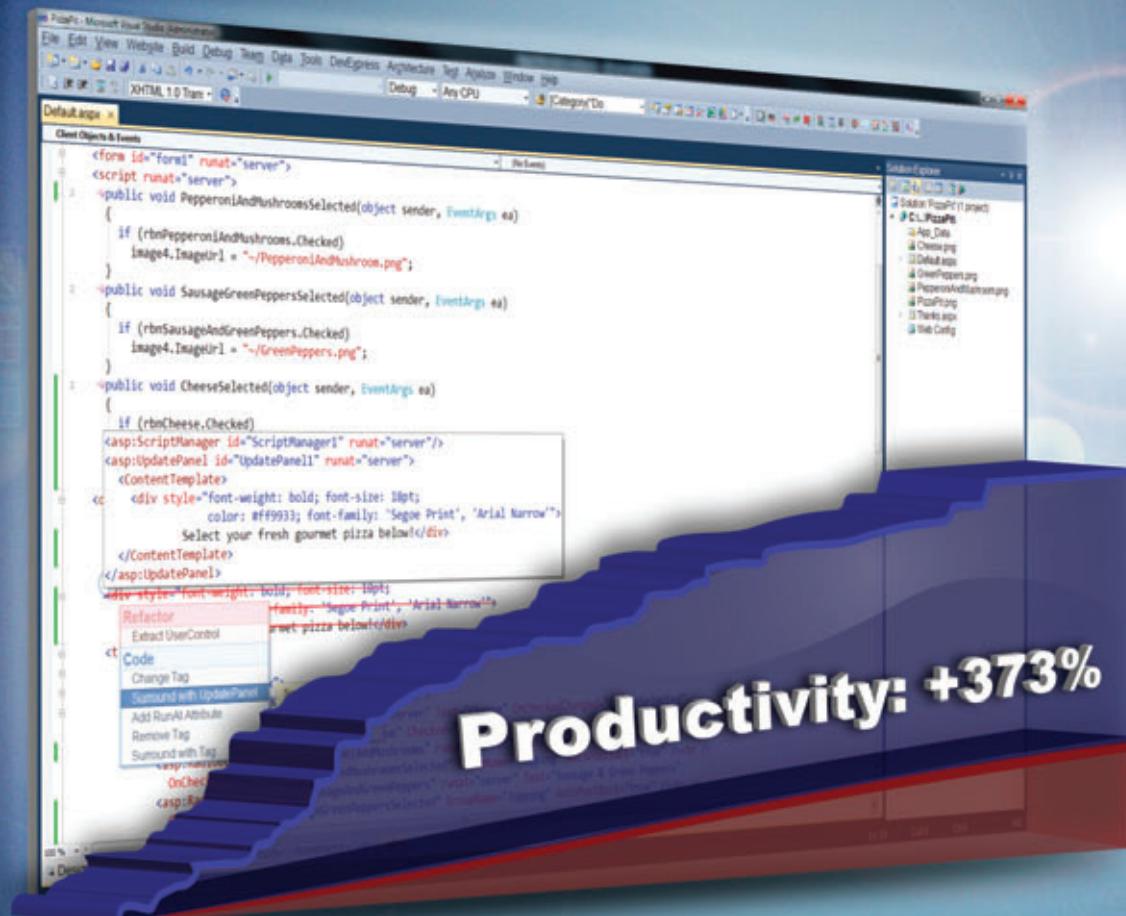
Using a database in the cloud is not much different from using one that's hosted on-premises—with the one notable exception that authentication needs to be handled via SQL Server Authentication. You might want to take a look at Microsoft Project Code-Named "Houston," which is a new management console being developed

The screenshot shows a window titled "Grid to SQL Azure" displaying a data grid with columns: au_id, au_lname, au_fname, phone, and add. The data consists of 15 rows of employee information from the AdventureworksLT database. The first few rows are: au_id 172-32-1176, au_lname White, au_fname Johnson, phone 408 496-7223, add 1093; au_id 213-46-8915, au_lname Green, au_fname Marjorie, phone 415 986-7020, add 309; au_id 238-95-7766, au_lname Carson, au_fname Cheryl, phone 415 548-7723, add 589; au_id 267-41-2394, au_lname O'Leary, au_fname Michael, phone 408 286-2428, add 22 C. The grid has scroll bars on the right and bottom.

Figure 2 SQL Azure Data in a Simple Grid

Rocket Fuel for Visual Studio®

(Superior code, up to **3X** faster)



See **CodeRush** in action: DevExpress.com/373



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

Devexpress
Download • Compare • Decide!

PRESENTATION CONTROLS | REPORTING CONTROLS

IDE PRODUCTIVITY TOOLS | BUSINESS APP FRAMEWORKS

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

The screenshot shows the LINQPad 4 interface. On the left, the Object Explorer tree shows a connection to 'gfkdgapzs5.database.windows.net' with the 'master' and 'pubs' databases selected. Under 'pubs', various tables like Authors, Discounts, Employees, etc., are listed. In the main pane, Query 1* contains the following C# Statement(s):

```
var _authors = from a in Authors
where a.Au_lname == "White"
select a;

_authors.Dump();
```

Below the query, the Results tab displays the output:

Au_Id	Au_lname	Au_fname	Phone	Address	City	State	Zip	Contract
172-32	White	Johnson	408 496-10932	Bigge Rd.	Menlo Park	CA	94025	True
1176			7223					

At the bottom, it says 'Query successful (00:01.017)'.

Figure 3 Using an Entity Model and LINQ

for SQL Azure, built with Silverlight. Details on this project are available at sqlazurelabs.cloudapp.net/houston.

SQL Azure Development

Writing a quick sample application that's just a Windows Form hosting a datagrid that displays data from the Pubs database is no more complicated than it was when the database was local. I fire up the wizard in Visual Studio to add a new data source and it walks me through creating a connection string and a dataset. In this case, I end up with a connection string in my app.config that looks like this:

```
<add name="AzureStrucutedStorageAccessExample.Properties.Settings.
pubsConnectionString"
      connectionString="Data Source=gfkdgapzs5.database.windows.
net;Initial Catalog=pubs;Persist Security Info=True;User
ID=jofultz;Password=[password]"
      providerName="System.Data.SqlClient" />
```

Usually Integrated Authentication is the choice for database security, so it is feels a little awkward using SQL Server Authentication again. SQL Azure minimizes exposure by enforcing an IP access list to which you will need to add an entry for each range of IPs that might be connecting to the database.

Going back to my purposefully trivial example, by choosing the Titleview View out of the Pubs database, I also get some generated code in the default-named dataset pubsDataSet, as shown in **Figure 1**.

I do some drag-and-drop operations by dragging a DataGridView onto the form and configure the connection to wire it up. Once

it's wired up, I run it and end up with a quick grid view of the data, as shown in **Figure 2**.

I'm not attempting to sell the idea that you can create an enterprise application via wizards in Visual Studio, but rather that the data access is more or less equivalent to SQL Server and behaves and feels as expected. This means that you can generate an entity model against it and use LINQ just as I might do so if it were local instead of hosted (see **Figure 3**).

A great new feature addition beyond the scope of the normally available SQL Server-based local database is the option (currently available via [sqlazurelabs.com/](https://odata.azure-labs.com/)) to expose the data as an OData feed. You get REST queries such as this:

```
https://odata.azure-labs.com/
OData.svc/v0.1/gfkdgapzs5/pubs/
authors?$top=10
```

This results in either an OData response or, using the \$format=JSON parameter, a JSON response. This is a huge upside for application developers, because not only do you get the standard SQL Server behavior, but you also get

additional access methods via configuration instead of writing a single line of code. This allows for the focus to be placed on the service or application layers that add business value versus the plumbing to get data in and out of the store across the wire.

If an application needs traditional relational data access, SQL Azure is most likely the better and easier choice. But there are a number of other reasons to consider SQL Azure as the primary choice over Windows Azure Table Storage.

If an application needs traditional relational data access, SQL Azure is most likely the better choice.

The first reason is if you have a high transaction rate, meaning there are frequent queries (all operations) against the data store. There are no per-transaction charges for SQL Azure.

SQL Azure also gives you the option of setting up SQL Azure Data Sync (datasync.azure-labs.com/SADataSync.aspx) between various Windows Azure databases, along with the ability to synchronize data between local databases and SQL Azure installations (microsoft.com/windowsazure/developers/sqlazure/datasync). I'll cover design

FREE ASP.NET Controls

(Yes, they are absolutely free)



Over 20 Free ASP.NET WebForms Controls Including:

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



Download and Start Using Today
Visit DEVEXPRESS.COM/FREEASP

Devexpress
Download • Compare • Decide!

PRESENTATION CONTROLS | REPORTING CONTROLS

IDE PRODUCTIVITY TOOLS | BUSINESS APP FRAMEWORKS

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

Figure 4 Accessing Windows Azure Table Storage

```
public class AuthorDataServiceContext : TableServiceContext {
    public IQueryable<TableStorageAuthor> AuthorData {
        get {
            return this.CreateQuery<TableStorageAuthor>("Authors");
        }
    }

    public AuthorDataServiceContext (
        Uri baseAddress, StorageCredentials credentials)
        : base(baseAddress.AbsoluteUri, credentials) {}

    public void Add(TableStorageAuthor author) {
        this.AddObject("Authors", author);
        DataServiceResponse dsResponse = SaveChanges();
    }
}
```

and use of SQL Azure and DataSync with local storage in a future column, when I cover branch node architecture using SQL Azure.

Windows Azure Table Storage

Now you've seen the advantages of using SQL Azure for your storage. So when is it more beneficial to rely on Windows Azure Table Storage? Here are a number of scenarios where SQL Azure might not be the right choice.

If an application is being overhauled for moving to the Web or the data storage layer implementation isn't completed, you probably want to take a look at Windows Azure Table Storage. Likewise, Windows Azure Table Storage makes sense if you don't need a relational store or access is limited to a single table at a time and doesn't require joins. In this case, your data sets would be small and joins could be handled client-side by LINQ.

You'll also want to take a look at Windows Azure Table Storage if you have more data than the maximum amount supported by SQL Azure (which is currently 50GB for a single instance). Note that size limitation can be overcome with some data partitioning, but that could drive up the SQL Azure costs. The same space in Windows Azure Table Storage would probably be less expensive and has partitioning built-in by a declared partition key.

In addition, due to the per-transaction charges for Windows Azure Table Storage, data with a lower-access frequency or data that can be easily cached would be a good choice.

Some other things that make Windows Azure Table Storage appealing include if the application needs some structured style access such as indexed lookup, but stores primarily objects or

Figure 5 Adding Data to Windows Azure Table Storage

```
var TableClient = StorageAccount.CreateCloudTableClient();

TableStorageAuthor author = new TableStorageAuthor();
author.FirstName = "Joseph";
author.LastName = "Fultz";
author.RowKey = System.Guid.NewGuid().ToString();
author.Id = author.RowKey;
author.State = "TX";
author.PartitionKey = "TX";

AuthorDataServiceContext ctx =
    new AuthorDataServiceContext(
        StorageAccount.TableEndpoint,
        StorageAccount.Credentials);
ctx.Add(author);
```

Binary Large Objects (BLOBS)/Character Large Objects (CLOBs); if your app would benefit from supporting type variability for the data going into the table; or if the existing data structure (or lack thereof) in your SQL Server installation makes it difficult to migrate.

Using Windows Azure Table Storage

At first, working with Windows Azure Table Storage may seem a little unwieldy due to assumptions made by relating "table storage" to a SQL database. The use of "table" in the name doesn't help. When thinking about Windows Azure Table Storage, I suggest that you think of it as object storage.

As a developer, don't focus on the storage structure or mechanism; instead, focus on the object and what you intend to do with it. Getting the objects set up in Windows Azure Table Storage is often the biggest hurdle for the developer, but accessing Windows Azure Table Storage via objects is natural, particularly if you employ LINQ.

When thinking about
Windows Azure Table Storage,
I suggest that you think of it as
object storage.

To work with Windows Azure Table Storage, start by adding a reference to System.Data.Services.Client to your project. In addition, add a reference to Microsoft.WindowsAzure.StorageClient.dll if you aren't working in a Visual Studio Cloud template (which provides this reference for you).

Next, create an object/entity with which you can work (stealing from the Authors table):

```
public class TableStorageAuthor:
    Microsoft.WindowsAzure.StorageClient.TableServiceEntity {
    public int Id {get; set;}
    public string LastName { get; set; }
    public string FirstName { get; set; }
    public string Phone { get; set; }
    public string Address { get; set; }
    public string City { get; set; }
    public string State { get; set; }
    public string Zip { get; set; }
}
```

You can define a data service client context using TableServiceContext to handle connecting to the store and do the Create/Read/Update/Delete (CRUD) operations as shown in **Figure 4**. The TableStorageAuthor class is used as the template class to declare the AuthorData element for which a table query method for the Authors table is returned. It's also used as a parameter type to the implemented Add operation.

Create the target table:

```
TableClient.CreateTableIfNotExist("Authors");
```

Using a familiar object creation and property assignment paradigm, create some data and add it to the table that was created in storage (see **Figure 5**).

Once all of the data is there it can be manipulated with LINQ. For example, a select for the entities would be:

Figure 6 Comparing SQL Azure and Windows Azure Table Storage

Feature	SQL Azure	Common Benefit(s)	Windows Azure Table Storage
Select semantics	Cross-table queries	Primary key-based lookup	Single key lookup (by partition)
Performance and scale	High performance via multiple indices, normalized data structures and so on, and scalable via manual partitioning across SQL Azure instances		Automatic mass scale by partition and consistent performance even at large scale
User experience	Well-known management tools and traditional database design	Familiar high-level developer experience	Direct serialization; no ORM necessary; simplified design model by removing relational model
Storage style	Traditional relational design model	Data storage for all types of data	Type variability in a single table
Cost factors	No transaction cost, pay by database size	Network traffic cost outside of same datacenter	No space overhead cost, pay for what is used
Data loading and sync	Synchronizing between local and cloud-based stores; data easily moved in and out by traditional extract, transform and load (ETL) mechanisms; synchronizing between SQL Azure databases in different datacenters		

```

AuthorDataContext ctx =
    new AuthorDataContext(
        StorageAccount.TableEndpoint,
        StorageAccount.Credentials);

var authors =
    from a in ctx.AuthorData
    select a;

foreach (TableStorageAuthor ta in authors) {
    Debug.WriteLine(ta.FirstName + " " + ta.LastName);
}

```

I didn't implement update and delete, but they would be similar. The only thing that might be a little different from those that have used LINQ with the Entity Framework is the code to create the TableServiceContext and the subsequent code to construct and use it. If you've been working with REST and the DataServiceContext, doing this work will be quite natural.

Many applications will need a hybrid data approach to make the best use of technology.

By using the TableServiceContext, TableServiceEntity and LINQ, you get about the same feel as using the Entity Framework and LINQ with SQL Azure—albeit with a little more hand-coding on the Windows Azure Table Storage side.

Solution-Based Evaluation

As mentioned before, if the application has a relational store already established, it's likely best to migrate that to SQL Azure with a little help from a tool like the SQL Azure Migration Wizard. However, if that's not the case, or the cloud piece of the application doesn't need the full functionality of an RDBMS, then take a look at the matrix in **Figure 6** and see which columns meet most of the needs of the solution requirements and architecture.

It's important to note that for some of the items in **Figure 6** (for example, those related to management and data loading for Windows Azure Table Storage) there are already third-party solutions entering the market to provide the missing functionality. As such, the cost and functionality of such tools will need to be considered for significant projects.

I expect that many applications will need a hybrid data approach to make the best use of technology. For example, Windows Azure Table Storage would be used to optimize fetch times while still providing mass scale for resources such as documents, videos, images and other such media. However, to facilitate searching metadata for the item, related data and object pointers would be stored in SQL Azure. Such a design would also reduce the transaction traffic against Windows Azure Table Storage. This type of complementary design would provide the following benefits:

- Keep the throughput high for queries to find resources
- Keep the SQL Azure database size down, so cost for it remains a minimum
- Minimize the cost of storage by storing the large files in Windows Azure Table Storage versus SQL Azure (though BLOB storage is preferred for files)
- Maintain a fast retrieval performance by having such resources fetch by key and partition, and offloading the retrieval query from the SQL Azure database
- Allow for automatic and mass scale for the data kept in Windows Azure Table Storage

Simply put: Your design should allow each storage mechanism to provide the part of the solution that it's best at performing, rather than trying to have one do the job of both. Either way, you're still looking to the cloud for an answer. ■

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 has spoken at events such as Tech-Ed and similar internal training events.

THANKS to the following technical experts for reviewing this article:
Jai Haridas, Tony Petrossian and Suraj Puri

POWERFUL DATA ACCESS.

POINT-AND-CLICK SIMPLICITY.

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

Powerful Visual Designer

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

Forward and Reverse Mapping capabilities

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

Saves development time

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

Full LINQ support

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

Automatic web service layer

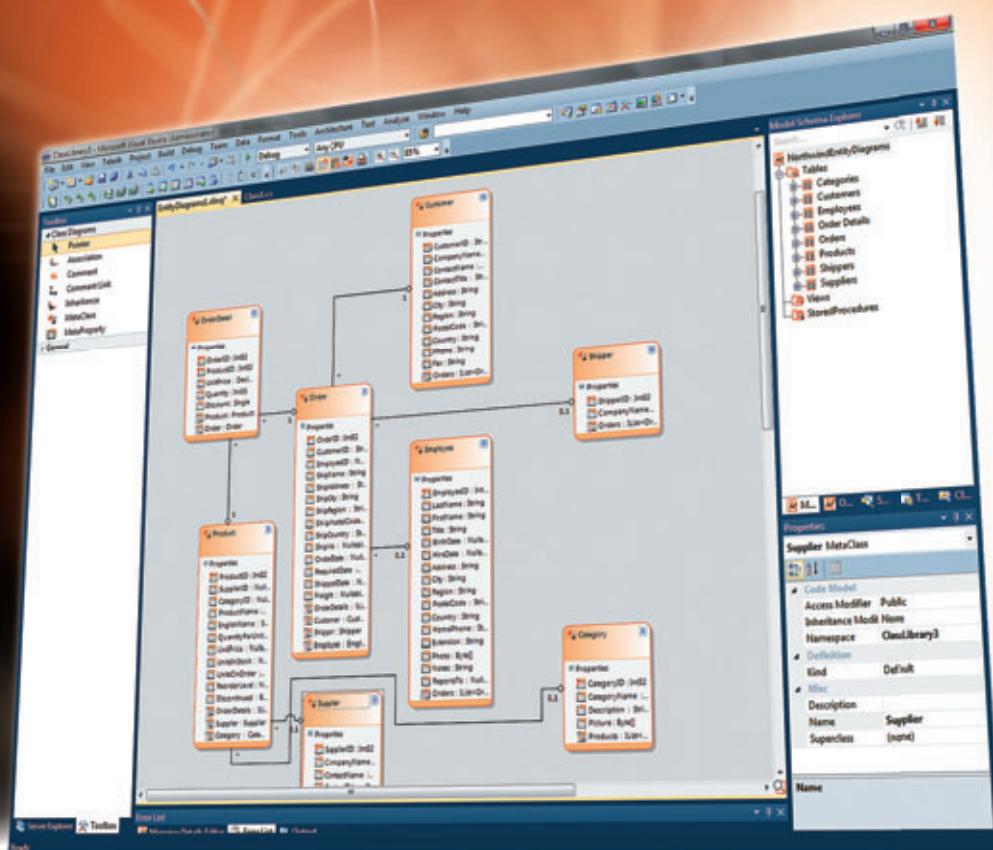
Automatically create a complete Plain WCF, Wcf Data or Wcf Ria web service layer for your OpenAccess ORM project using the built-in Telerik Data Services Wizard.

Silverlight support

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

Optimized for complex scenarios

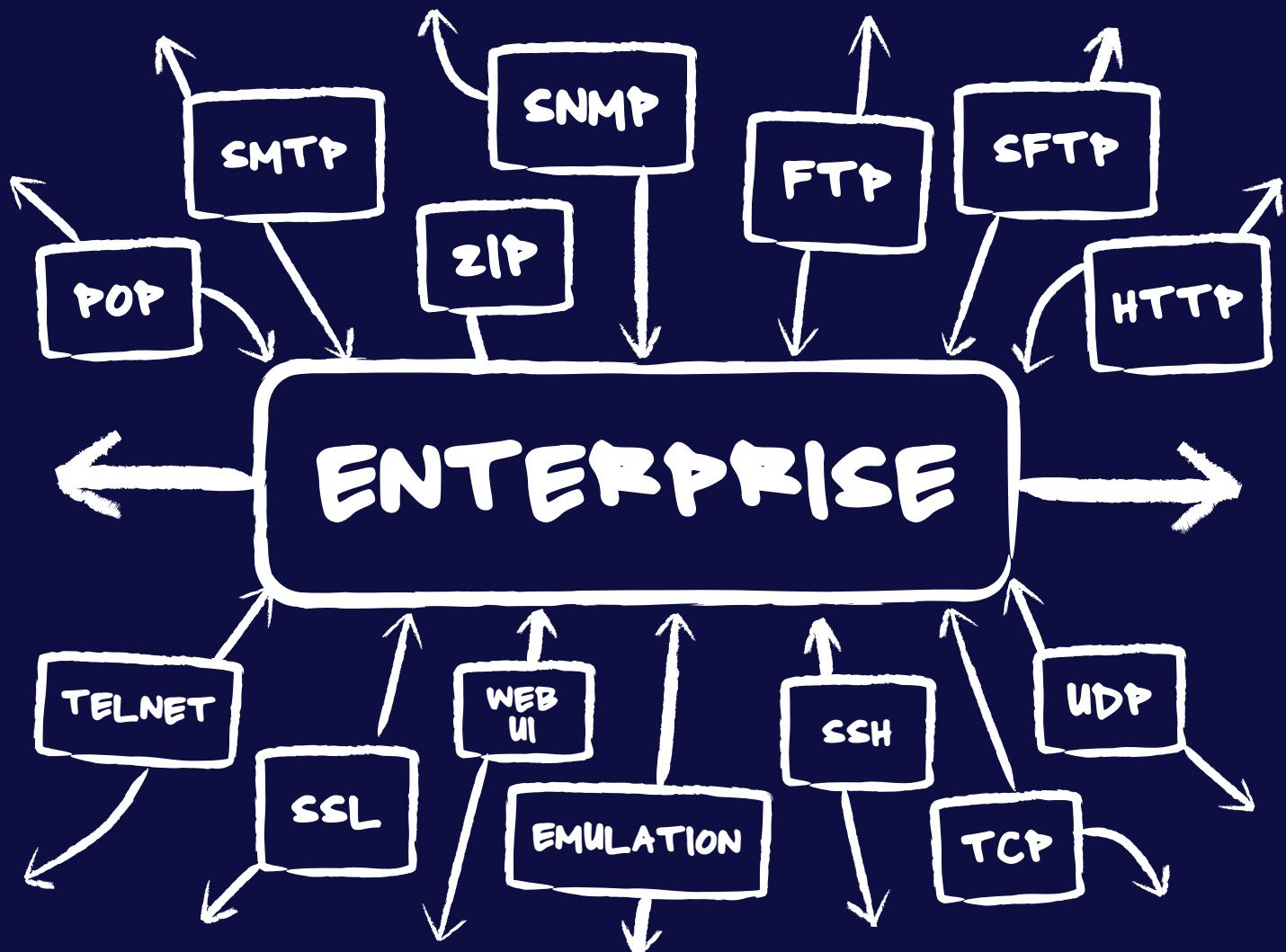
Tackle even the most complex scenarios with an array of built-in features such as distributed caching, configurable fetch plans, support for n-tier and disconnected applications, support for stored procedures, and many more.



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

www.telerik.com/orm

 **telerik**
deliver more than expected



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!



Using the Entity Framework to Reduce Network Latency to SQL Azure

At first glance, switching from a locally managed SQL Server database to the Microsoft cloud-based SQL Azure database sounds like it could be difficult. But in reality, it's a snap: Simply switch the connection string and you're done! As we developers love to see in these situations, it "just works."

However, making the switch introduces network latency, which can substantially affect overall application performance. Fortunately, a good understanding of the effects of network latency leaves you in a powerful position to use the Entity Framework to reduce that impact in the context of SQL Azure.

Profiling Your Data Access Code

I use Visual Studio 2010 profiler tools (msdn.microsoft.com/library/ms182372) to compare different data access activities against the AdventureWorksDW database residing on both my local network and my SQL Azure account. I use the profiler to investigate calls to load some customers from the database using the Entity Framework.

Figure 1 Queries Designed to Explore Performance

```
static void Main()
{
    using (var context = new AdventureWorksDWEntities())
    {
        var warmupquery = context.Accounts.FirstOrDefault();
    }
    DoTheRealQuery();
}

private static void DoTheRealQuery()
{
    using (var context = new AdventureWorksDWEntities())
    {
        var query = context.Customers.Where(c => c.InternetSales.Any()).Take(100);
        var customers = query.ToList();
        EnumerateCustomers(customers);
    }
}

private static void EnumerateCustomers(List<Customer> customers)
{
    foreach (var c in customers)
    {
        WriteCustomer(c);
    }
}

private static void WriteCustomer(Customer c)
{
    Console.WriteLine(
        "CustomerName: {0} First Purchase: {1} # Orders: {2}",
        c.FirstName.Trim() + " " + c.LastName, c.DateFirstPurchase,
        c.InternetSales.Count);
}
```

One test queries only the customers and then retrieves their sales information after the fact using the lazy loading feature of the Entity Framework 4.0. A later test eager loads customer sales along with the customers using the Include method. **Figure 1** shows the console app that I used to execute these queries and enumerate the results.

I begin with a warm-up query in order to get past the expense of loading the Entity Data Model (EDM) metadata into memory, pre-compiling views and other one-time operations. The DoTheRealQuery method then queries a subset of Customers entities, executes the query into a list and enumerates the results. During the enumeration, the customer's sales are accessed, forcing—in this case—a lazy load that goes back to the database to get the sales for each customer throughout the iteration.

Looking at the Performance on a Local Network

When running this on my local network against an on-premises SQL Server database, the first call to execute the query takes 233 ms. That's because I'm only retrieving the customers. When the code runs the enumeration that forces the lazy load, it takes 195 ms.

Now I'll change the query so that it eager loads the InternetSales along with Customers:

```
context.Customers.Include("InternetSales").Where(c => c.InternetSales.
    Any()).Take(100);
```

Now those 100 customers, along with all of their sales records, will be returned from the database. That's a lot more data.

The query.ToList call now takes about 350 ms, nearly 33 percent longer than returning just the 100 customers.

There's another effect of this change: As the code enumerates through the customers, the sales data is already there in memory. That means the Entity Framework doesn't have to make an extra 100 round-trips to the database. The enumeration, along with writing out the details, takes only about 70 percent of the time it took when lazy loading was doing its job.

Taking into account the amount of data, the computer and local network speed, overall, the lazy-loading route in this particular scenario is a little faster than when you eager load the data. However, it's still fast enough that the difference isn't even noticeable between the two. They both appear to run blazingly fast thanks to the Entity Framework.

Figure 2 shows a comparison of eager and lazy loading in the local network environment. The ToList column measures the query execution, which is the code line: `var customers = query.ToList();`. The Enumeration measures the `EnumerateCustomers` method. And finally, the `Query & Enumeration` column measures the complete

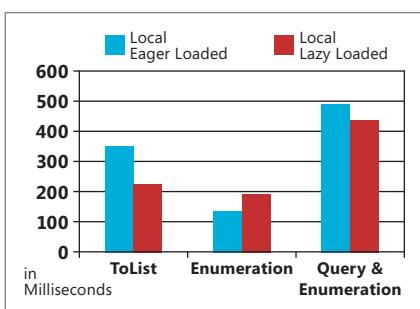


Figure 2 Comparing Eager Loading to Lazy Loading from a Local Database

DoTheRealQuery method, which combines the execution, enumeration, instantiation of the context and declaration of the query itself.

Switching to the SQL Azure Database

Now I'll switch the connection string to my SQL Azure database in the cloud. It shouldn't be surprising that network latency between my computer and the cloud database kicks in, making queries take longer than against the local database. You can't avoid latency in this scenario. However, what is notable is that the increase isn't equal among the various tasks. For some types of requests, the latency is much more exaggerated than others. Take a look at **Figure 3**.

Eager loading the graphs is still slower than only loading the customers up front. But where it was about 30 percent slower locally, on SQL Azure it now takes about three times as long as lazy loading.

But as you can see in **Figure 3**, it's the lazy loading that's most impacted by the network latency. After the InternetSales data was in memory thanks to the eager loading, enumerating over the data is as quick as the first set of tests. But the lazy loading is causing 100 additional round-trips to the cloud database. Because of the latency, each of these trips takes longer—and in combination, the resulting time to retrieve the results is visibly noticeable.

The enumeration takes more time than the in-memory enumeration by orders of magnitude. Each trip to the database to get a single customer's InternetSales data takes a significant amount of time. Overall—even though it's surely much faster to load only the Customers up front—in this environment it took almost six times longer to retrieve all of the data with lazy loading.

The point of all this isn't to incriminate SQL Azure, which in fact is highly performant, but to point out that your choice of Entity Framework query mechanism can have a negative impact on overall performance because of the latency issues.

The particular use case of this demo isn't typical for an application, because, typically, you wouldn't lazy load related data for a large series of objects. But it does highlight that, in this case, returning a lot of data all at once (via eager loading) is much more efficient than returning that same amount of data across a multitude of trips to the database. When you're using a local database, the difference may not be as significant as it is when your data is in the cloud, so it merits a close look when you switch from your local database to SQL Azure.

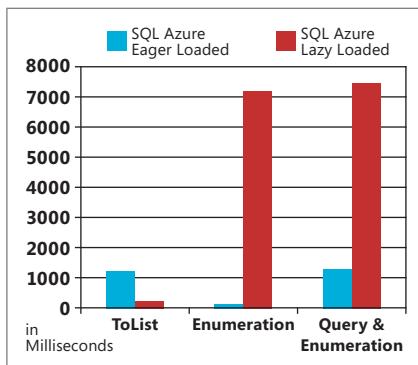


Figure 3 Comparing Eager Loading to Lazy Loading from SQL Azure

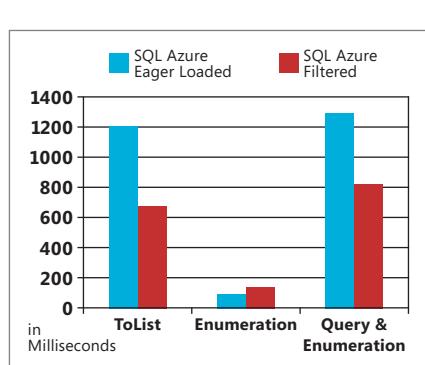


Figure 4 Comparing Eager Loading to a Filtered Projection from SQL Azure

Depending on the shape of the data that you're returning—perhaps a much more complex query with numerous Includes, for example—there are times when eager loading is more expensive and times when lazy loading is more expensive. There are even times when it will make sense to distribute the load: eager load some data and lazy load others based on what you learn from performance profiling. And in many cases, the ultimate solution is to move your application to the cloud, as well. Windows Azure and SQL Azure were designed to work as a team. By moving your application to Windows Azure and having that application get its data from SQL Azure, you'll maximize performance.

Use Projections to Fine-Tune Query Results

When using applications that are running locally, in some scenarios, you may be able to revise the queries to further refine the amount of data returned from the database. One technique to consider is using projections, which grant you much more control over what related data is brought back. Neither eager loading nor deferred/lazy loading in the Entity Framework allows you to filter or sort the related data being returned. But with a projection, you can.

For example, the query in this modified version of TheRealQuery method returns only a subset of InternetSales entities—those greater than or equal to 1,000:

```
private static void TheRealQuery()
{
    using (var context = new AdventureWorksDWEntities())
    {
        Decimal salesMinimum = 1000;
        var query =
            from customer in context.Customers.Where(c =>
                c.InternetSales.Any()).Take(100)
            select new { customer, Sales = customer.InternetSales };
        IEnumerable<Customer> customers = query.ToList();
        context.ContextOptions.LazyLoadingEnabled = false;
        EnumerateCustomers(customers);
    }
}
```

The query brings back the same 100 customers as the previous query, along with a total of 155 InternetSales records compared to the 661 sales records that are returned without the SalesAmount filter.

Mind this important note about projections and lazy loading: When projecting the data, the context doesn't recognize the related data as having been loaded. That only happens when it's loaded via the Include method, the Load method or lazy loading. Therefore, it's important to disable lazy loading prior to the enumeration as

I've done in the `TheRealQuery` method. Otherwise, the context will lazy load the `InternetSales` data even though it's already in memory, causing the enumeration to take much longer than necessary.

The modified enumeration method takes this into account:

```
private static void EnumerateCustomers(IEnumerable customers)
{
    foreach (var item in customers)
    {
        dynamic dynamicItem=item;
        WriteCustomer((Customer)dynamicItem.customer);
    }
}
```

The method also takes advantage of the `dynamic` type in C# 4 to perform late binding.

Figure 4 demonstrates the significant performance gain realized by the more finely tuned projection.

It may seem obvious that the filtered query would be faster than the eager-loaded query that returns more data. The more interesting comparison is the fact that the SQL Azure database processes the filtered projection about 70 percent faster, while on a local database the filtered projection is only about 15 percent faster than the eager-loaded query. I suspect that the eager-loaded `InternetSales` collection causes the in-memory enumeration to be faster because of the way the Entity Framework accesses it internally compared to the projected collection. But as the enumeration in this case is occurring completely in memory, it has no bearing on the network latency. Overall, the improvement seen with projection outweighs the small enumeration price when using the projection.

On your network, switching to a projection to fine-tune the query results may not seem necessary, but against SQL Azure, this type of tuning can realize significant performance gains in your application.

All Aboard the Cloud

The scenarios I've discussed revolve around locally hosted apps or services that use SQL Azure. You may instead have your application or service hosted in the cloud in Windows Azure, as well. For example, end users may be using Silverlight to hit a Windows Azure Web Role running Windows Communication Foundation that in turn accesses data in SQL Azure. In this case, you'll have no network latency between the cloud-based service and SQL Azure.

Whatever the combination, the most important point to remember is that even though your application continues to function as expected, performance can be affected by network latency. ■

JULIE LERMAN is a Microsoft MVP, .NET mentor and consultant who lives in the hills of Vermont. You can find her presenting on data access and other Microsoft .NET topics at user groups and conferences around the world. Lerman blogs at thedatafarm.com/blog and is the author of the highly acclaimed book, "Programming Entity Framework" (O'Reilly Media, 2010). Follow her on Twitter.com: [@julielerman](http://julielerman).

THANKS to the following technical experts for reviewing this article:
Wayne Berry, Kraig Brockschmidt, Tim Lavery and Steve Yi



The banner for the Microsoft Dynamics AX Technical Conference 2011 features the Microsoft Dynamics AX logo and the text "TECHNICAL CONFERENCE 2011". Below the text is a photograph of two men in a conference setting, one looking at a screen and the other listening. A blue bar at the bottom contains the words "Learn | Inspire | Innovate".

January 17 - 20, 2011 • Redmond, WA

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

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



Experience How. Get Involved.

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

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

Microsoft

Introducing DataMarket

Elisa Flasko

Windows Azure Marketplace DataMarket, which was first announced as Microsoft Project Codename “Dallas” at PDC09, changes the way information is exchanged by offering a wide range of content from authoritative commercial and public sources in a single marketplace. This makes it easier to find and purchase the data you need to power your applications and analytics.

If I were developing an application to identify and plan stops for a road trip, I would need a lot of data, from many different sources. The application might first ask the user to enter a final destination and any stops they would like to make along the way. It could pull the current GPS location or ask the user to input a starting location and use these locations to map the best route for the road trip. After the application mapped the trip, it might reach out to Facebook and identify friends living along the route who I may want to visit. It might pull the weather forecasts for the cities that have been identified as stops, as well as identify points of interest, gas stations and restaurants as potential stops along the way.

Before DataMarket, I would've had to first discover sources for all the different types of data I require for my application. That could entail visiting numerous companies' Web sites to determine whether or not they have the data I want and whether or not they offer it for sale in a package and at a price that meets my needs. Then I would've had to purchase the data directly from each company. For example, I might have gone directly to a company such as Infogroup to purchase the data enabling me to identify points of interest, gas stations and restaurants along the route; to a company such as NavTeq for current traffic reports; and to a company

such as Weather Central for my weather forecasts. It's likely that each of these companies would provide the data in a different format, some by sending me a DVD, others via a Web service, Excel spreadsheet and so on.

Today, with DataMarket, building this application becomes much simpler. DataMarket provides a single location—a marketplace for data—where I can search for, explore, try and purchase the data I need to develop my application. It also provides the data to me through a uniform interface, in a standard format (OData—see OData.org for more information). By exposing the data as OData, DataMarket ensures I'm able to access it on any platform (at a minimum, all I need is an HTTP stack) and from any of the many applications that support OData, including applications such as Microsoft PowerPivot for Excel 2010, which natively supports OData.

DataMarket provides a single marketplace for various content providers to make their data available for sale via a number of different offerings (each offering may make a different subset or view of the data available, or make the data available with different terms of use). Content providers specify the details of their offerings, including the terms of use governing a purchase, the pricing model (in version 1, offerings are made available via a monthly subscription) and the price.

Getting Started with DataMarket

To get started, I register on DataMarket using my Windows Live ID and log on to the site. From here, I can search through the datasets currently available on the site, viewing the details of each (description, pricing, visualizations, terms of use and so on) to determine which publishers deliver the type of data I'm looking for and which offers best suit my needs.

Although I need to find a dataset that meets the technical requirements of my application, it's important to also verify that the terms of use set forth by the publisher allow the data to be used in the way required by my application. Terms of use can vary widely across datasets and across publishers. I can view the terms of use for a specific dataset within the offering details.

This article discusses:

- From Project Codename “Dallas” to DataMarket
- Registering at DataMarket and purchasing data
- Consuming OData in a road trip application
- Selling data on DataMarket

Technologies discussed:

Windows Azure, OData

After digging around a bit, I find that the Infogroup Business Database service looks like it could be interesting for my road trip application. After looking through the available visualizations of the data (a sample of the data in table format, or in some cases a visualization of the data on a map or in a chart), it looks like it should provide me with points of interest including hotels, restaurants and gas stations along the route, and it should meet my requirements. After I've determined which datasets fit my needs, I can choose to purchase a monthly subscription to the service, allowing me an unlimited number of queries against the service per month, and I can simply pay with a credit card. After I've purchased something, I can manage my account and view all my current subscriptions in the Account section of the Marketplace. The Account section can also be used to create and manage account keys, which I use to access my subscriptions. When accessing data from an application such as PowerPivot, I'll need to provide my account key. Similarly, when developing an application, it will authenticate with an account key.

After I've purchased a data offering, DataMarket includes a Service Explorer, as seen in **Figure 1**, which allows me to explore a dataset by creating queries and previewing the results, effectively learning more about the data and the schema that each dataset makes available.

If I open up the Infogroup Business Database service in the Service Explorer, I can create queries that filter by city, state and ZIP code to find places along the planned route. For example, if I specify Seattle as the city and WA as the state and click on "Execute query," I get back a preview including the first 100 results from the service. I can see from the preview that there's a Westin Hotel in Seattle that I'll want my application to include in the list of stops. By default, when I click Preview, the results show up on the right side of the explorer in a basic tabular format to make it easy to read through. But I can also view the OData Atom format that I'll use in my application right there in the Service Explorer, or view the raw data in the same format originally provided by Infogroup.

If I wasn't a developer, or if my goal wasn't to build an application with this data, but rather just pull the data into an existing tool that understands OData—such as PowerPivot for Excel 2010—I could also do that directly from DataMarket simply by signing in, opening my current subscriptions list and clicking on the data offering. From there I will be given the option to open the data offering in the application of my choice. (For more information on using DataMarket with PowerPivot, including a step-by-step tutorial, be sure to check out the DataMarket team blog at blogs.msdn.com/dallas.)

Consuming OData

Have you previously built an application that consumes data from an OData service? If the answer is yes, you're already well on your way to consuming data from DataMarket in your app, because most DataMarket services are exposed using OData. I'll quickly go through the basics of consuming OData.

Whether I begin with an existing application and I'm simply adding in new features that utilize data from DataMarket, or if I'm creating a new application from scratch, the first step in pulling data in from a DataMarket service is to define the classes that I'll use to represent the data in my application. I can do this by coding my own Plain Old C# Object (POCO) classes or by using the Add Service Reference Wizard in Visual Studio to generate the necessary classes.

As described earlier, the publisher (the owner of the data) specifies how the data for each dataset is exposed and the terms of use dictating how the data may be used after purchase. This includes specifying how or if a user can create ad hoc queries against a dataset, which fields a user can query on to retrieve the data and which fields are returned. In some cases, a publisher may specify that users can't create ad hoc queries and must use fixed Web methods to request data from the service. As a developer, before beginning to write an application, I'll need to look at the Atom Service document for a particular offering to determine whether the offering exposes any entity sets for query. I can do this by pointing the browser at the root URI of an offering. If the service document doesn't contain any collections, the offering doesn't expose entity sets for query and I'll be required to access the dataset using fixed Web methods. For example, let's examine the Atom Service documents for two separate datasets, one that exposes an Entity Set for query (Data.Gov Crime Data), and one that doesn't allow query at all (AP Data). Notice in

The screenshot shows a web browser window titled "Windows Azure DataMarket - https://dallas-testportal.cloudapp.net". The main content area displays a table titled "2006 and 2007 Crime in the United States - DATA.gov". The table has columns: ROWID, State, City, Year, Population, ViolentCrime, MurderAndNonNigentManslaughter, and ForcibleF. The data is filtered for Alaska. On the left, there is a sidebar titled "Service parameters" with dropdowns for Series (set to "CityCrime [CityCrime]"), RowID, State, City, and Year. There are also checkboxes for "Enable paging", "Start page", and "Items per page". Below these are buttons for "Execute Query", "Analyze In", and "PowerPivot". The table itself contains approximately 20 rows of data, showing crime statistics for various Alaskan cities across the specified years.

	ROWID	State	City	Year	Population	ViolentCrime	MurderAndNonNigentManslaughter	ForcibleF
115856	Alaska	Anchorage	2007	284342	2405	32	0	257
115857	Alaska	Bethel	2007	6488	81	2	0	15
115858	Alaska	Bristol Bay Borough	2007	3528	8	0	0	0
115859	Alaska	Cordova	2007	2322	9	0	0	0
115860	Alaska	Craig	2007	1186	41	0	0	0
115861	Alaska	Dillingham	2007	2494	76	1	0	24
115862	Alaska	Fairbanks	2007	31287	256	5	0	43
115863	Alaska	Haines	2007	2260	16	0	0	3
115864	Alaska	Homre	2007	5629	52	0	0	2
115865	Alaska	Hebron	2007	2982	3	0	0	0
115866	Alaska	Juneau	2007	30746	126	1	0	22
115867	Alaska	Kemai	2007	3620	30	1	0	2
115868	Alaska	Ketchikan	2007	7384	21	0	0	9
115869	Alaska	Kodiak	2007	6242	33	0	0	5
115870	Alaska	North Pole	2007	1868	9	0	0	1
115871	Alaska	North Slope Borough	2007	8569	70	0	0	15
115872	Alaska	Palmer	2007	7938	65	0	0	4
115873	Alaska	Petersburg	2007	2890	1	0	0	0
115874	Alaska	Seward	2007	3054	3	0	0	1
115875	Alaska	Sitka	2007	8932	31	0	0	9
115876	Alaska	Skagway	2007	827	1	0	0	0
115877	Alaska	Soldotna	2007	4198	21	0	0	2
115878	Alaska	St. Paul	2007	431	6	0	0	2

Figure 1 DataMarket Service Explorer

the following code samples that the AP offering has no <collection> nodes (representing exposed entity sets) in the service document.

Here's an Atom Service document for the Data.gov Crime data offering, which exposes entity sets for query. This particular offering exposes a single entity set, CityCrime:

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<service xml:base=
  "https://api.datamarket.azure.com/Data.ashx/data.gov/Crimes"
  xmlns:atom="http://www.w3.org/2005/Atom"
  xmlns:app="http://www.w3.org/2007/app"
  xmlns="http://www.w3.org/2007/app">
<workspace>
  <atom:title>Default</atom:title>
  <collection href="CityCrime">
    <atom:title>CityCrime</atom:title>
  </collection>
</workspace>
</service>
```

Here's an Atom Service document for the AP offering, which doesn't allow ad hoc query:

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<service xml:base=
  "https://api.datamarket.azure.com/Data.ashx/data.gov/Crimes"
  xmlns:atom="http://www.w3.org/2005/Atom"
  xmlns:app="http://www.w3.org/2007/app"
  xmlns="http://www.w3.org/2007/app">
<workspace>
  <atom:title>Default</atom:title>
</workspace>
</service>
```

Figure 2 A Simple Console Application Accessing a Data Service and Printing the Results

```
public class Program
{
    static void Main(string[] args)
    {
        GetCityCrime X = new GetCityCrime();
        IList<CityCrime> stats = X.getStats();

        foreach (CityCrime c in stats)
        {
            Console.WriteLine(c.City + " : " + c.AggravatedAssault);
        }
        Console.ReadLine();
    }
}

public class GetCityCrime
{
    Uri serviceUri;
    datagovCrimesContainer context;

    public GetCityCrime()
    {
        serviceUri =
            new Uri("https://api.datamarket.azure.com/Data.ashx/data.gov/Crimes");
        context = new datagovCrimesContainer(serviceUri);
        context.Credentials = new NetworkCredential(" ", "my account key as copied from DataMarket");
    }

    public IList<CityCrime> getStats()
    {
        IEnumerable<CityCrime> query;

        query = from c in context.CityCrime
                where c.State == "Alaska"
                select c;

        return query.ToList();
    }
}
```

If the dataset I'm using exposes entity sets for query, I can use Add Service Reference in Visual Studio, just as I do for a Windows Communication Foundation (WCF) service. I simply right-click on the project and choose Add Service Reference. In the Add Service Reference dialog box, I enter the URI for the entry point of the service (I can get this from the Service Explorer page in DataMarket) as the Address and then click OK.

If the dataset I'm using doesn't allow for ad hoc query, however, I can't use Add Service Reference. In this case, I can code my own POCO classes representing the entities (or objects) returned from the fixed Web methods that I call. DataMarket uses Basic Authentication, passing in a Windows Live ID as the username and an associated DataMarket account key in the password field. For the purpose of the console application example (see **Figure 2**), I used Add Service Reference in Visual Studio to generate the classes. Visual Studio will access the service and generate the associated classes based on the data service definition, and add them to the project. After my classes have been generated, I continue to develop my application in the same way I would any application that consumes an OData service, using either basic authentication or Audit Collection Services to authenticate with the service. The **Figure 2** example sets up a simple console application using Basic Authentication.

*Note: To run the sample code in **Figure 2**, you'll need to register as a DataMarket customer and subscribe to the Data.Gov Crime Statistics offering. Also note that the URI used in this sample refers to a pre-release to manufacturing version of the service; you can find the correct URI by logging into DataMarket, going to My Datasets and accessing your subscription to the Data.gov Crime Statistics offering.*

For more details on writing applications that consume data services, check out "Entity Framework 4.0 and WCF Data Services 4.0 in Visual Studio 2010" (msdn.microsoft.com/magazine/ee336128) and "Expose and Consume Data in a Web Services World" (msdn.microsoft.com/magazine/cc748663).

Selling Data on DataMarket

Many popular apps and Web sites generate, store and consume large amounts of valuable data. But typically that data is only leveraged within the app for which it was created. With the introduction of WCF Data Services and OData, we had a simple way for developers to expose their data for broader use, offering a data services platform and making it easier for that data to be used not only within the initially intended application, but within third-party apps. Now, with DataMarket, there's a simple opportunity for developers to not only expose that data to apps they build, but also to generate profit by selling data they're already required to store and maintain.

DataMarket is built on Windows Azure and SQL Azure and allows publishers to create datasets for data that they host in SQL Azure. If my data is already stored in SQL Azure, I'm well on my way to being able to offer data for sale in DataMarket. To learn more about how you can become a DataMarket publisher, check out blogs.msdn.com/dallas.

ELISA FLASKO is a program manager in the Windows Azure Marketplace DataMarket team at Microsoft. She can be reached at blogs.msdn.com/elisaj.

THANKS to the following technical experts for reviewing this article:
Rene Bouw, Moe Khosravy and Adam Wilson

version 17

LEADTOOLS®



The World Leader in Imaging Development SDKs

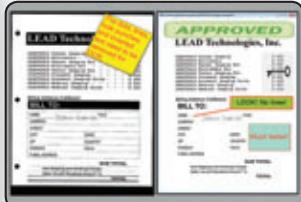
Silverlight, .NET, WPF, WCF, WF, C API, C++ Class Lib, COM & more!

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

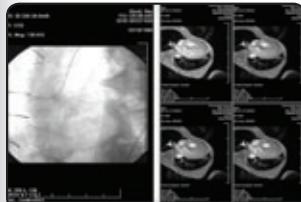
LEADTOOLS provides developers easy access to decades of expertise in color, grayscale, document, medical, vector and multimedia imaging development. Install LEADTOOLS to eliminate months of research and programming time while maintaining high levels of quality, performance and functionality.

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

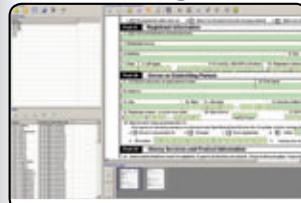
Document



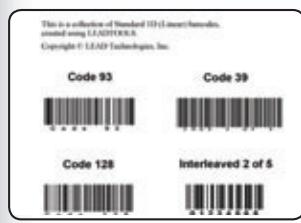
DICOM Medical



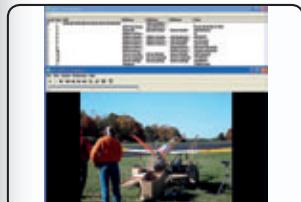
Form Recognition & Processing



Barcode



Multimedia



Vector



Free 60 Day Evaluation!

www.leadtools.com/msdn

(800) 637-1840

Getting Started with SQL Azure Development

Lynn Langit

Microsoft Windows Azure offers several choices for data storage. These include Windows Azure storage and SQL Azure. You may choose to use one or both in your particular project. Windows Azure storage currently contains three types of storage structures: tables, queues and blobs.

SQL Azure is a relational data storage service in the cloud. Some of the benefits of this offering are the ability to use a familiar relational development model that includes much of the standard SQL Server language (T-SQL), tools and utilities. Of course, working with well-understood relational structures in the cloud, such as tables, views and stored procedures, also results in increased developer productivity when working in this new platform. Other benefits include a reduced need for physical database-administration tasks to perform server setup, maintenance and security, as well as built-in support for reliability, high availability and scalability.

This article discusses prerelease features of SQL Azure and other tools. All information is subject to change.

This article discusses:

- Setting up databases
- Creating an application
- Supported database features
- Data-migration tools

Technologies discussed:

Windows Azure, SQL Azure, SQL Server, Visual Studio 2010

I won't cover Windows Azure storage or make a comparison between the two storage modes here. You can read more about these storage options in Julie Lerman's July 2010 Data Points column (msdn.microsoft.com/magazine/ff796231). It's important to note that Windows Azure tables are not relational tables. The focus of this is on understanding the capabilities included in SQL Azure.

This article will explain the differences between SQL Server and SQL Azure. You need to understand the differences in detail so that you can appropriately leverage your current knowledge of SQL Server as you work on projects that use SQL Azure as a data source.

If you're new to cloud computing you'll want to do some background reading on Windows Azure before continuing with this article. A good place to start is the MSDN Developer Cloud Center at msdn.microsoft.com/ff380142.

Getting Started with SQL Azure

To start working with SQL Azure, you'll first need to set up an account. If you're an MSDN subscriber, then you can use up to three SQL Azure databases (maximum size 1GB each) for up to 16 months (details at msdn.microsoft.com/subscriptions/ee461076) as a developer sandbox. To sign up for a regular SQL Azure account (storage and data transfer fees apply) go to microsoft.com/windowsazure/offers/.

After you've signed up for your SQL Azure account, the simplest way to initially access it is via the Web portal at sql.azure.com. You must sign in with the Windows Live ID that you've associated to your Windows Azure account. After you sign in, you can create your server installation and get started developing your application.

An example of the SQL Azure Web management portal is shown in **Figure 1**. Here you can see a server and its associated databases. You'll notice that there's also a tab on the Web portal for managing the Firewall Settings for your particular SQL Azure installation.

As you initially create your SQL Azure server installation, it will be assigned a random string for the server name. You'll generally also set the administrator username, password, geographic server location and firewall rules at the time of server creation. You can select the location for your SQL Azure installation at the time of server creation. You will be presented with a list of locations (datacenters) from which to choose. If your application front end is built in Windows Azure, you have the option to locate both that installation and your SQL Azure installation in the same geographic location by associating the two installations.

By default there's no access to your server, so you'll have to create firewall rules for all client IPs. SQL Azure uses port 1433, so make sure that port is open for your client application as well. When connecting to SQL Azure you'll use the *username@servername* format for your username. SQL Azure supports SQL Server Authentication only; Windows Authentication is not supported. Multiple Active Result Set (MARS) connections are supported.

Open connections will time out after 30 minutes of inactivity. Also, connections can be dropped for long-running queries and transactions or excessive resource usage. Development best practices in your applications around connections are to open, use and then close those connections manually, to include retry connection logic for dropped connections and to avoid caching connections because of these behaviors. For more details about supported client protocols for SQL Azure, see Steve Hale's blog post at blogs.msdn.com/sqlnativeclient/archive/2010/02/12/using-sql-server-client-apis-with-sql-azure-version-1-0.aspx.

Another best practice is to encrypt your connection string to prevent man-in-the-middle attacks.

You'll be connected to the master database by default if you don't specify a database name in the connection string. In SQL Azure the T-SQL statement USE is not supported for changing databases, so you'll generally specify the database you want to connect to in the connection string (assuming you want to connect to a database other than master). Here's an example of an ADO.NET connection:

```
Server=tcp:server.ctp.database.windows.net;
Database=<databasename>;
User ID=user@server;
Password=password;
Trusted_Connection=False;
Encrypt=true;
```

Setting up Databases

After you've successfully connected to your installation you'll want to create one or more databases. Although you can create databases using the SQL Azure portal, you may prefer to do so using some of the other tools, such as SQL Server Management Studio 2008 R2. By default, you can create up to 149 databases for each SQL Azure server installation. If you need more databases than that, you must call the Windows Azure business desk to have this limit increased.

When creating a database you must select the maximum size. The current options for sizing (and billing) are Web or Business Edition. Web Edition, the default, supports databases of 1GB or 5GB total. Business Edition supports databases of up to 50GB, sized in increments of 10GB—in other words, 10GB, 20GB, 30GB, 40GB and 50GB.

You set the size limit for your database when you create it by using the MAXSIZE keyword. You can change the size limit or the edition (Web or Business) after the initial creation using the ALTER DATABASE statement. If you reach your size or capacity limit for the edition you've selected, then you'll see the error code 40544. The database size measurement doesn't include the master database, or any database logs. For more details about sizing and pricing, see microsoft.com/windowsazure/pricing/#sql.

It's important to realize that when you create a new SQL Azure database, you're actually creating three replicas of that database. This is done to ensure high availability. These replicas are completely transparent to you. The new database appears as a single unit for your purposes.

Once you've created a database, you can quickly get the connection string information for it by selecting the database in the list on the portal and then clicking the Connection Strings button. You can also quickly test connectivity via the portal by clicking the Test Connectivity button for the selected database. For this test to succeed you must enable the Allow Microsoft Services to Connect to this Server option on the Firewall Rules tab of the SQL Azure portal.

Creating Your Application

After you've set up your account, created your server, created at least one database and set a firewall rule so that you can connect to the database, you can start developing your application using this data source.

Unlike Windows Azure data storage options such as tables, queues or blobs, when you're using SQL Azure as a data source for your project, there's nothing to install in your development environment. If you're using Visual Studio 2010, you can just get

The screenshot shows the Windows Azure Services management portal. The top navigation bar includes links for Windows Azure, SQL Azure, AppFabric, and Marketplace, along with Billing and a search bar. The main content area has a green header bar with 'Summary' and 'Help and Resources'. Below this is a 'Server Administration' section with 'Server Information' fields: Server Name (database.windows.net), Administrator Username (not specified), and Server Location (North Central US). There are buttons for 'Reset Password' and 'Drop Server'. At the bottom, there are tabs for 'Databases' and 'Firewall Settings', with a table of database details. The table columns are Database Name, Size, Max Size, and Edition. The data shows several databases: Adven...ksDWAZZ2008R2 (86.7 MB, 1 GB, Web), Adven...ksLTAZZ2008R2 (2.7 MB, 1 GB, Web), DatingGame (152 KB, 1 GB, Web), master (256 KB, 1 GB, Web), TestForLynn (0 B, 1 GB, Web), and WednesdayTest (0 B, 1 GB, Web).

Database Name	Size	Max Size	Edition
Adven...ksDWAZZ2008R2	86.7 MB	1 GB	Web
Adven...ksLTAZZ2008R2	2.7 MB	1 GB	Web
DatingGame	152 KB	1 GB	Web
master	256 KB	1 GB	Web
TestForLynn	0 B	1 GB	Web
WednesdayTest	0 B	1 GB	Web

Figure 1 Summary Information for a SQL Azure Database

started—no additional SDKs, tools or anything else are needed.

Although many developers will choose to use a Windows Azure front end with a SQL Azure backend, this configuration is not required. You can use any front-end client with a supported connection library such as ADO.NET or ODBC. This could include, for example, an application written in Java or PHP. Connecting to SQL Azure via OLE DB is currently not supported.

If you're using Visual Studio 2010 to develop your application, you can take advantage of the included ability to view or create many types of objects in your selected SQL Azure database installation directly from the Visual Studio Server Explorer. These objects are Tables, Views, Stored Procedures, Functions and Synonyms. You can also see the data associated with these objects using this viewer. For many developers, using Visual Studio 2010 as the primary tool to view and manage SQL Azure data will be sufficient.

The Server Explorer View window

is shown in **Figure 2**. Both a local installation of a database and a cloud-based instance are shown. You'll see that the tree nodes differ slightly in the two views. For example, there's no Assemblies node in the cloud installation because custom assemblies are not supported in SQL Azure.

As I mentioned earlier, another tool you may want to use to work with SQL Azure is SQL Server Management Studio (SSMS) 2008 R2. With SSMS 2008 R2, you actually have access to a fuller set of operations for SQL Azure databases than in Visual Studio 2010. I find that I use both tools, depending on which operation I'm trying to complete. An example of an operation available in SSMS 2008 R2 (and not in Visual Studio 2010) is creating a new database using a T-SQL script. Another example is the ability to easily perform index operations (create, maintain, delete and so on). An example is shown in **Figure 3**.

Newly released in SQL Server 2008 R2 is a data-tier application, or DAC. DAC pacs are objects that combine SQL Server or SQL Azure database schemas and objects into a single entity. You can use either Visual Studio 2010 (to build) or SQL Server 2008 R2 SSMS (to extract) to create a DAC from an existing database.

If you wish to use Visual Studio 2010 to work with a DAC, then you'd start by selecting the SQL Server Data-Tier Application project type in Visual Studio 2010. Then, on the Solution Explorer, right-click your project name and click Import Data-Tier Application. A wizard opens to guide you through the import process. If you're using SSMS, start by right-clicking on the database you want to use in the Object Explorer, click Tasks, then click Extract Data-Tier Application to create the DAC.

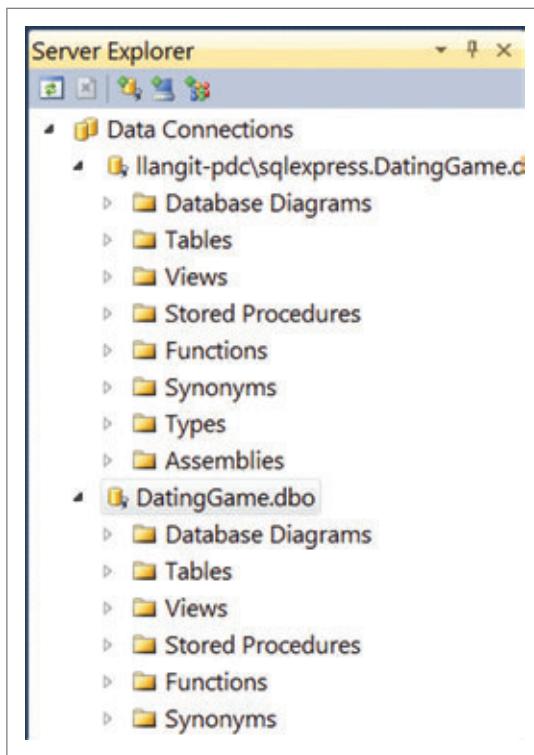


Figure 2 Viewing Data Connections in Visual Studio Server Explorer

The generated DAC is a compressed file that contains multiple T-SQL and XML files. You can work with the contents by right-clicking the .dacpac file and then clicking Unpack. SQL Azure supports deleting, deploying, extracting and registering DAC pacs, but does *not* support upgrading them.

Another tool you can use to connect to SQL Azure is the latest community technology preview (CTP) release of the tool code-named "Houston." Houston is a zero-install, Silverlight-based management tool for SQL Azure installations. When you connect to a SQL Azure installation using Houston, you specify the datacenter location (as of this writing North Central U.S., South Central U.S., North Europe, Central Europe, Asia Pacific or Southeast Asia).

Houston is in early beta and the current release (shown in **Figure 4**) looks somewhat like SSMS. Houston supports working with Tables, Views, Queries and Stored Procedures in a SQL Azure database installation. You can access

Houston from the SQL Azure Labs site at sqlazurelabs.com/houston.aspx.

Another tool you can use to connect to a SQL Azure database is SQLCMD (msdn.microsoft.com/library/ee336280). Even though SQLCMD is supported, the OSQl command-line tool is not supported by SQL Azure.

Using SQL Azure

So now you've connected to your SQL Azure installation and have created a new, empty database. What exactly can you do with SQL Azure? Specifically, you may be wondering what the limits are on creating objects. And after those objects have been created, how do you populate those objects with data?

Houston is a zero-install, Silverlight-based management tool for SQL Azure installations.

As I mentioned at the beginning of this article, SQL Azure provides relational cloud data storage, but it does have some subtle feature differences to an on-premises SQL Server installation. Starting with object creation, let's look at some of the key differences between the two.

You can create the most commonly used objects in your SQL Azure database using familiar methods. The most commonly used relational objects (which include tables, views, stored procedures,

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



ACTIVE REPORTS



ACTIVE REPORTS

SPREAD

DATA DYNAMICS REPORTS

ACTIVE ANALYSIS

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

indices and functions) are all available. There are some differences around object creation, though. Here's a summary of those differences:

- SQL Azure tables must contain a clustered index. Non-clustered indices can be subsequently created on selected tables. You can create spatial indices, but you cannot create XML indices.
- Heap tables are not supported.
- CLR geo-spatial types (such as Geography and Geometry) are supported, as is the HierarchyID data type. Other CLR types are not supported.
- View creation must be the first statement in a batch. Also, view (or stored procedure) creation with encryption is not supported.
- Functions can be scalar, inline or multi-statement table-valued functions, but cannot be any type of CLR function.

There's a complete reference of partially supported T-SQL statements for SQL Azure on MSDN at msdn.microsoft.com/library/ee336267.

Before you get started creating your objects, remember that you'll connect to the master database if you don't specify a different one in your connection string. In SQL Azure, the USE (database) statement is not supported for changing databases, so if you need to connect to a database other than the master database, then you must explicitly specify that database in your connection string, as shown earlier.

Data Migration and Loading

If you plan to create SQL Azure objects using an existing, on-premises database as your source data and structures, then you can simply use SSMS to script an appropriate DDL to create those objects on SQL Azure. Use the Generate Scripts Wizard and set the "Script for the database engine type" option to "for SQL Azure."

An even easier way to generate a script is to use the SQL Azure Migration Wizard, available as a download from CodePlex at sqlazuremw.codeplex.com. With this handy tool you can generate a script to create the objects and can also load the data via bulk copy using bcp.exe.

You could also design a SQL Server Integration Services (SSIS) package to extract and run a DDM or DDL script. If you're using SSIS, you'd most commonly design a package that extracts the DDL from the source database, scripts that DDL for SQL Azure and then executes that script on one or more SQL Azure installations. You might also choose to load the associated data as part of the package's execution path. For more information about working with SSIS, see msdn.microsoft.com/library/ms141026.

Also of note regarding DDL creation and data migration is the CTP release of SQL Azure Data Sync Services (sqlazurelabs.com). You can see this service in action in a Channel 9 video, "Using SQL Azure Data Sync Service to provide Geo-Replication of SQL Azure Databases," at tinyurl.com/2we4d6q. Currently, SQL Azure Data Sync services works via Synchronization Groups (HUB and MEMBER servers) and then via scheduled synchronization at the level of individual tables in the databases selected for synchronization.

You can use the Microsoft Sync Framework Power Pack for SQL Azure to synchronize data between a data source and a SQL Azure installation. As of this writing, this tool is in CTP release and is available from tinyurl.com/2ecjwku. If you use this framework to

perform subsequent or ongoing data synchronization for your application, you may also wish to download the associated SDK.

What if your source database is larger than the maximum size for the SQL Azure database installation? This could be greater than the absolute maximum of 50GB for the Business Edition or some smaller limit based on the other program options.

Currently, customers must partition (or shard) their data manually if their database size exceeds the program limits. Microsoft has announced that it will be providing an auto-partitioning utility for SQL Azure in the future. In the meantime, it's important to note that T-SQL table partitioning is not supported in SQL Azure. There's a free utility called Enzo SQL Shard (enzosqlshard.codeplex.com) that you can use for partitioning your data source.

You'll want to take note of some other differences between SQL Server and SQL Azure regarding data loading and data access.

Added recently is the ability to copy a SQL Azure database via the Database copy command. The syntax for a cross-server copy is as follows:

```
CREATE DATABASE DB2A AS COPY OF Server1.DB1A
```

The T-SQL INSERT statement is supported (with the exceptions of updating with views or providing a locking hint inside of an INSERT statement).

Related further to data migration, T-SQL DROP DATABASE and other DDL commands have additional limits when executed against a SQL Azure installation. In addition, the T-SQL RESTORE

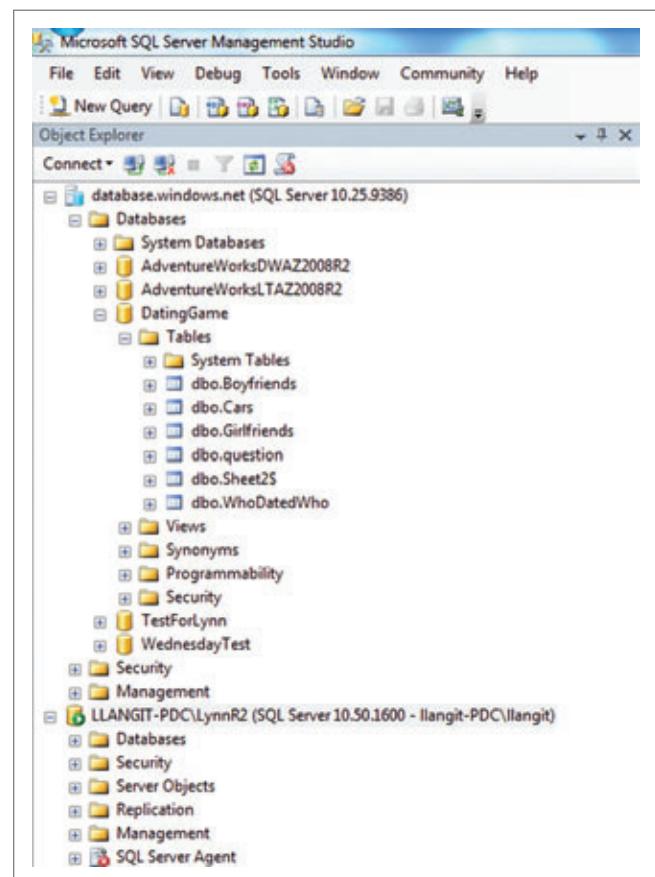


Figure 3 Using SQL Server Management Studio 2008 R2 to Manage SQL Azure

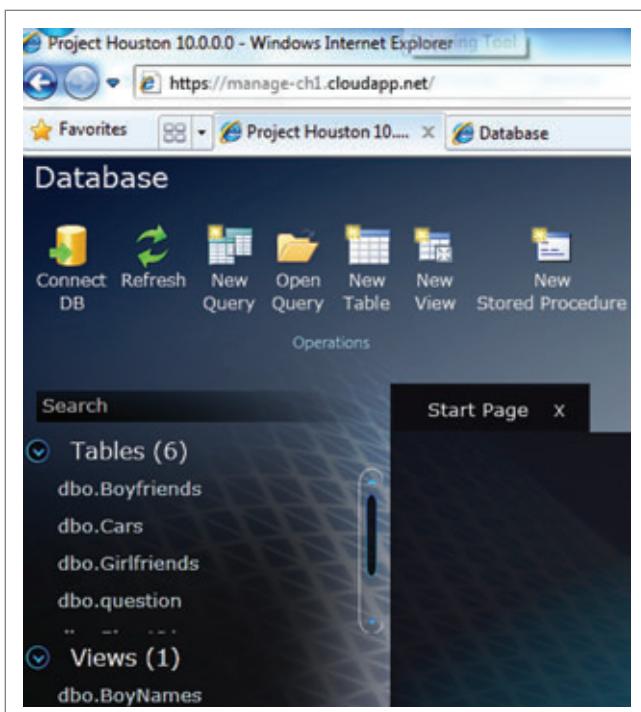


Figure 4 Using Houston to Manage SQL Azure

and ATTACH DATABASE commands are not supported. Finally, the T-SQL statement EXECUTE AS (login) is not supported.

Data Access and Programmability

Now let's take a look at common programming concerns when working with cloud data.

First, you'll want to consider where to set up your development environment. If you're an MSDN subscriber and can work with a database that's less than 1GB, then it may well make sense to develop using only a cloud installation (sandbox). In this way there will be no issue with migration from local to cloud. Using a regular (non-MSDN subscriber) SQL Azure account, you could develop directly against your cloud instance (most probably using a cloud-located copy of your production database). Of course, developing directly from the cloud is not practical for all situations.

If you choose to work with an on-premises SQL Server database as your development data source, then you must develop a mechanism for synchronizing your local installation with the cloud installation. You could do that using any of the methods discussed earlier, and tools like Data Sync Services and Sync Framework are being developed with this scenario in mind.

As long as you use only the supported features, the method for having your application switch from an on-premises SQL Server installation to a SQL Azure database is simple—you need only to change the connection string in your application.

Regardless of whether you set up your development installation locally or in the cloud, you'll need to understand some programmability differences between SQL Server and SQL Azure. I've already covered the T-SQL and connection string differences. In addition, all tables must have a clustered index at minimum (heap tables are not supported).

As previously mentioned, the USE statement for changing databases isn't supported. This also means that there's no support for distributed (cross-database) transactions or queries, and linked servers are not supported.

Other options *not* available when working with a SQL Azure database include:

- Full-text indexing
- CLR custom types (however, the built-in Geometry and Geography CLR types are supported)
- RowGUIDs (use the uniqueidentifier type with the NEWID function instead)
- XML column indices
- Filestream datatype
- Sparse columns

Default collation is always used for the database. To make collation adjustments, set the column-level collation to the desired value using the T-SQL COLLATE statement.

And finally, you cannot currently use SQL Profiler or the Database Tuning Wizard on your SQL Azure database.

Some important tools that you can use with SQL Azure for tuning and monitoring include:

- SSMS Query Optimizer to view estimated or actual query execution plan details and client statistics
- Select Dynamic Management views to monitor health and status
- Entity Framework to connect to SQL Azure after the initial model and mapping files have been created by connecting to a local copy of your SQL Azure database.

Depending on what type of application you're developing, you may be using SSAS, SSRS, SSIS or PowerPivot. You can also use any of these products as consumers of SQL Azure database data. Simply connect to your SQL Azure server and selected database using the methods already described in this article.

It's also important to fully understand the behavior of transactions in SQL Azure. As mentioned, only local (within the same database)

Status History							
Service [Sub-Region]	Aug 31	Aug 30	Aug 29	Aug 28	Aug 27	Aug 26	Aug 25
SQL Azure Database [East Asia]	Normal service availability						
SQL Azure Database [North Central US]	Normal service availability						
SQL Azure Database [North Europe]	Normal service availability						
SQL Azure Database [South Central US]	Normal service availability						
SQL Azure Database [Southeast Asia]	Normal service availability						
SQL Azure Database [West Europe]	Normal service availability						

Page Last Updated: 7 Sep 2010 4:51 PM UTC

Normal service availability Performance degradation Service interruption Additional information

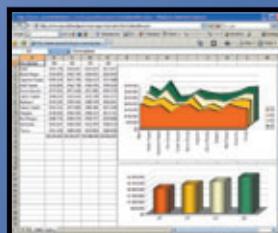
Figure 5 SQL Azure Status History

In a League by Itself

"SpreadsheetGear 2010 is Fantastic! These new capabilities just propelled this control way-way past any competition, of which you have none IMHO. SpreadsheetGear is in a league all by itself."

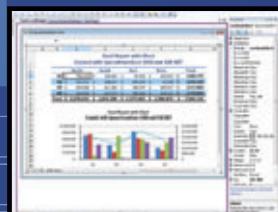
Greg Newman, Senior Software Engineer, WSFS Cash Connect

SpreadsheetGear 2010



ASP.NET / Excel / Reporting

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



Excel Compatible Windows Forms Control

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



Create Dashboards from Excel Charts and Ranges

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

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

www.SpreadsheetGear.com.

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

 **SpreadsheetGear**

The screenshot shows a SQL Server Management Studio window titled "SQLQuery1.sql - (269)†". A T-SQL query is run against the sys.bandwidth_usage system view:

```
SELECT * FROM sys.bandwidth_usage
WHERE database_name = 'Dating Game' OR database_name = 'TestForLynn'
ORDER BY quantity DESC
```

The results grid displays the following data:

	time	database_name	direction	class	time_period	quantity
1	2010-07-28 05:00:00.000	TESTFORLYNN	Ingress	External	OffPeak	442
2	2010-07-28 05:00:00.000	TESTFORLYNN	Egress	External	OffPeak	145
3	2010-07-28 04:00:00.000	TestForLynn	Egress	External	OffPeak	85
4	2010-07-28 04:00:00.000	TestForLynn	Ingress	External	OffPeak	47
5	2010-07-28 06:00:00.000	TESTFORLYNN	Egress	External	OffPeak	27
6	2010-07-28 06:00:00.000	TESTFORLYNN	Ingress	External	OffPeak	12
7	2010-07-27 04:00:00.000	TestForLynn	Egress	InterSubregion	OffPeak	5
8	2010-07-27 04:00:00.000	TestForLynn	Ingress	InterSubregion	OffPeak	1

Figure 6 Bandwidth Usage in SQL Query

transactions are supported. In addition, the only transaction-isolation level available for a database hosted on SQL Azure is READ COMMITTED SNAPSHOT. Using this isolation level, readers get the latest consistent version of data that was available when the statement STARTED.

SQL Azure doesn't detect update conflicts. This is also called an optimistic concurrency model, because lost updates, non-repeatable reads and phantoms can occur. Of course, dirty reads cannot occur.

There are no physical servers to buy, install, patch, maintain or secure.

Database Administration

Generally, when using SQL Azure, the administrator role becomes one of logical installation management. Physical management is handled by the platform. From a practical standpoint this means there are no physical servers to buy, install, patch, maintain or secure. There's no ability to physically place files, logs, tempdb and so on in specific physical locations. Because of this, there's no support for the T-SQL commands USE <database>, FILEGROUP, BACKUP, RESTORE or SNAPSHOT.

There's no support for the SQL Agent on SQL Azure. Also, there is no ability (or need) to configure replication, log shipping, database mirroring or clustering. If you need to maintain a local, synchronized copy of SQL Azure schemas and data, then you can use any of the tools discussed earlier for data migration and synchronization—they work both ways. You can also use the DATABASE COPY command.

Other than keeping data synchronized, what are some other tasks that administrators may need to perform on a SQL Azure installation?

Most commonly, there will still be a need to perform logical administration. This includes tasks related to security and performance management. Additionally, you may be involved in monitoring for capacity usage and associated costs. To help you with these tasks, SQL Azure provides a public Status History dashboard that shows current service status and recent history (an example of history is shown in Figure 5) at microsoft.com/windowsazure/support/status/servicedashboard.aspx.

SQL Azure provides a high-security bar by default. It forces SSL encryption with all permitted (via firewall rules) client connections. Server-level logins and database-level users and roles are also secured. There are no server-level roles in SQL Azure. Encrypting the connection string is a best practice. Also, you may want to use Windows Azure certificates for additional security. For more details, see blogs.msdn.com/b/sqlazure/archive/2010/09/07/10058942.aspx.

In the area of performance, SQL Azure includes features such as automatically killing long-running transactions and idle connections (more than 30 minutes).

Although you can't use SQL Profiler or trace flags for performance tuning, you can use

SQL Query Optimizer to view query execution plans and client statistics. You can also perform statistics management and index tuning using the standard T-SQL methods.

There's a select list of dynamic management views (covering database, execution or transaction information) available for database administration as well. These include sys.dm_exec_connections, _requests, _sessions, _tran_database_transactions, _active_transactions and _partition_stats. For a complete list of supported dynamic management views for SQL Azure, see msdn.microsoft.com/library/ee336238.aspx#dmv.

There are also some new views such as sys.database_usage and sys.bandwidth_usage. These show the number, type and size of the databases and the bandwidth usage for each database so that administrators can understand SQL Azure billing. A sample is shown in Figure 6. In this view, quantity is listed in KB. You can monitor space used via this command:

```
SELECT SUM(reserved_page_count) * 8192
FROM sys.dm_db_partition_stats
```

You can also access the current charges for the SQL Azure installation via the SQL Azure portal by clicking on the Billing link at the top-right corner of the screen.

Learn More

To learn more about SQL Azure, I suggest you download the Windows Azure Training Kit. This includes SQL Azure hands-on learning, white papers, videos and more. The training kit is available from microsoft.com/downloads/details.aspx?FamilyID=413E88F8-5966-4A83-B309-53B7B77EDF78.

Also, you'll want to read the SQL Azure Team Blog at blogs.msdn.com/b/sqlazure/ and check out the MSDN SQL Azure Developer Center at msdn.microsoft.com/windowsazure/sqlazure.

If you want to continue to preview upcoming features for SQL Azure, be sure to visit SQL Azure Labs at sqlazurelabs.com. ■

Lynn Langit is a developer evangelist for Microsoft in Southern California. She's published two books on SQL Server Business Intelligence and has created a set of courseware to introduce children to programming at TeachingKidsProgramming.org. Read her blog at blogs.msdn.com/SoCalDevGal.

THANKS to the following technical experts for reviewing this article:
George Huey and David Robinson

JUST HOW EPIC

IS WINDWARD REPORTS'
EARTH-SHATTERING
APPROACH TO CUSTOM
ENTERPRISE REPORTING?



It's so powerful it makes the Kraken look like a gimp sea monkey.

It's so fast it makes warp drive look like a hobbit running backwards.

IT'S SO EASY IT MIGHT AS WELL BE "GOD MODE"

It makes other reporting solutions seem like you're trying to crack **RIJNDAEL ENCRYPTION**, or like driving the "**ROAD OF DEATH**" in the Bolivian Andes, backwards, while blindfolded.

No other solution can match Windward's array of features, **STREAMLINED IMPLEMENTATION**, and already familiar interface. You create custom report templates with Word, Excel, or PowerPoint. Even your co-workers **who need IT to turn on their computers** can use Microsoft Office to format reports.

Windward Reports
hasn't simply
shifted the
reporting paradigm,

**IT HAS NINJA
ROUND HOUSE
KICKED IT
ALL THE WAY
I N T O A N
ENTIRELY
DIFFERENT
DIMENSION.**



IF YOU FIND ALL OF THIS HARD TO SWALLOW,
DOWNLOAD THE FREE TRIAL AT
www.WindwardReports.com/msdn.aspx
AND SEE FOR YOURSELF. **

- Design Reports in Microsoft Word, Excel, and PowerPoint.
- Drag N'Drop data into report templates *no coding required!*
- Solutions for .Net, Java and SharePoint platforms
- Integrates easily into any software application

the ONLY EPIC REPORTING & DOCUMENT GENERATING SOLUTION



Unless of course you enjoy designing report templates with endless code, apologies for keeping you from your current mind-numbingly dull solution.

(303) 499-2544

Studio Enterprise has 8,310,799 lines of code and would cost
\$710,461.541
and take **10 Years** to develop*

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

Do you develop for the web?
Our ASP.NET
AJAX &
Silverlight controls
would cost

\$159,009,712

Only build desktop apps?
Our
WinForms
& WPF controls
would cost

\$233,189,875

Do you have mobile versions of your ASP.NET apps?
Building
our Mobile Web
controls would
cost
\$3,217,145

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



DESKTOP WinForms & WPF



 ComponentOne®

Welcome to

DEVTOPIA

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

275
CONTROLS in
Studio Enterprise



Get developing right away.

2,791
WinForms

Code snippets per Studio
847 Silverlight

1,340 ActiveX

706 ASP.NET
AJAX

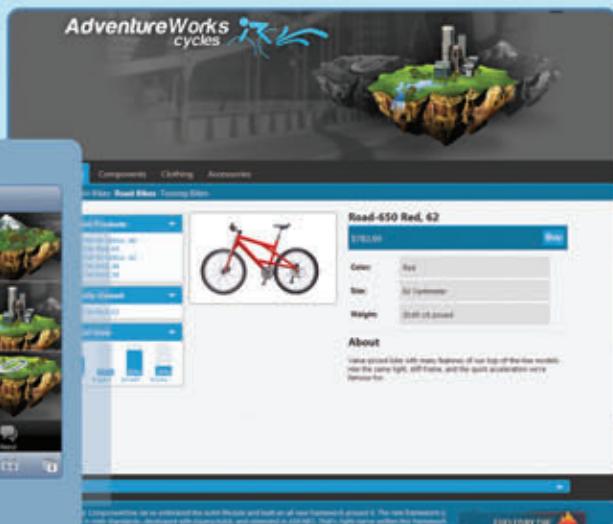
513 WPF

691 iPhone
& Mobile

Studio Enterprise Code Snippets
6,888



Check out the demo with source code



STUDIO
ENTERPRISE 2010 v2 New!

DOWNLOAD TODAY @ COMPONENTONE.COM/DEVTOPIA

Synchronizing Multiple Nodes in Windows Azure

Josh Twist

The cloud represents a major technology shift, and many industry experts predict this change is of a scale we see only every 12 years or so. This level of excitement is hardly surprising when you consider the many benefits the cloud promises: significantly reduced running costs, high availability and almost infinite scalability, to name but a few.

Of course, such a shift also presents the industry with a number of challenges, not least those faced by today's developers. For example, how do we build systems that are optimally positioned to take advantage of the unique features of the cloud?

Fortunately, Microsoft in February launched the Windows Azure Platform, which contains a number of right-sized pieces to support the creation of applications that can support enormous numbers of users while remaining highly available. However, for any

application to achieve its full potential when deployed to the cloud, the onus is on the developers of the system to take advantage of what is arguably the cloud's greatest feature: elasticity.

Elasticity is a property of cloud platforms that allows additional resources (computing power, storage and so on) to be provisioned on-demand, providing the capability to add additional servers to your Web farm in a matter of minutes, not months. Equally important is the ability to remove these resources just as quickly.

A key tenet of cloud computing is the pay-as-you-go business model, where you only pay for what you use. With Windows Azure, you only pay for the time a node (a Web or Worker Role running in a virtual machine) is deployed, thereby reducing the number of nodes when they're no longer required or during the quieter periods of your business, which results in a direct cost savings.

Therefore, it's critically important that developers create elastic systems that react automatically to the provision of additional hardware, with minimum input or configuration required from systems administrators.

Scenario 1: Creating Order Numbers

Recently, I was lucky enough to work on a proof of concept that looked at moving an existing Web application infrastructure into the cloud using Windows Azure.

Given the partitioned nature of the application's data, it was a prime candidate for Windows Azure Table Storage. This simple but high-performance storage mechanism—with its support for

This article discusses:

- Elasticity in the cloud
- Moving an existing Web app to Windows Azure
- Creating unique IDs
- Polling versus listening to determine released Worker Roles

Technologies discussed:

Windows Azure

Code download available at:

code.msdn.microsoft.com/mag201011Sync

almost infinite scalability—was an ideal choice, with just one notable drawback concerning unique identifiers.

The target application allowed customers to place an order and retrieve their order number. Using SQL Server or SQL Azure, it would've been easy to generate a simple, numeric, unique identifier, but Windows Azure Table Storage doesn't offer auto-incrementing primary keys. Instead, developers using Windows Azure Table Storage might create a GUID and use this as the "key" in the table:

505EAB78-6976-4721-97E4-314C76A8E47E

The problem with using GUIDs is that they're difficult for humans to work with. Imagine having to read your GUID order number out to an operator over the telephone—or make a note of it in your diary. Of course, GUIDs have to be unique in every context simultaneously, so they're quite complex. The order number, on the other hand, only has to be unique in the Orders table.

Creating a Simple Unique ID in Windows Azure

A number of relatively simple solutions to the GUID problem were considered:

1. Use SQL Azure to Generate Unique IDs

For a number of reasons, the proof of concept had already discounted SQL Azure in favor of Windows Azure Table Storage—primarily due to the need for the system to scale out to many nodes, each with many threads executing against the data.

2. Use Blob Storage to Manage an Incrementing Value

Store a central counter in Windows Azure Blob Storage. Nodes could read and update the order number,

Figure 1 Allocating a Range of Order Numbers to Each Node to Ensure Unique IDs

Node	Range
A	0-1,000,000
B	1,000,001-2,000,000

providing a simple sequential order number generation mechanism for use by multiple nodes. However, the contention at this point for a busy system requiring many new order numbers per second would likely impede the scalability of the system.

3. Partition Unique IDs Across Each Node Create a lightweight in-memory counter that generates unique order numbers. In order to ensure uniqueness across all nodes, each node would be allocated a range of order numbers, as shown in **Figure 1**.

However, this approach raises a number of questions. What happens when a node exhausts a range? What happens when hundreds of nodes are added to the system at one time? What if a node crashes and is replaced by the Windows Azure runtime with a fresh node? Administrators would need to closely monitor these ranges and be careful to ensure the configuration is correct or face data corruption.

Instead, a much more elegant approach was needed—a solution that required no per-node configuration, demonstrated little contention and guaranteed uniqueness at all times. To achieve this, I created a hybrid of the second and third options.

The concept was relatively straightforward: use a small text file in blob storage to store the last order number. When a new order number is required, a node can access this blob, increment the value and write it back to storage. Of course, there's a reasonable chance that another node will have accessed the blob with the same intention during this read-increment-write process. Without some kind of concurrency management, the order numbers wouldn't be

Figure 2 The Full UniqueIdGenerator Class

```
public class UniqueIdGenerator
{
    private readonly object _padLock = new object();
    private Int64 _lastId;
    private Int64 _upperLimit;
    private readonly int _rangeSize;
    private readonly int _maxRetries;
    private readonly IOptimisticSyncStore _optimisticSyncStore;

    public UniqueIdGenerator(
        IOptimisticSyncStore optimisticSyncStore,
        int rangeSize = 1000,
        int maxRetries = 25)
    {
        _rangeSize = rangeSize;
        _maxRetries = maxRetries;
        _optimisticSyncStore = optimisticSyncStore;
        UpdateFromSyncStore();
    }

    public Int64 NextId()
    {
        lock (_padLock)
        {
            if (_lastId == _upperLimit)
            {
                UpdateFromSyncStore();
            }
            return _lastId++;
        }
    }

    private void UpdateFromSyncStore()
    {
        int retryCount = 0;
        // maxRetries + 1 because the first run isn't a 're'try.
        while (retryCount < _maxRetries + 1)
        {
            string data = _optimisticSyncStore.GetData();

            if (!Int64.TryParse(data, out _lastId))
            {
                throw new Exception(string.Format(
                    "Data '{0}' in storage was corrupt and " +
                    "could not be parsed as an Int64", data));
            }

            _upperLimit = _lastId + _rangeSize;

            if (_optimisticSyncStore.TryOptimisticWrite(
                _upperLimit.ToString()))
            {
                return;
            }

            retryCount++;
            // update failed, go back around the loop
        }

        throw new Exception(string.Format(
            "Failed to update the OptimisticSyncStore after {0} attempts",
            retryCount));
    }
}
```

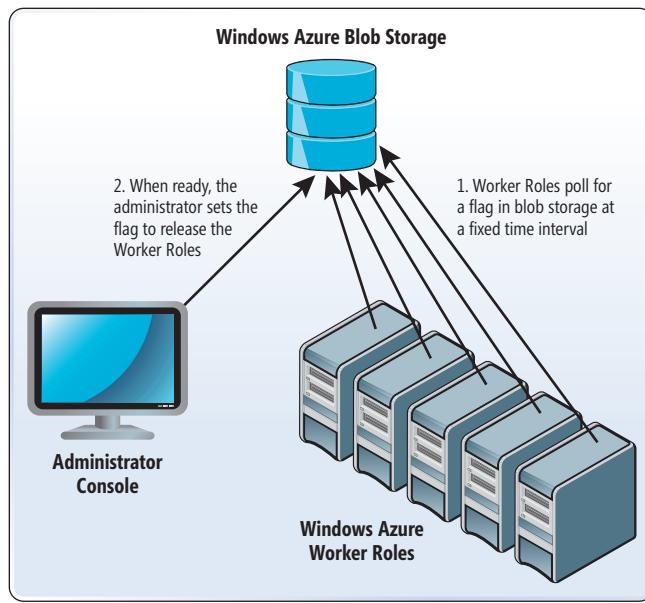


Figure 3 Nodes Polling a Central Status Flag

unique and the data would be corrupt. Traditionally, I might have considered creating a locking mechanism that prevents multiple nodes from working with the blob simultaneously. However, locks are expensive, and if throughput and massive scalability are guiding themes for the implementation, they're to be avoided.

Instead, an approach using optimistic concurrency is favorable. With optimistic concurrency, we allow multiple actors to interact with a resource. When the resource is retrieved by an actor, the actor is also issued a token that indicates the version of the resource. When an update takes place, the token can be included to indicate which version of the resource is being modified. If the resource has already been modified by another actor, then the update will fail and the original actor can retrieve the latest version and try the update again. Optimistic concurrency works well provided the chances of

contention on updates are low. The cost and complexity of a lock is avoided and the resource is protected from corruption.

Imagine that, during peak times, the system issues about 100 new order numbers per second. This would mean 100 requests to update the blob every second, causing an extremely high chance of contention, which would mean many retries, exacerbating the situation. Therefore, to reduce the possibility of this occurring, I decided to allocate order numbers in ranges.

The problem with using GUIDs is that they're difficult for humans to work with.

A class called the UniqueIdGenerator was created to encapsulate this behavior. The class removes a range of order numbers from blob storage by incrementing the value in configurable chunks. If each UniqueIdGenerator was to reserve 1,000 order numbers at a time, the blob would only be updated on average every 10 seconds, significantly reducing the chances of contention. Each UniqueIdGenerator is free to issue its reserved 1,000 order numbers at will, confident that no other instance of the class pointing to the same blob resource will issue the same order number.

In order to make this new component testable, an interface called IOptimisticSyncStore was specified that de-coupled the UniqueIdGenerator from the specific storage mechanism. This has an added advantage: In the future, the component could use a different type of storage where appropriate. Here's the interface:

```
public interface IOptimisticSyncStore
{
    string GetData();
    bool TryOptimisticWrite(string data);
}
```

As you can see, it's quite a simple interface with just two methods: one to retrieve the data and another to update it, the latter returning

Figure 4 The IGlobalFlag Interface and Implementation for Blob Storage

```
public interface IGlobalFlag
{
    bool GetFlag();
    void SetFlag(bool status);
}

public class BlobGlobalFlag : IGlobalFlag
{
    private readonly string _token = "Set";
    private readonly CloudBlob _blobReference;
    public BlobGlobalFlag(CloudStorageAccount account, string container,
        string address)
    {
        var blobClient = account.CreateCloudBlobClient();
        var blobContainer =
            blobClient.GetContainerReference(container.ToLower());
        _blobReference = blobContainer.GetBlobReference(address);
    }

    public void SetFlag(bool status)
    {
        if (status)
        {
            _blobReference.UploadText(_token);
        }
        else
        {
            _blobReference.DeleteIfExists();
        }
    }

    public bool GetFlag()
    {
        try
        {
            _blobReference.DownloadText();
            return true;
        }
        catch (StorageClientException exc)
        {
            if (exc.StatusCode == System.Net.HttpStatusCode.NotFound)
            {
                return false;
            }
            throw;
        }
    }
}
```

Figure 5 The PollingRelease Class

```
public class PollingRelease
{
    private readonly IGlobalFlag _globalFlag;
    private readonly int _intervalMilliseconds;

    public PollingRelease(IGlobalFlag globalFlag,
        int intervalMilliseconds)
    {
        _globalFlag = globalFlag;
        _intervalMilliseconds = intervalMilliseconds;
    }

    public void Wait()
    {
        while (!_globalFlag.GetFlag())
        {
            Thread.Sleep(_intervalMilliseconds);
        }
    }
}
```

a Boolean where false indicates that there was an optimistic concurrency failure and the process should be retried.

An implementation of `IOptimisticSyncStore` that uses blob storage is available in the code download (details at the end of the article). For the most part, the implementation is simple; however, it's worth looking at the `TryOptimisticWrite` method in more detail to understand how optimistic concurrency has been implemented.

It's simple to use optimistic concurrency when updating resources in Windows Azure Blob Storage, thanks to Preconditions and Entity Tags (ETags). A Precondition is a statement a developer asserts must be true for an HTTP request to succeed. If the Web server evaluates the statement to false, it should respond with an HTTP Status Code 412: "Precondition failed." ETags are also part of the HTTP specification and identify a particular version of a resource, such as a blob. If the blob changes, the ETag should change also, as shown here:

```
try
{
    _blobReference.UploadText(
        data,
        Encoding.Default,
        new BlobRequestOptions {
            AccessCondition = AccessCondition.IfMatch(
                _blobReference.Properties.ETag) });
}
```

To specify a Precondition in code, we use the `BlobRequestOptions` type and set the `AccessCondition` property. If this access condition isn't satisfied (for example, if another node updated the blob in the short time since it was retrieved), the ETags wouldn't match and a `StorageClientException` would be thrown:

```
catch (StorageClientException exc)
{
    if (exc.StatusCode == HttpStatusCode.PreconditionFailed)
    {
        return false;
    }
    else
    {
        throw;
    }
}
return true;
```

The implementation checks the exception for the `PreconditionFailed` status code and returns false in this instance. Any other type of exception is a serious failure and is rethrown for handling and logging further on. No exception means the update took place and

the method returns true. The full listing for the `UniqueIdGenerator` class is shown in **Figure 2**.

The constructor takes three parameters. The first is an implementation of `IOptimisticSyncStore`, such as our `BlobOptimisticSyncStore` discussed previously. The second parameter is `rangeSize`, an integer value that indicates how large the range of numbers allocated from the blob should be. The larger this range, the less chance of contention. However, more numbers would be lost if this node were to crash. The final parameter is `maxRetries`, an integer value that indicates how many times the generator should attempt to update the blob in the event of an optimistic concurrency failure. Beyond this point, an exception is raised.

Locks are expensive, and if throughput and massive scalability are guiding themes for the implementation, they're to be avoided.

The `NextId` method is the only public member of the `UniqueIdGenerator` class and is used to fetch the next unique number. The body of the method is synchronized to ensure that any instance of the class is thread-safe and could, for example, be shared among all the threads running your Web application. An `if` statement checks to see if the generator has reached the upper limit of its range allocation and, if so, calls `UpdateFromSyncStore` to fetch a new range from blob storage.

The `UpdateFromSyncStore` method is the final but most interesting part of the class. The implementation of `IOptimisticSyncStore` is used

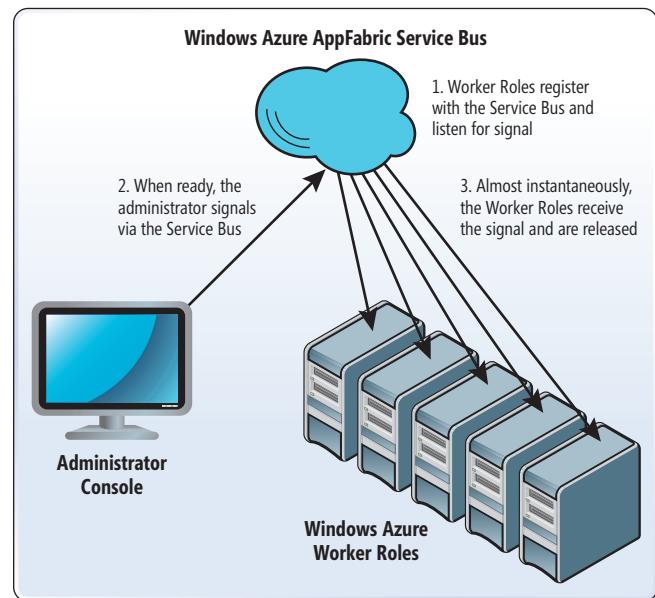


Figure 6 Using the Windows Azure AppFabric Service Bus to Simultaneously Communicate with All Worker Roles

Figure 7 The ConnectToServiceBus Method

```
private IDisposable ConnectToServiceBus()
{
    Uri address = ServiceBusEnvironment.CreateServiceUri("sb",
        _serviceNamespace, _servicePath);
    TransportClientEndpointBehavior sharedSecretServiceBusCredential =
        new TransportClientEndpointBehavior();
    sharedSecretServiceBusCredential.CredentialType =
        TransportClientCredentialType.SharedSecret;
    sharedSecretServiceBusCredential.Credentials.SharedSecret.
        IssuerName = _issuerName;
    sharedSecretServiceBusCredential.Credentials.SharedSecret.
        IssuerSecret = _issuerSecret;

    // Create the single instance service, which raises an event
    // when the signal is received.
    UnleashService unleashService = new UnleashService();
    unleashService.Unleashed += new
        EventHandler(unleashService_Unleashed);

    // Create the service host reading the configuration.
    ServiceHost host = new ServiceHost(unleashService, address);

    IEndpointBehavior serviceRegistrySettings =
        new ServiceRegistrySettings(DiscoveryType.Public);

    foreach (ServiceEndpoint endpoint in host.Description.Endpoints)
    {
        endpoint.Behaviors.Add(serviceRegistrySettings);
        endpoint.Behaviors.Add(sharedSecretServiceBusCredential);
    }

    host.Open();
    return host;
}
```

to fetch the upper limit of the previous allocation issued. The value is incremented by the generator's range size, and this is written back to storage. A simple "while" loop encloses the body to ensure that the appropriate number of retries takes place if TryOptimisticWrite returns false.

The following code snippet shows a UniqueIdGenerator being constructed, using a BlobOptimisticSyncStore with a file called "ordernumber.dat" in a container called "uniqueids" (note: containers in blob storage must have lowercase names):

```
IOptimisticSyncStore storage = new BlobOptimisticSyncStore(
    CloudStorageAccount.DevelopmentStorageAccount,
    "uniqueids",
    "ordernumber.dat");
UniqueIdGenerator
generator = new UniqueIdGenerator(storage, 1000, 10);
```

This instance removes 1,000 IDs from the central range and will retry 10 times in the event of an optimistic concurrency failure before throwing an exception.

Using the UniqueIdGenerator is even simpler. Wherever you need a new unique order number, simply call NextId:

```
Int64 orderId = generator.NextId();
```

The sample code shows a Windows Azure Worker Role that uses multiple threads to quickly allocate unique order numbers and write them to a SQL database. The use of SQL in this instance is simply to prove that every order number is unique—any violation of this would result in a Primary Key violation and throw an exception.

The advantage to this approach—other than creating the blob and setting its value to 0 at the very start of the application's lifetime—is that no effort is required of the systems administrator. The UniqueIdGenerator carefully manages the allocation of IDs based on your settings, recovers gracefully in the event of a failure and scales effortlessly even in the most elastic of environments.

Scenario 2: Release the Hounds!

Another interesting requirement posed by the application was the need to rapidly process large amounts of data following a specified event that would occur at an approximately known time. Due to the nature of the processing, work couldn't commence on any of the data until after this event.

Worker Roles are an obvious choice in this scenario, and it would have been possible to simply ask Windows Azure to provision the necessary number of Worker Roles in response to the aforementioned event. However, provisioning new roles can take as long as 30 minutes, and speed was of the essence in this scenario. Therefore, it was decided that the roles would be hydrated in advance but in a paused state until released by an administrator—I called this "Release the Hounds!" Two possible approaches were considered, and I'll review each in turn.

It should be noted that, because Windows Azure Worker Roles are charged based on the time they're deployed (not on how actively they use the CPU), this approach would cost more compared to simply creating the Worker Roles in response to the event. However, the customer was clear that this investment was worthwhile to ensure that processing could begin as quickly as possible.

Approach I: Polling

The first approach, shown in Figure 3, had each node poll a central status flag at a regular interval (again, stored in a Windows Azure blob) to determine whether work could yet commence.

To un-pause the nodes, a client application simply had to set this flag to true, and with the subsequent poll, each node would be released. The primary disadvantage of this approach is latency, potentially as large as the polling interval. On the other hand, this is a very simple and reliable mechanism to implement.

This design is demonstrated by the PollingRelease class available in the sample code. In order to support testability, the flag-storage mechanism was abstracted behind an interface in much the same way as for the UniqueIdGenerator class. The interface IGlobalFlag and accompanying implementation for blob storage are shown in Figure 4.

Notice that in this example, the mere existence of a file in blob storage indicates true, no matter what the content.

The PollingRelease class itself is straightforward, as shown in Figure 5, with just one public method called Wait.

Figure 8 The UnleashService Class

```
[ServiceBehavior(InstanceContextMode= InstanceContextMode.Single)]
public class UnleashService : IUnleashContract
{
    public void Unleash()
    {
        OnUnleashed();
    }

    protected virtual void OnUnleashed()
    {
        EventHandler temp = Unleashed;
        if (temp != null)
        {
            temp(this, EventArgs.Empty);
        }
    }

    public event EventHandler Unleashed;
}
```

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

Copyright 1996-2010 Infragistics, Inc. All rights reserved. Infragistics, the Infragistics logo and NetAdvantage are registered trademarks of Infragistics, Inc. xamMap is a trademark of Infragistics, Inc.

Infragistics Sales 800 231 8588

Infragistics Europe Sales +44 (0) 800 298 9055

Infragistics India +91-80-6785-1111

t@infragistics.com

This method blocks any caller as long as the IGlobalFlag implementation indicates that its status is false. The following code snippet shows the PollingRelease class in use:

```
BlobGlobalFlag globalFlag = new BlobGlobalFlag();
CloudStorageAccount developmentStorageAccount =
    "globalflags",
    "start-order-processing.dat");
PollingRelease pollingRelease = new
PollingRelease(globalFlag, 2500);
pollingRelease.Wait();
```

A BlobGlobalFlag instance is created pointing at a container called “globalflags.” The PollingRelease class will poll every 2.5 seconds for the presence of a file called “start-order-processing.dat”; any call to the Wait method will be blocked until this file exists.

Approach II: Listening

The second approach uses the Windows Azure AppFabric Service Bus to simultaneously communicate with all worker roles directly and release them (see **Figure 6**).

The Service Bus is a large-scale messaging and connectivity service, also built on Windows Azure. It facilitates secure communication among different components of a distributed application. The Service Bus provides an ideal way to connect two applications that would otherwise find it difficult to communicate, due to their location behind a network address translation (NAT) boundary or a frequently changing IP address, for example. It’s beyond the scope of this article to give a detailed overview of the Windows Azure AppFabric Service Bus, but an excellent tutorial is available on MSDN at msdn.microsoft.com/library/ee706736.

The Service Bus is a
large-scale messaging and
connectivity service.

To demonstrate this approach, a class called ListeningRelease was created that, like PollingRelease, has one public method called Wait. This method connects to the Service Bus and uses a ManualResetEvent to block the thread until a signal is received:

```
public void Wait()
{
    using (ConnectToServiceBus())
    {
        _manualResetEvent.WaitOne();
    }
}
```

The full ConnectToServiceBus method is listed in **Figure 7**. It uses types from the System.ServiceModel and Microsoft.ServiceBus assemblies to expose a class called UnleashService to the cloud via the Windows Azure AppFabric Service Bus, shown in **Figure 8**.

The UnleashService is hosted by Windows Communication Foundation (WCF) as a single instance and implements the IUnleashService contract, which has just one method: Unleash. ListeningRelease listens for an invocation of this method through the Unleashed event shown earlier. When the ListeningRelease

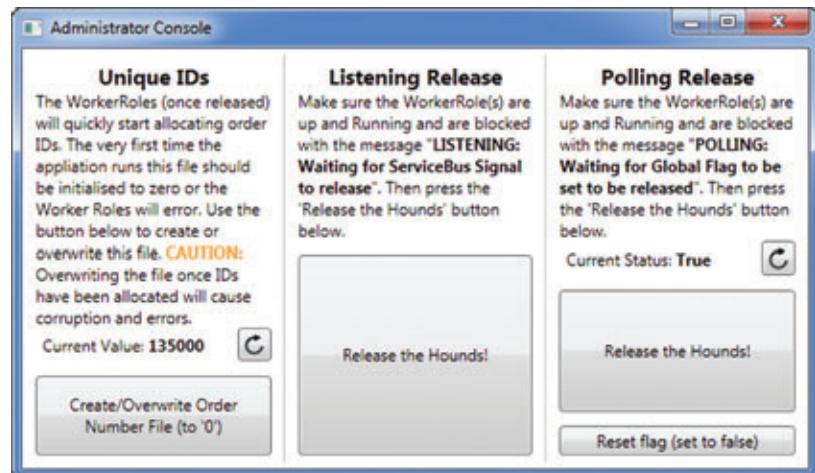


Figure 9 The Administrator Console

class observes this event, the ManualResetEvent that’s currently blocking any calls to Wait is set and all blocked threads are released.

In the configuration for the service, I used the NetEventRelay-Binding, which supports multicasting through the Service Bus, allowing any number of publishers and subscribers to communicate through a single endpoint. The nature of this broadcast communication requires that all operations are one-way, as demonstrated by the IUnleashContract interface:

```
[ServiceContract]
public interface IUnleashContract
{
    [OperationContract(IsOneWay=true)]
    void Unleash();
}
```

The endpoint is secured using a Shared Secret (username and complex password). With these details, any client with access to the Internet could invoke the Unleash method—including, for example, the Administrator Console provided in the sample (see **Figure 9**).

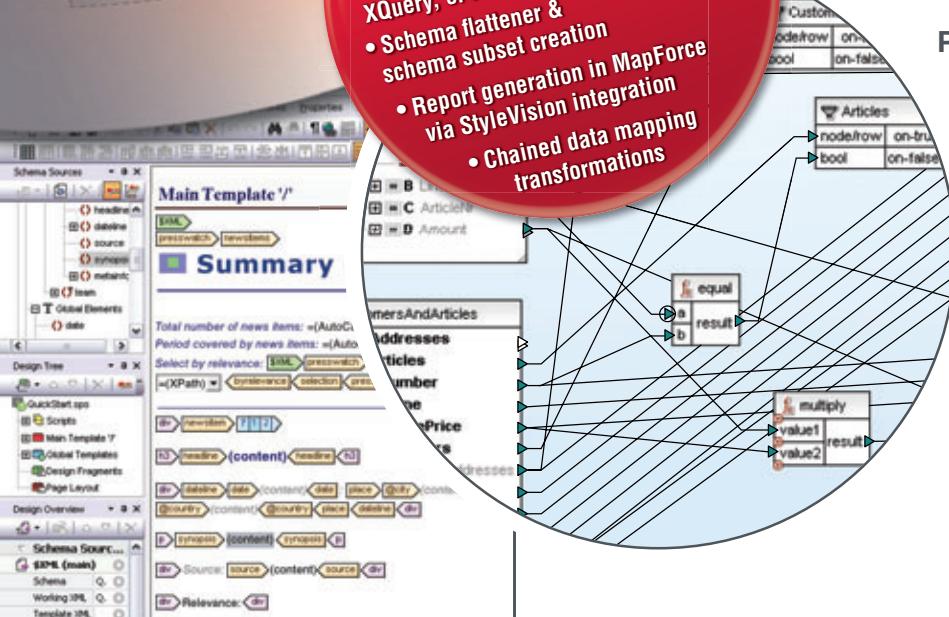
Although the ListeningRelease approach does away with the inherent latency in the PollingRelease class, there’s still some latency to deal with. However, the main disadvantage with the listening approach is its stateless nature, such that any nodes provisioned after the release signal has been transmitted wouldn’t see this event and would remain paused. Of course, an obvious solution might be to combine both the Service Bus and a global flag in blob storage, but I’ll leave that as an exercise for the reader.

Sample Code

The accompanying sample solution is available at code.msdn.microsoft.com/mag201011Sync and includes a ReadMe file that lists the prerequisites and includes the setup and configuration instructions. The sample uses the ListeningRelease, PollingRelease and UniqueIdGenerator in a single Worker Role.

JOSH TWIST is a principal application development manager with the Premier Support for Developers team in the United Kingdom, and can be found blogging at thejoyofcode.com.

THANKS to the following technical experts for reviewing this article:
David Goon, Morgan Skinner and Wade Wegner



Bring your
XML development
projects to light
with the complete set
of tools from Altova®



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

The Altova MissionKit includes multiple intelligent XML tools – now with cutting edge chart and report generation:

XMLSpy® – industry-leading XML editor

- Support for all XML-based technologies
- Graphical editing views, powerful debuggers, code generation, & more

MapForce® – graphical data mapping tool

- Drag-and-drop data conversion with code generation
- Support for XML, DBs, EDI, Excel® 2007+, XBRL, flat files & Web services

StyleVision® – visual stylesheet & report designer

- Graphical stylesheet and report design for XML, XBRL & databases
- Report designer with chart creation
 - Output to HTML, PDF, Word & e-Forms

Plus up to five additional tools...



Download a 30 day free trial!

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



XCEED DataGrid for WPF

Serious recognition

Microsoft®
Visual Studio® Team System 2010

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

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

IBM®
U2 SystemBuilder™

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

Vincent Smith,
U2 Tools Product Manager at IBM

**NEW FOR
SILVERLIGHT!**



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.

Datagrids, transformed

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



The smooth-scrolling Tableflow™ view provides a rich, fluid, and high-performance experience. Its innovative group navigation control redefinesdatagrid usability.

The Cardflow™ 3D view lets you add a true 3D experience to your application. Also offered is a classic 2D card view.

The first WPF datagrid on the market and under constant development. Build apps you can trust in mission-critical situations.

Try it live on xceed.com

XCEED
MULTI-TALENTED COMPONENTS



PARTNER

©2010 Xceed Software Inc. All rights reserved. All product and brand names are trademarks and/or registered trademarks of their respective owners.

Connecting SharePoint to Windows Azure with Silverlight Web Parts

Steve Fox

Microsoft SharePoint 2010 is enjoying much-deserved praise as a solid developer platform. Augmented with new services, APIs, data programmability, and UI support via the dialog framework and Silverlight, many options exist for developers to really sink their teeth into this evolved platform.

With the growing interest in cloud computing, though, I increasingly get questions about how developers can integrate their SharePoint apps with cloud-based technologies. As a platform, many of the aforementioned features can be integrated with Windows Azure in some way. Further, you can integrate SharePoint with the cloud through a host of other technologies such as OData, REST, Web 2.0 social APIs for applications like Twitter or Facebook, and, of course, through a service-oriented architecture using SOAP or Windows Communication Foundation (WCF) services.

Knowing that there's broad integration potential between the cloud and SharePoint, in this article I'll explore some specific integration points between SharePoint and Windows Azure. Along the way I'll walk through the steps for creating your first integration.

This article discusses:

- Platform basics
- Creating a WCF service
- The Silverlight Web Part
- Deploying the solution

Technologies discussed:

Windows Azure, SharePoint 2010, Visual Studio 2010, Silverlight

Code download available at:

code.msdn.microsoft.com/mag201011Azure

Platform Basics

The Windows Azure platform is made up of three parts. First, Windows Azure itself provides data and management capabilities. Second, SQL Azure provides highly available and transactional data in the cloud. Third, Windows Azure AppFabric provides a service bus for more advanced, direct service call scenarios.

Using Windows Azure, you can support a number of different types of integration. For example, you can build and deploy a WCF service to the cloud, then integrate that service within SharePoint. Or you can consume data from Windows Azure, modeling that data within SharePoint. Further, you can use the Windows Azure AppFabric Service Bus to generate more complex service scenarios that connect SharePoint Online with SharePoint on-premises.

With any integration, you need to understand the possibilities. **Figure 1** provides a starting point, listing the different ways in which you can integrate SharePoint with Windows Azure. This table is specific to SharePoint 2010, and some of these options require more coding than others.

Whatever integration you choose, it's important to note that in this article, when integrating with Windows Azure, SharePoint is consumptive and not being hosted. In other words, SharePoint is not a service that is hosted by Windows Azure, but rather an application that consumes Windows Azure data or services. Windows Azure provides applications or resources that will be consumed by SharePoint artifacts such as a Web Part or Silverlight application. In this article, you'll see a specific example of how you integrate a Silverlight application within SharePoint that leverages a custom WCF service deployed to Windows Azure.

If you're starting fresh, you'll need to make sure you have an appropriate development environment set up. Your development environment will, at the least, include the following:

Figure 1 Common Integration Points

Azure Integration	How
SharePoint Client Object Model	Interact with Windows Azure data in a list.
Business Connectivity Services (BCS)	Model data from Windows Azure or build external list to SQL Azure.
Silverlight	Create UI against Windows Azure services or data.
Sandboxed Solutions/ SharePoint Online	Silverlight application leveraging Windows Azure deployed to site collection.
Office Custom Client	Consume data directly from Windows Azure or BCS list exposing data.
Standard/Visual Web Parts	Leverage services and data from Windows Azure.
Open XML	Manage Windows Azure data in a document.
REST	Use REST to interact with Windows Azure data to integrate with SharePoint.
Office Server Services	Combine with Open XML to auto-gen docs (such as PDFs) on a server.
Workflow/ Event Receivers	State or events that tie into Windows Azure services, workflows or data.
LINQ	Use for querying Windows Azure data objects.
Search	Federate search to include Windows Azure data.

- Visual Studio 2010
- Windows Azure SDK and tools
- SharePoint Server 2010
- Office 2010 Professional Plus
- SQL Server 2008 R2
- Silverlight runtime, SDK and tools

For Windows Azure, you'll need to make sure you have a developer account set up so that you can create a developer portal to manage your cloud applications, data and services. You can find all of the Windows Azure tools that you'll use to build these applications and services at microsoft.com/windowsazure/getstarted. Note that you can install the items listed earlier on your existing development machine or you can download a pre-configured virtual machine that has everything except the Windows Azure tools from tinyurl.com/33bgpy6. (Also, you can optionally install the Silverlight Web Part Visual Studio extension available at code.msdn.microsoft.com/vsixforsp.)

When you've got your development environment set up, you can get started developing your first integration. In this article, I'll work through three steps:

1. Create and deploy the custom Windows Azure WCF service.
2. Create a Silverlight-enabled Web Part that can consume the custom Windows Azure service.
3. Deploy and use the Silverlight-enabled Web Part in your SharePoint site.

Let's walk through each of these steps.

Creating the WCF Service

Imagine you want to deploy a service to your entire sales organization, but you want to host that service in the cloud. The service will retrieve competitor information and will support two methods: It will enable you to get *specific* competitor information, and it will return a list of *all* competitor information. You'll create both

methods, but will implement only the bulk return of competitor information. You can extend the code after reading this article to leverage the request for specific competitor information.

To create the WCF service, open up Visual Studio 2010 and start a new project. In the New Project wizard, select the Cloud template. Provide a name for the project (I called mine Competitors) and click OK. Select the WCF Service Web Role project and click OK.

Visual Studio creates a new solution with a number of resources, which include Windows Azure role configuration files and your WCF service code and contract. You'll use an in-memory object in this example, so right-click the WCF project, select Add, then select Class. Provide a name for the class (Competitor), and add the following code to the class file:

```
namespace WCFServiceWebRole1 {
    public class Competitor {
        public string svcCompeteID { get; set; }
        public string svcCompeteName { get; set; }
        public string svcCompeteFY09 { get; set; }
        public string svcCompeteFY10 { get; set; }
    }
}
```

The code includes four properties (a competitor ID, name, and sales for fiscal year 2009 and fiscal year 2010).

Because you'll also be using Silverlight as your presentation layer, you'll need to add a client access policy file to the project to support Silverlight calling the Windows Azure service across domains. To do this, right-click the WCF project, select Add, then click New Item. In the New Item dialog, select the Data category and select XML. Name the new file clientaccesspolicy.xml and click Add. Replace the XML code in the new file with this code:

```
<?xml version="1.0" encoding="utf-8"?>
<access-policy>
    <cross-domain-access>
        <policy>
            <allow-from http-request-headers="SOAPAction">
                <domain uri="*"/>
            </allow-from>
            <grant-to>
                <resource path="/" include-subpaths="true"/>
            </grant-to>
        </policy>
    </cross-domain-access>
</access-policy>
```

Next, create the service contract for your WCF service. For this example, you'll keep things simple and just include two operational contracts. The first operation will get a single competitor (the getACOMPETITOR method) and the second operation will get all

Figure 2 Service Contract

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Text;

namespace WCFServiceWebRole1 {
    [ServiceContract]
    public interface IService1 {
        [OperationContract]
        string[] getACOMPETITOR(string custID);

        [OperationContract]
        List<Competitor> getAllCOMPETITORS();
    }
}
```

Figure 3 Service Code

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Text;

namespace WCFServiceWebRole1 {
    public class Service1 : IService1 {
        List<Competitor> myCompetitors = new List<Competitor>();

        public string[] getACompetitor(string custID) {
            generateCompeteData();
            string[] returnCompete = new string[4];

            var returnListOfData =
                from compete in myCompetitors
                where compete.svcCompeteID == custID
                select compete).ToArray();

            foreach (var competeRecord in returnListOfData) {
                returnCompete[0] = competeRecord.svcCompeteID;
                returnCompete[1] = competeRecord.svcCompeteName;
                returnCompete[2] = competeRecord.svcCompeteFY09;
                returnCompete[3] = competeRecord.svcCompeteFY10;
            };
            return returnCompete;
        }

        public List<Competitor> getAllCompetitors() {
            generateCompeteData();
            List<Competitor> returnlistOfCompetitors =
                new List<Competitor>();

            var returnListOfData =
                from customer in myCompetitors
                select customer).ToArray();

            foreach (var compete in returnListOfData) {
                Competitor tempCompeteRecord = new Competitor();
                tempCompeteRecord.svcCompeteID = compete.svcCompeteID;
                tempCompeteRecord.svcCompeteName = compete.svcCompeteName;
                tempCompeteRecord.svcCompeteFY09 = compete.svcCompeteFY09;
                tempCompeteRecord.svcCompeteFY10 = compete.svcCompeteFY10;
                returnlistOfCompetitors.Add(tempCompeteRecord);
            };

            return returnlistOfCompetitors;
        }

        private void generateCompeteData() {
            Competitor compete1 = new Competitor();
            compete1.svcCompeteID = "BR-CAN-8909";
            compete1.svcCompeteName = "Bauer - Canada";
            compete1.svcCompeteFY09 = "$45,093,028.00";
            compete1.svcCompeteFY10 = "$50,493,820.00";
            myCompetitors.Add(compete1);

            Competitor compete2 = new Competitor();
            compete2.svcCompeteID = "NK-USA-8665";
            compete2.svcCompeteName = "Nike - USA";
            compete2.svcCompeteFY09 = "$50,492,331.00";
            compete2.svcCompeteFY10 = "$52,019,828.00";
            myCompetitors.Add(compete2);

            Competitor compete3 = new Competitor();
            compete3.svcCompeteID = "GF-EU-9912";
            compete3.svcCompeteName = "Graf - Europe";
            compete3.svcCompeteFY09 = "$24,403,920.00";
            compete3.svcCompeteFY10 = "$24,001,926.00";
            myCompetitors.Add(compete3);

            Competitor compete4 = new Competitor();
            compete4.svcCompeteID = "CCM-USA-8843";
            compete4.svcCompeteName = "CCM Hockey";
            compete4.svcCompeteFY09 = "$12,209,105.00";
            compete4.svcCompeteFY10 = "$10,092,813.00";
            myCompetitors.Add(compete4);
        }
    }
}

```

competitors (getAllCompetitors method). Note that you'll need to pass an ID (custID) if you want to return a specific competitor record. **Figure 2** provides a summary of the contracts.

With the contracts complete, you can now add some code that implements the contracts. The service code that corresponds

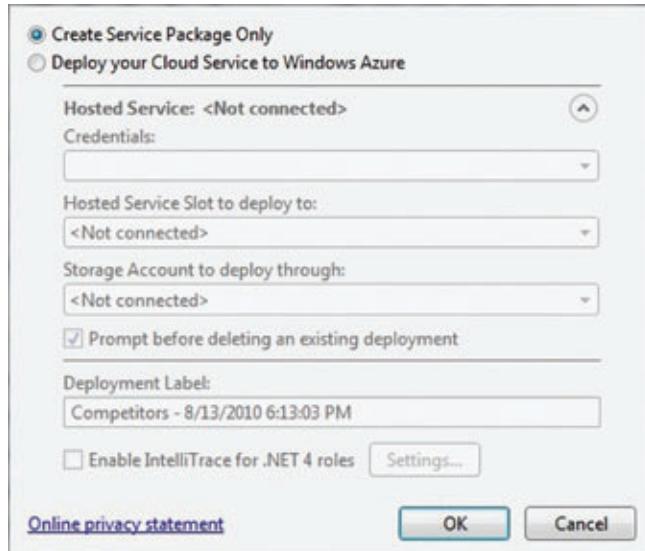


Figure 4 Service Publishing Options

to the service contract is shown in **Figure 3**. This code includes a method to get a single competitor record (getACompetitor), another to get all competitor information (getAllCompetitors) and a third to generate the competitor information (generateCompeteData). The code is straightforward, leveraging in-memory data structures such as list collections and arrays along with LINQ to create and pass data back to the calling application. In this example, the calling application is a Silverlight application that will be deployed into SharePoint.

Windows Azure provides applications or resources that will be consumed by SharePoint.

At this point, you've now created the WCF service and you're almost ready to start the SharePoint part of this integration. Before you can do that, though, you need to deploy the service to Windows Azure.

To deploy the service, you must have a Windows Azure developer account set up and ready to go. Assuming you have this, you simply right-click the Windows Azure project and select Publish.



RadControls for Silverlight

The industry leading UI components for Silverlight with unmatched performance and pioneering support for Silverlight 4.

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

www.telerik.com/Silverlight

 **telerik**
deliver more than expected

Europe HQ: +359.2.80.99.850 • US Sales: +1.888.365.2779 • Germany Sales: +49.89.8780687.70 e-mail: sales@telerik.com

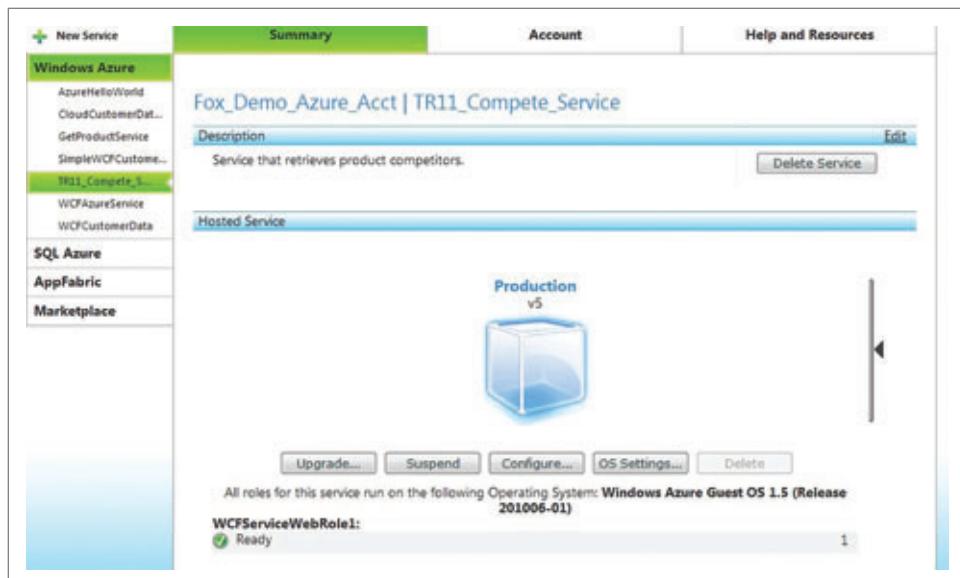


Figure 5 Manually Deploying Services to Windows Azure

Publishing your service invokes a dialog box where you can provide your credentials. You can see in **Figure 4** that you have the choice to create a service package only (Visual Studio creates the two core files that need to be added to your Windows Azure developer portal in a local folder) or to deploy your service automatically using the pre-configured information. In this example, click Create Service Package Only and click OK.

Two files, Competitors and ServiceConfiguration, will be created. Competitors is a service package file—essentially a resource archive—and ServiceConfiguration is an XML configuration file.

You can now navigate to your Windows Azure developer portal and add these files to your service. To do this, navigate to your service and click Deploy (or if you've deployed already and are upgrading the service, click Upgrade as shown in **Figure 5**). You can then browse to the two files and click OK. You'll want to give your service files a few minutes to upload.

When you see the Ready message, you can click the link that's displayed on the same Web page to test the service endpoint. Note that you'll likely have to add the service name at the end of the service URL like so:

<http://serviceendpoint.azure.com/Service1.svc>.

At this point, you can now put Windows Azure aside and move on to SharePoint.

Creating the Silverlight-Enabled Web Part

You can create the Silverlight Web Part for SharePoint in a couple of ways. One way is to simply create a Silverlight application in Visual Studio, deploy the XAP file to SharePoint (by uploading it to a document library, for example), and using the native

Silverlight Web Part in SharePoint 2010 to load your Silverlight application. This is the quickest way to deploy the Silverlight application to SharePoint, and requires less coding.

A second, slightly more interesting way is to use the Silverlight and SharePoint Web Part project template (code.msdn.microsoft.com/vsixforwp). This automatically wraps a Silverlight app with a Web Part, which means you simply create the Silverlight application and deploy it as a Web Part to SharePoint. You have a little more control over your code, plus you're deploying a real Web Part to SharePoint.

To use the template, navigate to the Codeplex site, click the Silverlight and SharePoint VSIX link,

download and unzip the files. After you unzip the files, simply install the .vsix file by double-clicking it, then restart Visual Studio 2010.

After you install the Silverlight Web Part template, return to your Visual Studio solution and click File | Add | New Project, and select Add to Solution in the Solution field. Navigate to the SharePoint 2010 folder and select the Silverlight Web Part project template. Provide a name for your Web Part (I used CompetitorSPWebPart) and click OK.

After you click OK, you'll be prompted for a SharePoint site URL. Add your site URL here and then select Deploy as Farm Solution. You're next prompted for a number of items, including the name for the Silverlight project, version, location to deploy the XAP file,

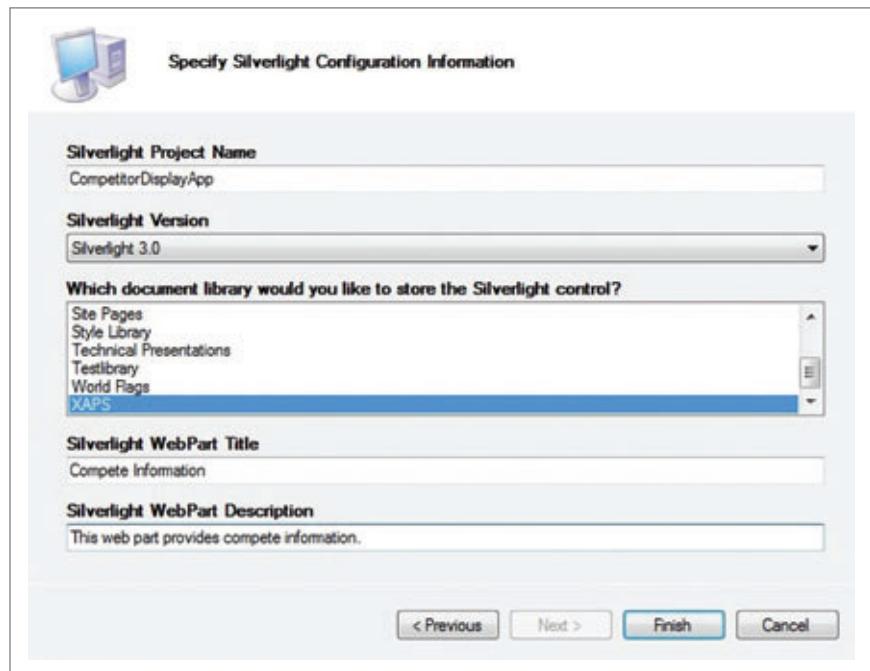


Figure 6 Configuring the Silverlight Web Part

A New Evolution, a New Opportunity with Microsoft Dynamics AX

Q What is Microsoft Dynamics?

A Microsoft Dynamics is a line of ERP and CRM applications developed by the Microsoft Business Solutions group within Microsoft. You can read more at <http://www.microsoft.com/dynamics>.

Q What is Microsoft Dynamic AX?

A Microsoft Dynamics AX is a Microsoft enterprise resource planning product that provides solutions for industries like distribution, manufacturing, professional services, retail, and public sector. <http://www.microsoft.com/dynamics/en/us/products/ax-overview.aspx>

Q Why is this interesting for developers?

A Microsoft Dynamics AX is an application built on the Microsoft stack. Developers can leverage their skills in the .NET Framework to extend the solution into specific industry vertical.

Q How is the .NET Framework is leveraged?

A For example, the Windows Workflow foundation is the orchestration engine coordinating business processes in Microsoft Dynamics AX. The Windows Communication Foundation provides a unified programming model for leveraging business processes through services.

Q How is the application lifecycle managed?

A Microsoft Dynamics AX is integrated with Visual Studio Team System allowing developers to manage their solution with standard Microsoft tools.

Q Is there a new release coming?

A Yes. A new version of Microsoft Dynamics AX is expected in 2011.

Q Is there somewhere that I could go and learn about the upcoming release?

A Yes. We are hosting a pre-release conference called the Microsoft Dynamics AX Technical Conference 2011 focused on helping developers get their solutions prepared for the next release.

Q What can people expect when attending this conference?

A We have a great lineup. There will be over 65 sessions covering the upcoming release, with time set aside to collaborate with the development team in Chalk and Talk



and Ask the Experts sessions. Microsoft Dynamics AX Certified Trainers will host instructor-led labs.

Also, partners and customers can network with the development team, not only from Microsoft Dynamics AX, but with others that work on the core technologies that AX builds on.

Q What topics will be covered?

A With all of the innovation in this new version there is something for everyone. Topics cover developer tools, database modeling, application component design, business intelligence tools, office integration around Microsoft SharePoint and the core Microsoft Office products, services integration, IT management, performance, and application lifecycle management.

Q Sounds great! How do I sign up?

A We are excited to be able to put on this event, as we think it is our little version of PDC for developers or TechEd for the IT folks, but just focused on Microsoft Dynamics AX and specifically the next release.

Register at <http://www.microsoft.com/dynamics/daxconf2011/msdn>. And be sure to check out the sample of the session content. Look for the full catalog by the end of October.

Experience How. Get Involved.

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

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

Microsoft

Figure 7 Custom Competitor Object

```
using CompetitorDisplayApp.GetCompeteAzureService;

namespace CompetitorDisplayApp {
    public partial class MainPage : UserControl {
        public string SiteUrl { get; set; }

        List<Competitor> myCompetitors = new List<Competitor>();

        public MainPage() {
            InitializeComponent();

            this.Loaded += new RoutedEventHandler(MainPage_Loaded);
        }

        void MainPage_Loaded(object sender, RoutedEventArgs e) {
            LoadAzureData();
        }

        private void LoadAzureData() {
            GetCompeteAzureService.Service1Client azureSvcProxy =
                new Service1Client();
            azureSvcProxy.getAllCompetitorsAsync();
        }
    }

    public class Competitor {
        public string competeID { get; set; }
        public string competeName { get; set; }
        public string competeFY09 { get; set; }
        public string competeFY10 { get; set; }
    }
}

azureSvcProxy.getAllCompetitorsCompleted +=
    new EventHandler<getAllCompetitorsCompletedEventArgs>(
        azureSvcProxy_getAllCompetitorsCompleted);
}

void azureSvcProxy_getAllCompetitorsCompleted(
    object sender, getAllCompetitorsCompletedEventArgs e) {

    var competeData = e.Result;
    foreach (var item in competeData) {
        Competitor tempRecord = new Competitor();
        tempRecord.competeID = item.svcCompeteID;
        tempRecord.competeName = item.svcCompeteName;
        tempRecord.competeFY09 = item.svcCompeteFY09;
        tempRecord.competeFY10 = item.svcCompeteFY10;
        myCompetitors.Add(tempRecord);
    }

    competeList.ItemsSource = myCompetitors;
}
}
```

title for your Web Part and description for your Web Part (see **Figure 6**). Click Finish when you've completed this part of the wizard. (Note that you can use Silverlight 3 or 4 with the project template, and Microsoft is currently upgrading the template for re-release on Codeplex.)

Now you have a SharePoint Web Part that wraps a Silverlight application, and you can use the Silverlight application to build out the UI and functionality of the Web Part.

First, in the Silverlight project, add a reference to the Windows Azure service by right-clicking References and selecting Add Service Reference. Provide a name for the namespace for your service (in my case, GetCompeteAzureService) and click OK. This is the same as consuming any other service reference in an application, except in this case the endpoint is pointing to a Windows Azure service.

At this point, you can code against the Windows Azure service. As mentioned earlier, you'll leverage the *getAllCompetitors* method in the SharePoint application.

You can create the Silverlight Web Part for SharePoint in a couple of ways.

You'll need a UI for your application. I created a simple UI that will render the data returned from the call to the Windows Azure service. The core control is a listbox control with a couple of images added for flair. See the code download for this article for details.

Next, add a custom class called Competitor to the Silverlight application. Competitor has four properties that correspond to the competitor data defined for the service code in **Figure 3**:

```
namespace CompetitorDisplayApp {
    public class Competitor {
        public string competeID { get; set; }
        public string competeName { get; set; }
        public string competeFY09 { get; set; }
        public string competeFY10 { get; set; }
    }
}
```

In the XAML code-behind, you can now add some code that will implement the *getAllCustomers* method. In **Figure 7**, you can see that I'm using a list collection called myCompetitors to store the data being returned from the service call to Windows Azure. There isn't a lot of heavy lifting here at all; the code is using the Competitor object to help populate the myCompetitors list collection, which is then bound to the listbox (competeList).

At this point, you're finished with your coding. However, it's worth taking a quick look at the default code that's created by the

Figure 8 Default Web Part Code

```
[ToolboxItemAttribute(false)]
public class SilverlightWebPart : WebPart {
    private SilverlightPluginGenerator _silverlightPluginGenerator = null;

    public SilverlightWebPart() {
        this._silverlightPluginGenerator =
            new SilverlightPluginGenerator {

                Source = new Uri(
                    "/XAPS/Silverlight/CompetitorDisplayApp/CompetitorDisplayApp.xap",
                    UriKind.Relative),
                Width = new Unit(400, UnitType.Pixel),
                Height = new Unit(300, UnitType.Pixel),
                BackGround = Color.White,
                Version = SilverlightVersion.v3,
                AutoUpgrade = true,
                OnError = "onSilverlightError",
            );
    }

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

        this.Controls.Add(new LiteralControl(
            @"<script type=""text/javascript"">" +
            Resources.onSilverlightErrorHandler +
            "</script>"));

        this._silverlightPluginGenerator.InitParams.Add(new InitParam(
            "SiteUrl", SPContext.Current.Site.Url));

        this.Controls.Add(new LiteralControl(
            this._silverlightPluginGenerator.ToString()));
    }

    protected override void RenderContents(HtmlTextWriter writer) {
        base.RenderContents(writer);
    }
}
```

Silverlight Web Part template to show why it can be more useful than just using the default Silverlight Web Part that ships with SharePoint.

Figure 8 shows the default Web Part code that's created when Visual Studio creates the Silverlight Web Part project. You can see the wrapper code where a SilverlightPluginGenerator object is created and properties are set for your Web Part. These are properties that are managed at design time (as opposed to opening the Web Part, for example, and editing the height and width through the Tools pane in SharePoint). Further, you can avoid copy and paste, given this Web Part is deployed into the Web Part gallery—with the stitchwork already in place to render the Silverlight application.

Finally, the properties of the Web Part were set when you walked through the initial configuration wizard. For example, if you open the .webpart file, you'll see the name and description of the Web Part (which you can change here if you wanted):

```
<?xml version="1.0" encoding="utf-8"?>
<webParts>
  <webPart xmlns="http://schemas.microsoft.com/WebPart/v3">
    <metadata>
      <type name="CompetitorSPWebPart.SilverlightWebPart".
SilverlightWebPart, $SharePoint.Project.AssemblyFullName$" />
      <importErrorMessage>$Resources:core,ImportErrorMessage;</
importErrorMessage>
    </metadata>
    <data>
      <properties>
        <property name="Title" type="string">Compete Information</property>
        <property name="Description"
          type="string">This Web Part provides compete information.</property>
      </properties>
    </data>
  </webPart>
</webParts>
```

With the Silverlight Web Part complete, you're now ready to deploy the Web Part to SharePoint for use.

Deploying the Web Part

To deploy, right-click the Web Part project (CompetitorSPWebPart in my example) and select Deploy. When you deploy the Web Part, the Silverlight application is deployed into the XAPS document library, and a link to that XAP file is automatically generated within the Web Part. (If you chose to use the Silverlight application instead of the Silverlight Web Part template, then you'd simply upload the XAP file into the XAPS document library and then use the native Silverlight Web Part in SharePoint.)

Now open your SharePoint site and navigate to (or create) a new Web Part page. (Note that you can create a new Web Part page by clicking Site Actions | View All Site Content | Create | Web Part Page.) Click Site Actions | Edit Page | Add a Web Part. Navigate to the Custom category, then select the Web Part (which is called Compete Information in this example), and click Add. When added, click Stop Editing. You should have something similar to what's shown in **Figure 9**.



Figure 9 Final Silverlight Web Part Calling Windows Azure Service

for more code and documentation on how to integrate SharePoint and Windows Azure. ■

STEVE FOX is a senior evangelism manager at Microsoft. He's worked in IT for 15 years, 10 of which have been spent at Microsoft across natural language, search, and SharePoint and Office development. Fox has authored many articles and books, including the recently released "Beginning SharePoint 2010 Development" (Wrox, 2010).

THANKS to the following technical expert for reviewing this article:
Paul Stubbs

TeeChart

Charts

Maps

Gauges

Endless

Charting for Developers

Charts, Maps and Gauges for Microsoft.NET (WinForms, Pocket, WebForms and Ajax), for Delphi/C++Builder, as native Java, ActiveX or PHP.

Scalable Multithreaded Programming with Tasks

Ron Fosner

PCs are evolving away from faster and faster processors and toward more and more cores. That means increases in latent processing power are available at relatively low cost. But it also means programming these systems to take advantage of that latent processing power is more challenging. To use all of those multiple processors, you need to delve into the world of parallel processing.

There are many different ways to distribute your work across multiple cores. In the October issue of *MSDN Magazine* (msdn.microsoft.com/magazine/gg232758), I introduced you to some basic concepts of multithreaded programming and showed how to add threaded execution into your code with OpenMP and thread pools. I also demonstrated how to use tools in Visual Studio 2010 to measure core and thread utilization as a measure of how well a threading implementation improves the performance of your app.

In this article, I'll look at a more sophisticated multithreading technique called task-based programming. Tasks let you spread your application's work across some or all of the CPU cores that are available. With a bit of thoughtful programming you can minimize and even eliminate any data dependency or time-synchronization constraints.

Building on what you learned from my previous article, I'll take you through a more sophisticated multithreaded application that uses tasks. Tasks enable the application to scale itself up to the number of cores available and the amount of work that needs to be accomplished.

A Mouse in a Maze

When I sat down to write this article, I tried to come up with a problem that was complicated to parallelize but still easy to visualize what was occurring. I hit upon the idea of creating a solution that would solve a 2D maze. While at first glance it may seem a bit trivial, it's actually quite challenging to implement correctly—I know because it took me three tries to get it right.

Figure 1 shows a simple, serial maze solver in action. The solution to the maze is just a long, sinuous pathway with many branches that lead to dead ends. Not surprisingly, I called the solution algorithm a "mouse." The mouse will observe its current cell and try to go forward to the next cell. If it hits a wall it will try to go left. If it can't go left it will try to go right. If it can't go into any of those directions, it will mark its current path as a dead end and back up.

When the mouse moves into a new cell it makes note of any pathways that it didn't take. For example, if a mouse is able to move forward, but it was also able to go left, then it will remember that cell and that direction. So as the mouse moves down a hallway it will take note of doorways on either side and push these on a stack. When the mouse reaches a dead end it pops one of these locations, backs up, and heads off in the saved direction. Eventually it will reach the endpoint though sheer persistence.

After a mouse has backed up, it's necessary to prevent it from trying a direction it has already searched. I do this by marking a cell as visited when a mouse has successfully moved into a cell. So

This article discusses:

- Parallelizing problems
- Task queues and dependencies
- Using thread pools
- A custom task class wrapper

Technologies discussed:

Visual Studio 2010, Task Parallel Library, Parallel Patterns Library

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!



NetAdvantage®
for Silverlight Data Visualization



NetAdvantage®
for WPF Data Visualization

Infragistics

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

when a mouse attempts to move in a new direction, it first checks to see that there's no wall. If there's no wall, it checks to see if the cell that it's considering moving into has been visited before. It will discard any movement into cells that have already been visited. This is illustrated by the grayed out pathways in **Figure 1**.

The Mouse Solution

This solution is pretty easy to visualize and, as a result, it's easy to understand the logic behind the search. It's also somewhat hypnotic to watch—at least for a little while. A simplified flow chart for a serial mouse is shown in **Figure 2**.

While it's very simple to understand the problem conceptually, there are some unconstrained elements that make it challenging. First, you have no idea how long a mouse will run before it hits a dead end. Second, you have no idea how many branches it will discover along the way.

This problem is doubly interesting when you try running it on multiple threads. The most straightforward way to make this problem multi-core friendly is to make multiple mice and give each mouse its own thread—which is the approach I've taken. As an added bonus, this enhances the visualization because I can switch the active mouse color as a new thread takes over.

In fact, it was a bit more challenging than I originally considered. Once I had a working single-threaded version, I made the mistake of trying to adapt that version and make it multithreaded. This was my single biggest architectural mistake. I went through three different revisions before I stood back and reworked the architecture to be task-friendly.

I won't cover my failed efforts except to state that they all focused on me trying to optimize performance by not copying data and by trying to minimize memory and optimize access to the shared data by the various threads. Essentially, in my original design, I

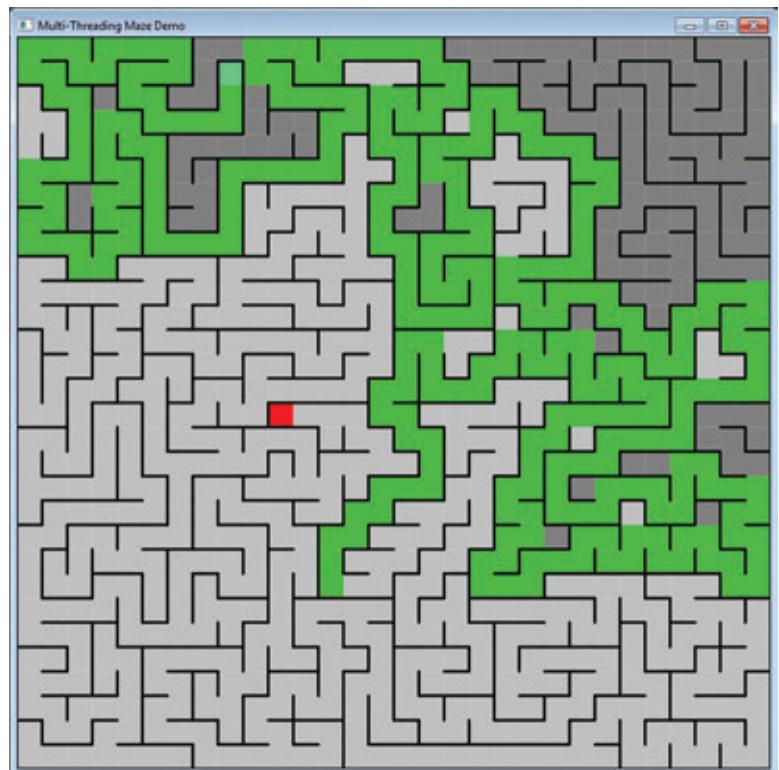


Figure 1 Single-Threaded Maze

When you find yourself adding in mid-state information in a multithreaded program to make up for some partial state you're starting with, or special casing behavior, or in general doing something that's not generic to your tasking code, then it's time to rethink your design.

What I ended up doing was placing the current path information in each mouse and making this part of the mouse's initialization information. When I reached a branch point, I created a mouse data structure and initialized it from the current mouse's information—thus creating a clone mouse that, for example, goes left when the original goes right. The clone contains the original's memory. The only thing different in each mouse is a counter that keeps track of the number of mice created—this is how the mice get assigned a color.

I also turned about and made one global copy of the maze that contains the individual cells' state information and did *not* place any locks around the writing of the state information. This was a simplification I accepted as a tradeoff—a mouse will be working on a path by itself. It always checks to see whether the cell is marked as visited before it moves into the cell.

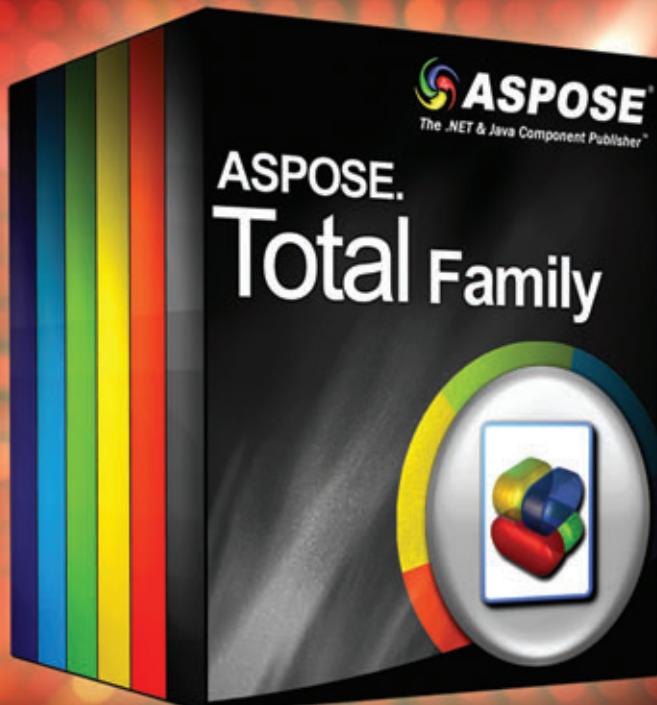
Because there are no locks around the global cell data, it's possible, though unlikely, that two mice might both start down a path at the same time. This could happen either through a duplicate stack entry or through a looped pathway where two mice run into each other. In either case, I accept the fact that a mouse might be happily running down a path, its thread might get suspended, and when it resumes it discovers that some other mouse has crossed its path. In this case the mouse just backs up as if it has hit a wall, because the mouse that crossed it is follow-

Tasks let you spread
your application's work across
some or all of the CPU cores that
are available.

had a global stack that I could lock, then I'd push the locations of the untaken branches onto the stack as I ran across them. When a thread finished work and went in search of more work to process, I would lock the stack (to prevent simultaneous access by another thread), pop off the location and direction, then unlock the stack. While this worked to some degree, it was clunky and forced me to consider adding in new data in each mouse to keep track of the path taken so that it would have a path from the starting point to its current location.



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



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

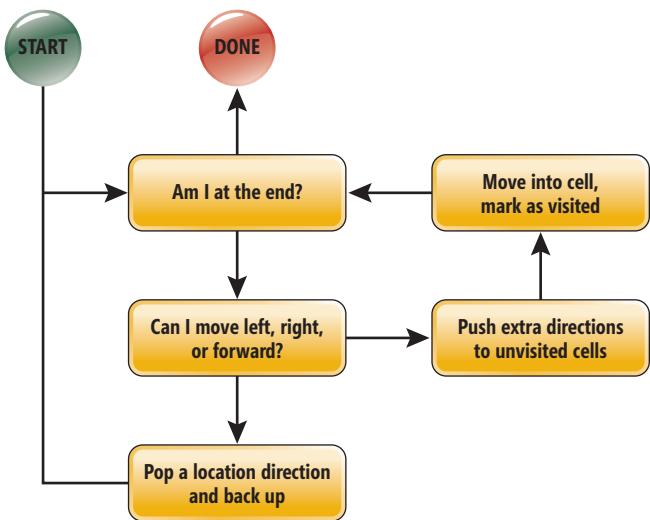


Figure 2 Flow Chart of Mouse Actions

ing a path successfully. The suspended mouse missed its chance and did some extra work.

If there were a lot more processing involved in marking the cells, then I might be more reluctant to accept the fact that some useless work might get done. Eliminating any locks of shared data simply means I have to make the algorithm a bit more robust. Designing it to handle such situations means that there's less room to make mistakes. The most likely source of errors in multithreaded programs usually involves some form of locking mistake, such as a race condition or making assumptions about when data or counters get updated.

I chose to use the most generic of the multithreading algorithms: tasks that can have dependencies.

If you can make your algorithms robust enough to handle data that might be slightly out of date and able to recover from those situations, then you're on your way to making a resilient multithreaded architecture.

Task Queues and Dependencies

Windows Vista introduced new scheduling algorithms and new primitives that are the underlying support for a number of Microsoft .NET Framework 4 features. One of these features is the Task Parallel Library (TPL), which provides quite a few of the more common parallel programming algorithms, including fork/join, parallel for, work stealing and task inlining. If you're coding in unmanaged C++, you can take advantage of Intel Threading Building Blocks (TBB) or the Microsoft Parallel Patterns Library (PPL).

These libraries come with classes that provide multithreaded programming support for jobs and tasks. They also have many thread-safe container classes. These classes have been tested and optimized for performance, so unless you've got a deep-seated need to write a customized variation for some reason, you'll be better off using some tested, robust code.

Because this is an introductory article on threading and tasks, and you might benefit from some insight into how these threading libraries work, I wrote my own set of wrappers around two of the new features in Windows Vista: ThreadPool and SlimReaderWriterLock (SRWLock). These are both low-cost ways of making data threads safe for those situations where you've got one writer and many readers, and the data usually isn't locked for a long time. Note that the goal of this article is to step you through how I've chosen to implement a thread pool that consumes tasks—tasks which can have dependencies. To illustrate the basic mechanisms I've taken some liberties with the code to make it easier to understand. The code works, but you're better off choosing one of the threading libraries for any real implementation.

For my maze algorithm I chose to use the most generic of the multithreading algorithms: tasks that can have dependencies (implemented using SRWLock) and a task scheduler (using the OS's ThreadPool). This is the most generic because basically a task is just some bit of work that needs to get done and the task scheduler takes care of communication with the OS to get the task running on a thread. They're generic because it's possible to create tasks out of any code that needs to get executed and throw it into a ThreadPool.

The challenge is creating tasks that take up enough time to make overcoming the overhead of getting them scheduled worthwhile. If you've got some big monolithic tasks that need to get executed, then feel free to create some threads and execute the code on them. On the other hand, there are many applications where there are groups of tasks that need to get done, some serially, some not. Sometimes you'll know beforehand how much work needs to get done; other times—particularly fetching or reacting to user input or some communication—you're just polling for something that will only occasionally require extended processing. This is easily handled by the generic nature of ThreadPool and its associated task queue.

Customizing the ThreadPool

To give you a clear understanding of how to build a tasking system on top of the ThreadPool, I really only need to use three of its interfaces:

```

CreateThreadpool();
CloseThreadpool();
TrySubmitThreadpoolCallback();

```

The first two are just the bookkeeping functions. TrySubmitThreadpoolCallback basically takes a pointer to a function to execute plus some context variables. You call this function repeatedly to load up the thread pool with tasks to execute and it will serve them in a first-in-first-out (FIFO) manner (no guarantees here).

To make it work with my tasks, I wrote a short wrapper about ThreadPool that lets me customize the number of threads in the thread pool (see **Figure 3**). I also wrote a submit function that will take care of tracking the context variables associated with the task.

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

Figure 3 ThreadPool Wrapper

```
class ThreadPool {
    PTP_POOL m_Pool;
public:
    static VOID CALLBACK WorkCallback(
        PTP_CALLBACK_INSTANCE instance,
        void* Context);
    ThreadPool(void);
    ~ThreadPool(void);

    static unsigned GetHardwareThreadsCount();

    // create thread pool that has optimal
    // number of threads for current hardware
    bool create() {
        DWORD tc = GetHardwareThreadsCount();
        return create(tc,tc);
    }
    bool create(
        unsigned int minThreads,
        unsigned int maxThreads = 0);
    void release();
    bool submit(ITask* pWork);
};
```

The interesting thing to note is that you usually only want to create as many software threads as there are hardware threads. Sometimes this might be one less if you've got a main thread off doing something else. Note that the number of threads you create has nothing to do with the size of the task queue—it's perfectly legitimate to create a ThreadPool with four threads and then submit hundreds of tasks. It's probably not a good idea, however, to take a single serial job and break it up into thousands of tasks. This is an indication that you've got too many finely grained tasks. If you find yourself in this situation, then you'll just have create a task whose job it is schedule the next 100 tasks—or if you're using one of the tasking libraries, then create a work-stealing, inlining or follow-on task.

My Task class (note the capital T, which I'll use to denote my task wrapper class as shown in **Figure 4**) has the capability of being dependent upon other Tasks. Since the OS's ThreadPool doesn't have this capability, I'll need to add it. Thus, when a Task starts executing on a thread, the first thing it does is check to make sure

Figure 4 Task Wrapper

```
class ITask {
protected:
    vector<ITask*> m_dependentsList; // Those waiting for me
    ThreadSafe_Int32 m_dependenciesRemaining; // Those I'm waiting for

    // The blocking event if we have dependencies waiting on
    SRWLOCK m_SRWLock;
    CONDITION_VARIABLE m_Dependencies;

    void SignalDependentsIMDone();
    ITask();
    virtual ~ITask();

public:
    void blockIfDependenciesArePending();
    void isDependentUpon(ITask* pTask);

    unsigned int queryNumberDependentsRemaining()

    // A parent Task will call this
    void clearOneDependency();
    virtual void doWork(void*) = 0;
    virtual void* context() = 0;
};
```

it has no outstanding dependencies. If it does, the thread executing the task blocks by waiting on the SRWLock. The Task will only get rescheduled when the SRWLock is freed.

Again, let me point out that this is not code I'd want to see in a non-academic application, but putting the block here lets you see exactly what's happening. The OS will notice the block and schedule another Task. Eventually—unless there's a programming error—the blocked task will unblock and get rescheduled to run.

Generally it's not a good idea
to schedule a Task that will
immediately get suspended.

Generally, it's not a good idea to schedule a Task that will immediately get suspended. Because ThreadPool's task queue is by and large FIFO, you'll want to schedule tasks that have no dependencies first. If I were writing this for optimal performance rather than illustrative purposes, I'd add a layer that only submitted tasks that had no dependencies to the thread pool. I can get away with this because blocked threads will eventually get swapped out. In any case, you'll need to have some thread-safe way of signaling that a task is done and SRWLocks can be used for this situation. Incorporating them into my Task class is natural, rather than having to write specialty code to handle each case.

By design a Task can have any number of Task dependencies. Ordinarily, you want to reduce or eliminate any waiting if you can, and using a tool such as Visual Studio Task List or Intel Graphics Performance Analyzers will help you track these down. The implementation I present here is a very basic tasking system and should not be used for code that requires high performance. It's good sandbox code for getting your multithreaded feet wet, but you should look toward TBB, TPL or PPL for more efficient code.

The ThreadPool will call the WorkCallback function, which executes some prefix code that will query the Task data structure, like so:

```
VOID CALLBACK ThreadPool::WorkCallback(
    PTP_CALLBACK_INSTANCE instance, void* pTask) {

    ITask * pCurrentTask = (ITask*) pTask;
    pCurrentTask->blockIfDependenciesArePending( );
    pCurrentTask->doWork(pCurrentTask->context( ) );
    pCurrentTask->SignalDependentsIMDone();
}
```

The basic operation is:

1. The ThreadPool loads up the WorkCallback from its internal task queue.
2. The code queries the Task to see if there are any dependencies (parent dependencies). If dependencies are found, block execution.
3. Once there are no dependencies, call doWork, which is the actual part of the Task code that's unique per Task.
4. Upon returning from doWork, clear any child dependencies on this Task.

The important thing to note is there is some preamble and postscript code residing in my ThreadPool class that checks and clears



Spread for Windows Forms | from \$959.04



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



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



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



Word processing components for Visual Studio .NET.

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



FusionCharts | from \$195.02



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

- Liven up your web applications using animated & data-driven charts
- Create AJAX-enabled charts with drill-down capabilities in minutes
- Export charts as images/PDF and data as CSV from charts itself
- Create gauges, dashboards, financial charts and over 550 maps
- Trusted by over 17,000 customers and 330,000 users in 110 countries

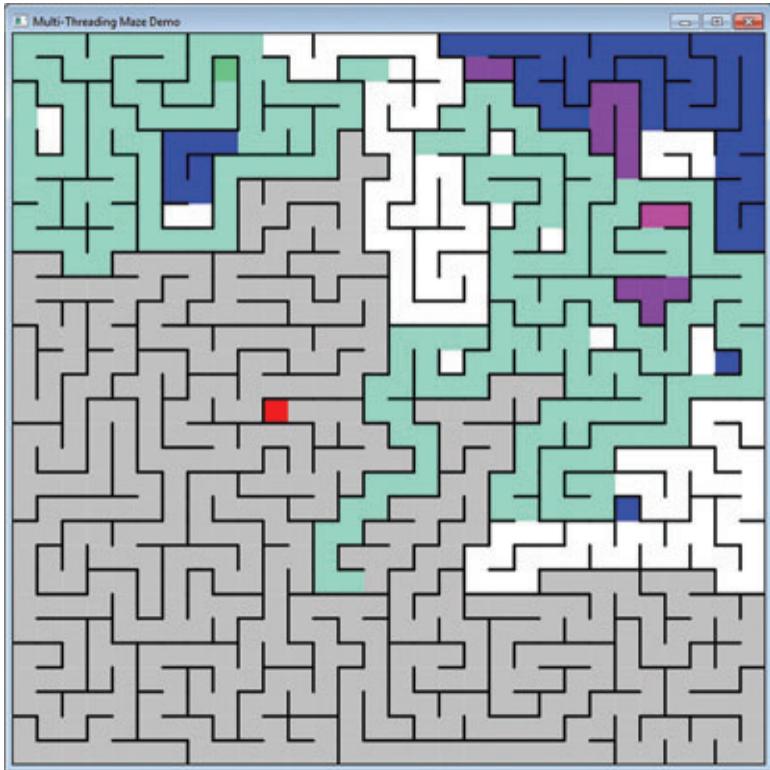


Figure 5 Solving the Original Long-Maze Algorithm

dependencies in the Task. This code gets run on each thread but has a unique Task associated with it. Once the preamble code gets run, the actual Task work function gets called.

Creating a Custom Task Class Wrapper

The basic job of a task is to provide some context and a function pointer to get the task executed by the thread pool. Because I wanted to create Tasks that were capable of having dependencies, I needed some code to handle the bookkeeping of tracking blocking, and unblocking dependencies (see Figure 4).

When a Task is created you can then tell it that it's dependent upon other tasks by providing a pointer to that dependant Task. A Task contains a counter for the number of Tasks that it has to wait for, plus an array of pointers to Tasks that have to wait for it.

When a Task has no dependants, its work function will get called. After the work function returns, it loops through all the Task pointers in the array, calling clearOneDependency on each class, which decrements the number of that Task's remaining dependencies. When the number of dependencies drops to zero, the SRWLock is released, unblocking the thread that executed the task waiting on those dependencies. The thread running the task gets unblocked and execution will continue.

That's the basic overview of how I designed my Task and Thread-Pool classes. I ended up designing it this way because the OS's native thread pool doesn't have quite this behavior and I wanted to give you some code to play with where you're in control of the dependency mechanism. I originally had a much more complicated wrapper around ThreadPool that included a priority queue, but realized that I was unnecessarily complicating things and that

a simple child-parent dependency relationship was all I needed. If you really want to take a look at customizing a thread scheduler, take a look at Joe Duffy's article, "Building a Custom Thread Pool (Part 2): A Work Stealing Queue," at tinyurl.com/36k4jcy.

I tend to write code pretty defensively. I tend to write simple implementations that work, then refactor and increase functionality with frequent checks along the way to make sure I haven't messed anything up. Unfortunately, I also tend to write code that passes around references to other variables—which is a bad thing in a multithreaded program if you're not careful.

I was undone more than once when converting my single-threaded maze solution code to a multithreaded one. I finally had to go through and make sure I was passing copies of data when there was a chance that the value would be modified in a thread.

I also tried to be conservative by starting with a single-threaded version that only kept the current mouse's path. That introduced the problem of keeping track of yet more state data. As mentioned earlier, I solved this by making clone mice that had all their parents' data. I also chose to eliminate the prioritized ThreadPool wrapper and any locking on the global maze cell data. In all likelihood I introduced some additional work, but I also eliminated many of the sources of errors that could have occurred by greatly simplifying my code.

The ThreadPool wrapper and the Task class worked exactly as designed. I utilized these classes in some unit tests to make sure they exhibited the behavior I was expecting. I also instrumented them using the Intel Graphics Performance Analyzers tasking tool, which has a feature that lets you dynamically tag threads and examine which pieces of code are executing on a particular thread. This visualization of thread execution let me verify the threads were executing, blocking and being rescheduled, just as I expected.

I was undone more than once
when converting my single-
threaded maze solution code to
a multithreaded one.

When I reworked the mice to be clones of their parents, this ended up greatly simplifying the bookkeeping required by the simulation because each mouse was self-contained. The only shared data I ended up requiring was the global cell array, which indicates if a cell was visited. I can't emphasize enough that having a good visualization on how your tasks are being scheduled is of paramount importance.

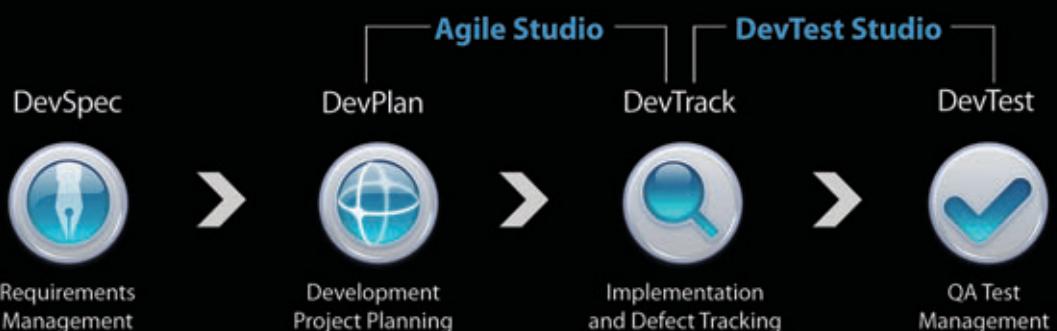
The Mouse Pool

I chose the maze problem because it illustrated a number of issues that can crop up in converting a single-threaded algorithm to a

Gain Full Traceability

Requirements Driven Quality Management

DevSuite



A Modular Approach to ALM

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

Agile Ready

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



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

www.techexcel.com

1-800-439-7782

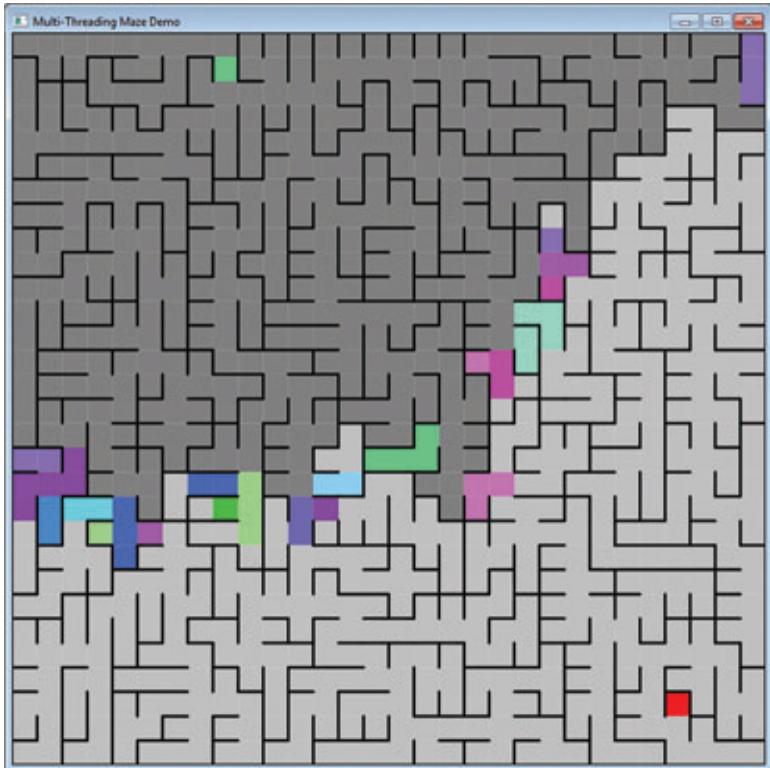


Figure 6 Multiple Mice Make It Easy to Solve the Maze in Much Less Time

multithreaded one. The biggest surprise to me was that, once I bit the bullet and rewrote the algorithm to eliminate some of the book-keeping I had been attempting to maintain, the maze-solving algorithm suddenly became much simpler. In fact, it became simpler than the single-threaded algorithm because there was no need to keep a stack of branch points—they were simply spawned off into a new mouse.

By design, each mouse was a clone of its parent, so each mouse had an inherited path tracing back to the spawn point. The mice didn't know it, but I purposely wrote the maze-generation algorithms to try to select the furthest path possible. No sense in making it easy on them. The test program allows selection between a number of maze-generation algorithms, ranging from the original algorithm—which generates long corridors with occasional branches eventually leading to dead ends—to algorithms that are very branchy with short corridors. Presented with these different mazes, the difference in behavior from the single-threaded solution to the multithreaded solution can be quite dramatic.

When I applied a multi-threaded method to solving the original maze algorithm, I reduced the search duration by 48 percent on a 4 CPU system. This is because the algorithm has a lot of long corridors and there aren't a lot of opportunities to spawn off additional mice (see **Figure 5**).

Figure 6 shows a maze with more branches. Now there are a lot more opportunities to spawn off mice and have them search simultaneously. **Figure 6** shows the multi-threaded solution to this short-path maze, where I get a reduction of the time to find a solution by 95 percent by using more tasks.

This just serves to illustrate that some problems are more amenable to breaking apart than others. I feel compelled to point out that the

maze program is designed to be visually interesting—that is, it lets you see the progression of the mice and steps through them. If I were interested in simply finding the solution to the maze in the shortest amount of time, the rendering and the mice would be decoupled—but then that would not be as fun to watch.

Loose Threads

One of the biggest problems that I see when I help folks try to make their applications run faster is a hesitation to try multithreading. I understand that hesitation. When you add in multithreading, you suddenly add in a layer of complexity that most programmers aren't used to in an area they don't have a lot of experience with.

Unfortunately, when you shy away from multithreading, you end up leaving a good portion of computing power unutilized.

I've covered the basics of how a tasking system works and given you the basics of how to go about breaking up large jobs into tasks. However, the current approach, while good, is not the best practice for getting the maximum performance across current and future multi-core hardware. If you're interested in getting further performance gains on any hardware that your application may be run on, then you'll need to architect your application with this in mind.

The best way of getting maximum, scalable performance in your application is to take advantage of one of the existing parallel libraries and see the best way to fit your application's needs into the various architectures that these libraries provide. Unmanaged, real-time or performance-critical applications are usually best served by using one of the interfaces provided in TBB, while managed apps have a wider variety of multithreading options in the .NET Framework 4. In either case, choosing one of these threading APIs will determine the overall structure of your application and how you design the tasks to work and coexist.

In a future article I'll take a look at actual implementations that take advantage of these techniques, and demonstrate how to construct applications around these various threading libraries so you can design your own implementations to use them.

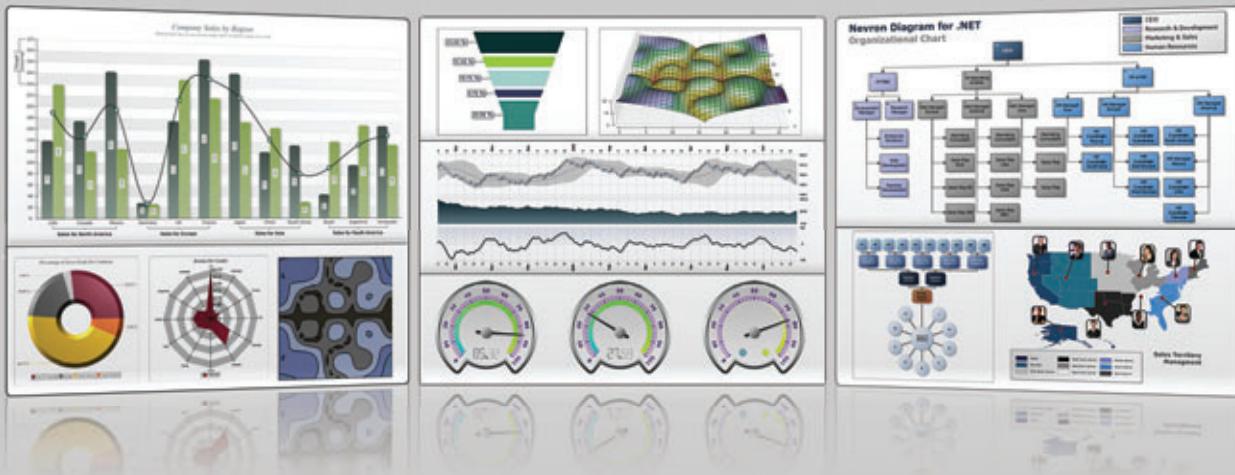
In any event, you've now got the basic knowledge to try out some basic threading approaches, and you should take a look at the threading libraries to start to figure out how best to design your future applications. While multithreading can be challenging, using one of these scalable libraries is the gateway to taking advantage of maximum performance of hardware, both current and future. ■

RON FOSNER has been optimizing high-performance applications and games on Windows for years and is starting to get the hang of it. He's a graphics and optimization expert at Intel and is happiest when he sees all CPU cores running flat out. You can reach him at Ron@directx.com.

THANKS to the following technical experts for reviewing this article:
Aaron Coday, Orion Granatir and Brad Werth

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.



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

A Coder's Guide to Writing API Documentation

Peter Gruenbaum

Ever been in a situation where your manager asks you to write documentation for the APIs that you developed? Let's face it, if you're like most developers, you love to code and hate to write. Furthermore, writing takes time away from critical tasks you need to do, such as feature development and bug fixing.

It's no surprise that API documentation often ends up being frustrating and confusing for the reader—it rarely gets the attention it deserves.

This article is a guide on how to write API documentation. I'll describe the most important components of API documentation and provide some suggestions about how to make it effective. I'll also give you some hints for creating good overviews, sample code and reference material, including where you should focus your time and attention to get the best effect.

Why Document Your APIs?

Let's start with the non-technical side of the issue. API documentation has been around ever since the first programming languages

were created. There's been plenty of time to develop effective processes for creating quality documentation, yet well-written API documentation is still quite rare. Why doesn't it happen?

First, documentation is seldom prioritized. Even though it has a large impact on how much a software platform is adopted, the actual impact of documentation is difficult to measure. As a result, documentation is hardly ever given enough time and budget. When developers are asked to write documentation, it's typically on top of their other responsibilities, and they must somehow fit it into their already overloaded schedule.

Second, developing code and writing documentation are two different skills. Sometimes developers are asked to write in a language that's not their first language. But even if they were born in an English-speaking region and are asked to write in English, there's a good chance that they struggled through their literature and social studies classes in school, and would've much rather spent that time solving problems in their math and science classes.

The first step in creating good API documentation is to ask management for the time and budget to do it well. There are two main points to make to managers:

1. Good documentation can increase the likelihood of platform adoption because it means a less-frustrating experience for developers.
2. Good documentation reduces the cost of support because developers can find the answers to their questions more easily.

Arguing the importance of good documentation may be a challenge for you if you don't enjoy writing or if you're completely

This article discusses:

- The importance of API docs
- API documentation components
- Writing useful sample code
- Reference docs and Web APIs

Technologies discussed:

API Documentation, Best Practices

Figure 1 Sample Code Example

```
/// <summary>
/// Returns an array of user IDs for users that
/// are connected to the specified user. Note that
/// this is a simple, synchronous way to obtain the data.
/// </summary>
/// <param name="userId">The ID of the specified user.</param>
/// <returns>An array of user IDs that are connected to
/// the specified user.</returns>

public int[] GetConnectedUserIds(int userId) {
    // Create variable to hold the returned IDs
    int[] connectedIds;

    // Construct a URL using the userId and the authorization token
    string url =
        "http://webservices.contoso.com/users/connections?userid=" +
        userId.ToString() +
        "&token=" +
        authorizationToken;

    // Create the Web request using the url
    HttpWebRequest request =
        WebRequest.Create(url) as HttpWebRequest;

    // Get the response
    using (HttpWebResponse response =
        request.GetResponse() as HttpWebResponse) {

        // Read the response XML
        Streamreader reader =
            new Streamreader(response.GetResponseStream());
        string xmlResponse = reader.ReadToEnd();

        // Process XML to extract user IDs for connected users
        // and responseStatus
        ...

        if (responseStatus != "ok") {
            // Handle errors here
            ...
        }

        reader.Close();
    }

    return connectedIds;
}
```

overloaded with work, but there is an alternative. If there's enough budget, you can hire a technical writer who will gather information from you and write the documentation.

As with developers, you'll find technical writers with a range of experience and expertise. Many technical writers are more experienced in end-user documentation and support. For API documentation, however, you'll want to find one who's actually spent time as a software developer. In many companies, these types of writers have titles like programmer/writer.

Technical writers who have some coding experience under their belt understand the pain that developers go through in trying to get a software platform to work, and how good documentation can help improve the development of the platform. Also, they should have enough specialized knowledge of languages, algorithms and patterns to read your code and understand your libraries. With this knowledge and experience, technical discussion between the writer and development team will be more straightforward and productive.

That said, if the budget doesn't allow you to bring on a technical writer, then you'll need to write the documentation yourself. Make sure management understands that you need to carve out time to do this, just as you would for creating a new feature.

Components of API Documentation

There are four components of good API documentation:

1. **Overview** Explain what advantages developers have in using the platform, and in some cases, provide an architectural description of the platform.
2. **Getting started** Help the developer get started, in the form of step-by-step tutorials or simpler walkthroughs.
3. **Sample code** Provide well-commented code samples that developers can build on.
4. **Reference material** Provide detailed information about each class, member, function or XML element.

When developers first begin to read about an API, the first piece of information they need to know is: who would use the API and why they would use it. If the developers don't understand this, then they'll quickly move on to use something else. Unfortunately, this information often tends to be forgotten. To the people developing the API, the information is obvious—but to others, it's not.

Come up with clear examples of when you would use the API. If you've got existing customers, or even potential customers, then use those as real-world examples. List the advantages of the software platform, ideally contrasting it to existing approaches. You'll find that project managers often have this kind of information.

Overviews are also a good place to explain the API's overall architecture. For some types of APIs (for example, many Web APIs), the API is simple enough that an architecture discussion is not necessary. However, if you're documenting something complex, with many classes and an inheritance structure, then a full discussion of the architecture, along with accompanying diagrams, is often helpful for developers to understand it.

Getting Started

Once developers have decided to give your API a try, the first thing they'll want to know is how to get started. In 2009, my company (SDK Bridge LLC) ran a survey on documentation, and one of the most common threads running through the responses was that developers wanted help in getting started. (See my article "Survey on SDK Documentation" at tinyurl.com/35l66yk.) This is critical for adoption: if developers find it difficult to get started, they'll give up quickly and find another way to accomplish their goals.

Good documentation can
increase the likelihood of
platform adoption.

A great way to get developers started is through tutorials. This approach is often much more effective than descriptive text or architectural diagrams. A tutorial leads a developer step-by-step through the process of creating a simple application that demonstrates how the API works. It often starts with non-programming activities, such as setting up your development environment or obtaining authorization credentials. Then it directs the developer to gradually add code until they can demonstrate a simple task

from the API. If possible, try to structure your tutorial so that developers have something they can run and see results quickly. Then continue the tutorial, adding more features.

Chances are, you've worked so closely with your API that you've forgotten what it's like to come at it from a completely new perspective. As you work on this section, do your best to take a step back and put yourself in the shoes of a newcomer.

Good documentation reduces the cost of support.

Writing Sample Code

Another common thread in the SDK Bridge survey responses was the importance of good sample code. Developers learn a new platform by starting with code that they know already works, and then modifying it or adding to it. A lot, if not most, developers find it easier to learn by doing than by reading.

You probably already know how to create good production code. Good sample code shares some similarities to good production code, but there are some key differences as well. In general, good sample code should follow these guidelines:

1. Relevant information should be grouped together.
2. Clarity is more important than efficiency or robustness.
3. Simplicity is more important than a good-looking UI.

You can apply these guidelines to specific areas of software and see how sample code compares to production code.

Every programmer knows that they should never use hard-coded values in their code. Those values should be turned into constants and put somewhere that's easy to find in case someone wants to change them.

It turns out that that's true of production code, but not true for sample code. You should use hard-coded values in sample code in order to group all relevant information as closely together as possible. If you follow good practices for production code and define all of your constants at the top of your file, then when developers look at the line of code that uses the constant, they have to scroll to the top of the file to find out what its value is. That simple action can make them lose their train of thought. Strings, integers, hexadecimal values, and other simple values should all be hard-coded right where they're used.

Comments are good for both production code and sample code, but in sample code they're critical. Every class, member or function should start with at least one comment line explaining what it is or

what it does. You should use comments anywhere the code is not obvious, especially if you need to document a work-around or something equally unusual. These comments can be several lines long if required. Use complete sentences and don't be afraid to be wordy.

In general, you should have at least a line of comment for every five or 10 lines of code. However, there are some exceptions to this guideline. Code that's extraneous to what you're demonstrating doesn't need so many comments (for example, UI code that's required to display the results of your API). If you're writing a small snippet with just a few lines of code that's included in some reference material, you might not need comments at all. If you're providing a very large sample that's more like production code, then it may be more practical to lessen the number of lines of comments.

Variable, class, member and function names should be clear, regardless of whether you're writing production code or sample code. In sample code, though, you should take this idea farther than in production code because clarity is more important than efficiency. Long, unwieldy names can be a problem in production code, but they're usually worth it in sample code because of the added clarity. Try to make even the smallest variable name have meaning, and do not—no matter how much you're tempted—use meaningless variable names like "foo" or one-letter names.

A good way to get developers started is through tutorials.

Object-oriented programming is one of software engineering's best inventions. You may be surprised to learn that although it's highly desirable for production code, it's in fact generally not desirable for sample code. The reason is that object-oriented design distributes functionality so that data and functions are grouped together, and it uses inheritance to cut down on duplicate code. Remember, one of the fundamental principles of good sample code is that relevant information should be grouped together. Object-oriented code tends to distribute the relevant information among various classes. Therefore, developers may end up searching through an inheritance hierarchy for what a method does, which only serves to waste time and break their train of thought.

There are exceptions, of course. Some APIs require object-oriented programming to function properly. Very large samples that are more like a production application may also need object-

Figure 2 Reference Documentation Style

Type	Guideline	Examples
Class	Start with a word like "Represents"	"Represents a user's photo album."
Methods and functions	Start with a verb	"Returns the number of contacts for the specified area." "Pauses the video."
Properties	Use a noun or start with verbs such as "Gets" or "Gets and sets"	"The user's tasks." "Gets and sets a collection of the user's tasks."
Events	Start with a phrase such as "Raised when" or "Occurs when"	"Raised when the response from server is received."
XML elements	Use a noun-based phrase	"The city's postal code."
Boolean values	For Boolean properties, start with "Indicates whether"; for Boolean return values on methods and functions, start with "Returns whether"	"Indicates whether the control is visible." "Returns whether two regions intersect."

WORD PROCESSING COMPONENTS

MILES BEYOND RICH TEXT



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



The logo for Text Control features the word "TEXT" in a large, bold, black sans-serif font. Above the "T", there is a stylized, italicized "TX" in a light gray color. To the right of "TEXT", the word "CONTROL" is written in a large, bold, black sans-serif font. A registered trademark symbol (®) is positioned at the top right of "CONTROL". Below "CONTROL", the words "word processing components" are written in a smaller, regular black sans-serif font.



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

oriented design. Just be mindful that the user wants to see all the necessary information in one class if at all possible.

A basic rule of good software design is to encapsulate functionality in functions and methods. For production code, this adds clarity and reduces duplicate code. It's also good for sample code, because it often can create a block of code that developers can simply copy and paste into their own code, making it easy to use.

Occasionally, sample code requires a large number of lines of code that are not directly relevant to your API, but that you need in order to get your sample to run. In this case, it's a good idea to try to encapsulate those irrelevant lines of code into a function or method so that developers can more easily ignore them.

Unless your API specifically provides UI capabilities that you need to demonstrate, you should keep the UI elements in your sample code as simple as possible. UI code can take up a lot of space and dilute the important lines of code that you want to be demonstrating. Developers aren't concerned about whether your sample looks slick—they just want to be able to understand how your API works.

If you absolutely have to have a large number of lines of code for your UI, then package that code up into separate functions that are easy for developers to scan or ignore.

Finally, while exception handling is critical for production code to work well, in sample code it can dilute the relevant code and create a distraction. Often, a good solution is to not have exception handling, but to put in a comment indicating what kind of exceptions to handle in production code. However, there are situations where certain calls should always be made with exception handling, and in these situations, it's worth the extra lines of code to show exactly how that exception handling should work.

Figure 1 shows an example of a function from sample code that demonstrates how to make a REST request in C# for a social networking site, returning the user IDs of the users who are connected to the specified user. In production code, the REST endpoint URL would be stored as a constant along with other relevant URLs. In sample code, however, it's best to put this information where developers are most likely to see it and make the connection to its role in the function. Note also that error handling is suggested, but not implemented. The XML processing has been removed from this example for brevity.

Reference Material

Reference material typically makes up the bulk of the API documentation. For each class, member, function, XML element and so on, there needs to be detailed information about what it is and how it's used. At a minimum, reference material should cover:

- A short description
- Description of any parameters and return values
- Any important remarks that will assist the developer

If there's more time and budget, add this information:

- Exceptions that may need to be caught
- Links to other related overview or reference topics
- A snippet of sample code, ideally from the sample code you've already written

Good reference documentation has a consistent style throughout. Sometimes style guidelines will exist already, but often you're

Figure 3 Reference Documentation Example

Class or Member	Type	Description
Class description	Class	Represents a Windows button control.
Button constructor	Constructor	Initializes a new instance of the Button class.
Focus	Method	Sets input focus to the control.
Visible	Property	Gets or sets a value indicating whether the control and all its child controls are displayed.
Click	Event	Occurs when the control is clicked.

on your own to figure them out. **Figure 2** lays out some general guidelines for the short descriptions.

As an example, consider the descriptions shown in **Figure 3** for the Button class from the Microsoft .NET Framework. This is taken directly from the SDK documentation on MSDN.

Web APIs

The number of Web APIs has been growing rapidly over the past few years, and so it's worth thinking about how Web APIs are different from local APIs. Software as a Service is becoming a popular business model, and companies are quickly finding that their larger customers want to be able to use their services directly from their own systems. This means that the services provider needs to have a public API that their customers can call.

(A note on terminology: I use the term "local API" to describe the typical kind of API that existed before the Web. Technically, these APIs can be Remote Procedure Calls, therefore they're not local; technically, Web APIs can be called on a server that's the same computer as the client, therefore they are local. However, in most instances, Web APIs, which use standard protocols such as HTTP, are used remotely and other APIs are used locally.)

Developers learn a new platform
by starting with code that they
know already works.

Because Web APIs are relatively new, there's no standardization on how their documentation looks. The quality of Web API documentation varies dramatically—sometimes it's well-organized and complete, and sometimes it's the bare-minimum information thrown up on a wiki. If you're going to be writing Web API documentation, it's worth spending some time looking at how various companies have documented their APIs so that you can find a good template to follow. For example, Twilio, a platform for voice and messaging applications, has an excellent example of REST documentation, which can be found at twilio.com/docs. Hopefully, over time, the industry will settle into a small number of effective templates.

In some ways, Web API documentation is more critical than local API documentation because it can be more difficult for developers to experiment with Web APIs in order to figure out how they work.

Does your Team do more than just track bugs?

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.

Free Trial and Single User FreePack™ available at www.alexcorp.com

Track all your project tasks in one database so you can work together to get projects done.

- Quality Control / Compliance Tracking
- Project Management
- End User Accessible Service Desk Portal
- Bugs and Features
- Action Items
- Sales and Marketing
- Help Desk

Native
Smart Card Login
Support including
Government
and DOD

New in Team 2.11

- Full Windows 7 Support
- Windows Single Sign-on
- System Audit Log
- Trend Analysis
- Alternate Display Fields for Data Normalization
- Lookup Table Filters
- XML Export
- Network Optimized for Enterprise Deployment



Desk Tickets - Windows Internet Explorer

Alexsys Team Service Desk

Welcome: John Doe

My Tickets New Ticket Edit Profile Knowledge Base Logout Alexsys Team Service Desk

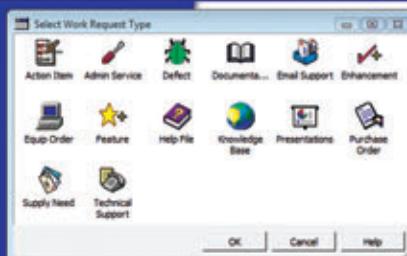
Active Tickets

Ticket No.	Subject	Priority	Ticket Status	Open Date
SD-026554	Test of new ticket	C - Low	Open	Thu Nov 20 14:30:01 EST 2008
SD-0265501	I need help setting up Team-Web	B - Moderate	Open	Fri Oct 17 09:24:32 EDT 2008
SD-0265520	Custom sorts do not apply immediately in Team-Web	B - Moderate	Open	Wed Sep 24 16:24:47 EDT 2008
SD-0265504	Another demo	A - High	Re-Open	Wed Sep 24 11:44:01 EDT 2008
SD-0265528	Second sample ticket with a longer subject	C - Low	Resolved	Tue Sep 23 15:28:23 EDT 2008

Done Internet 100%

Service Desk Features

- Fully Secure
- Unlimited Users Self Registered or Active Directory
- Integrated into Your Web Site
- Fast/AJAX Dynamic Content
- Unlimited Service Desks
- Visual Service Desk Builder



New Action Item - Windows Internet Explorer

New Technical Support - Windows Internet Explorer

New Defect - Windows Internet Explorer

User : Save | Cancel | Getto [] Edit Tables | Refresh | Options | ? Close

New Action Item

Title:
Status: Open
Customer Status:

New Technical Support

Title: ID: #100
How to configure an Exchange Server email account
Status: Open
Customer Status:

New Defect

Title: Status: Open
Customer Status:

Team 2 Features

- Windows and Web Clients
- Multiple Work Request Forms
- Customizable Database
- Point and Click Workflows
- Role Based Security
- Clear Text Database
- Project Trees
- Time Recording
- Notifications and Escalations
- Outlook Integration



Free Trial and Single User FreePack™ available at www.alexcorp.com.

FreePack™ includes a free single user Team Pro and Team-Web license.

Need more help? Give us a call at 1-888-880-ALEX (2539).

Team 2 works with its own standard database, while Team Pro works with Microsoft SQL, MySQL, and Oracle Servers.

Team 2 works with Windows 7/2008/2003/Vista/XP.

Team-Web works with Internet Explorer, Firefox, Netscape, Safari, and Chrome.

Developers may have limitations (quotas) on how many times they can make a call, or their experimentation may be affecting a live system, or it may be difficult to emulate specific conditions, such as when the server is under heavy use.

As mentioned earlier, developers rely heavily on sample code. One of the powerful things about Web APIs is that they're independent of platform and language. Unfortunately, this means extra work when creating sample code. You may find yourself writing sample code in Python, Ruby, Java, C# and so on. Try to find out what your customers use most and focus on the languages that are most important to them.

The two most common technologies for Web APIs are SOAP and REST. SOAP has a definition format (Web Services Description Language, or WSDL) that's a great starting point for reference documentation, whereas REST does not. Sample HTTP calls and XML/JSON files are useful for both of these technologies in illustrating how they work, but they're not sufficient. Samples should be followed by tables that describe each element as well as its data format.

For example, it may not be enough to describe a parameter as a string. Are there special characters it can't handle? Are there limitations on its length? If an XML element is a date, you should specify the format of the date. If it's a time, then you need to specify its time zone.

Also, you'll need to explain how errors are handled. This may vary for the different formats that your API supports. If your API uses HTTP response codes to flag errors, these should be documented. Error documentation should explain why an error occurs and how to fix the problem.

Good reference documentation has a consistent style throughout.

Authentication is often required for Web APIs, and this needs to be documented in detail as well. If developers need API keys, be sure to give them step-by-step instructions on how to obtain these. Also, don't forget that Web APIs are built on top of HTTP, which is an incredibly rich protocol. You may have HTTP-related information that requires documentation, such as caching, content type and status codes.

Web APIs are new enough that we're still in a period of figuring out the best way to document them. Expect to see standardization in the next few years.

Publishing

So far I've been focusing on content, but you'll also need to publish the documentation so that developers can read it. In general, developers expect to see Web-based, hyper-linked documentation rather than flat files such as a PDF. There are several ways to get your documentation on the Web.

If your API is small, it may be simplest just to create HTML files. Use CSS to get the look-and-feel to match your company's Web site.

Wikis provide a structure for more-complex APIs. Wikis also allow you to easily update or add to documentation over time without needing access to other tools or servers. In addition, the group collaboration aspects of wikis enable entire teams—even your users—to contribute. However, slapping together a wiki and hoping your developers and users will write the docs isn't a very viable API documentation strategy.

Developers expect to see Web-based, hyper-linked documentation.

Several free, open source wiki engines are available and are becoming popular for API documentation, such as the PHP-based MediaWiki (mediawiki.org/wiki/MediaWiki) and the PERL-based TWiki (twiki.org).

Commercial documentation tools such as Madcap Flare (see madcapsoftware.com/products/flare) and Adobe RoboHelp (see adobe.com/products/robohelp) are designed primarily for end-user documentation, but can be easily adopted for API documentation. They provide a simple UI for entering in information and give you a more polished look than a wiki. They can generate both Web and flat-file documentation from the same source.

Online collaboration services, such as PBworks (pbworks.com) and MindTouch (mindtouch.com), are also being used for API documentation. In addition to the collaborative features of wikis, these offer additional features, such as hosting, fine-grained access control and scripting capabilities. These services typically require a subscription fee for commercial use.

Ship It!

Good API documentation is critical for getting your platform adopted and for cutting down on the number of support calls your company receives. If you can convince your manager to hire a technical writer with the right skills, then do it. But if you can't, follow the guidelines in this article.

Your documentation should have an overview, help on getting started, sample code and reference material. In the overview, be sure to explain why your platform should be used. Put together tutorials to help developers get started. The sample code should focus on clarity and simplicity, and it won't always follow the same coding principles as production code. Your reference material should be detailed and consistent. There are a number of tools available to get your documentation published on the Web.

Now get writing!

PETER GRUENBAUM started out as a physicist but became a software developer, working on technologies as diverse as Tablet PCs, Augmented Reality, computer-aided design and surgical simulation. He founded SDK Bridge LLC to bring together his love of technology and writing, where he writes and teaches about technology.

THANKS to the following technical experts for reviewing this article:
John Musser (Programmable Web) and Eugene Osovetsky (WebServius)

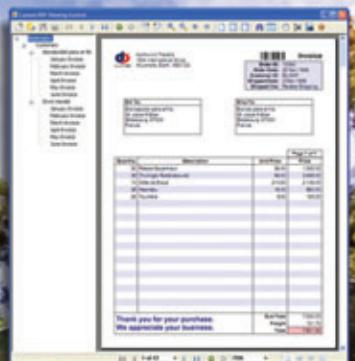


Dependable Developer Components

Try our three
new products
FREE today!

Fully functional and never
expiring evaluation
editions available at
www.cete.com/download

DynamicPDF Viewer
Our new, customizable DynamicPDF Viewer allows you to display PDF documents within any WinForm application. No longer rely on an external viewer for displaying your PDF documents. DynamicPDF Viewer utilizes the proven reliable and efficient Foxit PDF viewing engine and maximizes performance and compatibility with our other DynamicPDF products.



DynamicPDF Converter
Our DynamicPDF Converter library can efficiently convert over 30 document types (including HTML and all common Office file formats) to PDF. Events can be used to manage the action taken on a successful or failed conversion. It is highly intuitive and flexible and integrates well with our other DynamicPDF products.



DynamicPDF Rasterizer
Our DynamicPDF Rasterizer library can quickly convert PDF documents to over 10 common image formats including multi-page TIFF. Rasterizing form field values as well as annotations is fully supported. PDFs can also be rasterized to a System.Drawing.Bitmap class for further manipulation.



To learn more about these or any of our other popular tools:
DynamicPDF Generator, **DynamicPDF Merger**, **DynamicPDF ReportWriter**,
DynamicPDF Suite, **DynamicPDF WebCache** or **Firemail**, visit us online.



ceTe Software has been delivering quality software applications and components to our customers for over 10 years. Our DynamicPDF product line has proven our commitment to delivering innovative software components and our ability to respond to the changing needs of software developers. We back our products with a first class support team trained to provide timely, accurate and thorough responses to any support needs.

ceTe software
INFINITE POSSIBILITIES

www.cete.com
info@cete.com

800.631.5006
+1 410.772.8620



Web UI Test Automation with the WebBrowser Control

In this month's column I show you a new way to create UI test automation for Web applications. The technique I present provides one solution to a very common but tricky testing scenario: how to deal with modal message boxes generated by a Web application.

The best way for you to see where I'm headed is to take a look at the screenshots in **Figures 1** and **2**. The image in **Figure 1** shows a simplistic but representative Web application hosted in Internet Explorer. The application accepts user input into a text box, and after the user clicks on the button labeled Click Me, the app identifies the color of the input item, then displays the result in a second text box.

Notice that when the Click Me button is clicked, the application's logic checks to see if the user input box is empty. If so, it generates a modal message box with an error message. Dealing with a message box is problematic in part because the message box isn't part of the browser or the browser document.

Now take a look at the example test run in **Figure 2**. The test harness is a Windows Forms application. Embedded inside the Windows Forms app is a WebBrowser control that gives the Windows Forms the ability to display and manipulate the dummy Web application under test.

If you examine the messages in the ListBox control at the bottom of the Windows Forms harness, you'll see that the test harness begins by loading the Web app under test into the WebBrowser control. Next, the harness uses a separate thread of execution to watch for—and deal with—any message boxes generated by the Web app. The harness simulates a user-click on the Web app Click Me button, which in turn creates a modal error message box. The watcher thread finds the message box and simulates a user clicking it away. The test harness concludes by simulating a user typing "roses" into the first input box, clicking on the Click Me button, and looking for the test case expected response of "red" in the second text box.

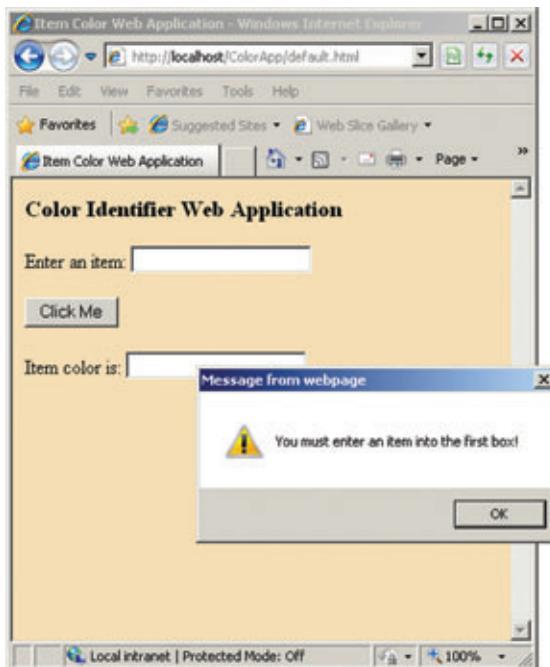


Figure 1 Example Web App under Test

In this article, I briefly describe the example Web application under test. I then walk you through the code for the Windows Forms test harness so you'll be able to modify my code to meet your testing scenarios. I conclude by describing situations where this technique is applicable and when alternative techniques may be better.

This article assumes you have basic Web development skills and intermediate C# coding skills, but even if you're new to C# you should be able to follow without much difficulty. I think you'll find the technique I present here a useful addition to your personal software testing, development and management tool kit.

The Application Under Test

Let's take a look at the code for the example Web app that's the target of my test automation. For simplicity I

created the application using Notepad. The application functionality is supplied by client-side JavaScript rather than by server-side processing. As I'll explain later, this test automation technique will work with applications based on most Web technologies (such as ASP.NET, Perl/CGI and so on), but the technique is best suited for applications that use JavaScript to generate message boxes. The entire Web application code is presented in **Figure 3**.

I saved my Web app as default.html in a directory named ColorApp located in the C:\Inetpub\wwwroot directory on my test host machine. To steer clear of security issues, the technique I present here works best when the test automation runs directly on the machine that acts as the Web server hosting the application under test. To keep my example Web app simple and not obscure details of the test automation, I took shortcuts you wouldn't see in a production Web app, such as eliminating error checks.

The heart of the Web app's functionality is contained in a JavaScript function named processclick. That function is called

Code download available at msdn.microsoft.com/mag201011TestRun.

when a user clicks on the application's button control with ID Button1 and label value "Click Me." The processclick function first checks to see if the value in input element TextBox1 is empty. If so, it generates an error message box using the JavaScript alert function. If the TextBox1 input element is not empty, the processclick function uses an if-then statement to produce a value for the TextBox2 element. Notice that because the Web application's functionality is provided by client-side JavaScript, the application doesn't perform multiple client-to-server round-trips, and therefore the Web application is loaded only once per functionality cycle.

The Windows Forms Test Harness

Now let's walk through the test harness code illustrated in **Figure 2** so that you'll be able to modify the code to meet your own needs. The test harness is a Windows Forms application; normally I'd use Visual Studio to create the program. However, I'm going to show you how to create the harness using Notepad and the command-line C# compiler because the ease of use and auto-generated code in Visual Studio hide some important concepts. Once you understand my example code you should have no trouble using Visual Studio rather than Notepad.

I open Notepad and begin my harness by declaring the namespaces used:

```
using System;
using System.Windows.Forms;
using System.Drawing;
using System.Runtime.InteropServices;
using System.Threading;
```

You need the System, Forms and Drawing namespaces for basic Windows Forms functionality. The InteropServices namespace allows the test harness to find and manipulate modal message boxes using the P/Invoke mechanism. P/Invoke allows you to create C# wrapper methods that call native Win32 API functions. The Threading namespace is used to spin off a separate thread that watches for the appearance of message boxes.

Next I declare a harness namespace and I begin the code for the main Windows Forms application class, which inherits from the System.Windows.Forms.Form class:

```
namespace TestHarness
{
    public class Form1 : Form
    {
        [DllImport("user32.dll",
            EntryPoint="FindWindow",
            CharSet=CharSet.Auto)]
        static extern IntPtr FindWindow(
            string lpClassName,
            string lpWindowName);
    ...
}
```

Immediately inside the Form1 definition I place a class-scope attribute that allows the test harness to call the external FindWindow API function, which is

located in user32.dll. The FindWindow API function is mapped to a C# method also named FindWindow, which accepts the internal name of a window control and returns an IntPtr handle to the control. The FindWindow method will be used by the test harness to get a handle to the message box generated by the Web application under test.

Next, I add two more attributes to enable additional Win32 API functionality:

```
[DllImport("user32.dll", EntryPoint="FindWindowEx",
    CharSet=CharSet.Auto)]
static extern IntPtr FindWindowEx(IntPtr hwndParent,
    IntPtr hwndChildAfter, string lpszClass,
    string lpszWindow);

[DllImport("user32.dll", EntryPoint="PostMessage",
    CharSet=CharSet.Auto)]
static extern bool PostMessage(IntPtr hWnd, uint Msg,
    int wParam, int lParam);
```

The C# FindWindowEx method associated with the FindWindowEx API function will be used to get a child control of the control found by FindWindow, namely the OK button on the message box. The C# PostMessage1 method associated with the PostMessage API function will be used to send a mouse-up and mouse-down message—in other words, a click—to the OK button.

Then I declare the three class-scope controls that are part of the Windows Forms harness:

```
private WebBrowser wb = null;
private Button button1 = null;
private ListBox listBox1 = null;
```

The WebBrowser control is a managed code wrapper around native code that houses the functionality of the Internet Explorer Web browser. The WebBrowser control exposes methods and properties that can be used to examine and manipulate a Web page housed in the control. The Button control will be used to launch the test automation, and the ListBox control will be used to display test harness logging messages.

Next I begin the code for the Form1 constructor:

```
public Form1()
{
    // button1
    button1 = new Button();
    button1.Location =
        new Point(20, 430);
    button1.Size = new Size(90, 23);
    button1.Text = "Load and Test";
    button1.Click +=
        new EventHandler(
            this.button1_Click);
}
```

This code should be fairly self-explanatory. It's quite possible that you've never had to write Windows Forms UI code like this from scratch before because Visual Studio does such a good job of generating UI boilerplate code. Notice the pattern: instantiate a control, set the properties and then hook up event

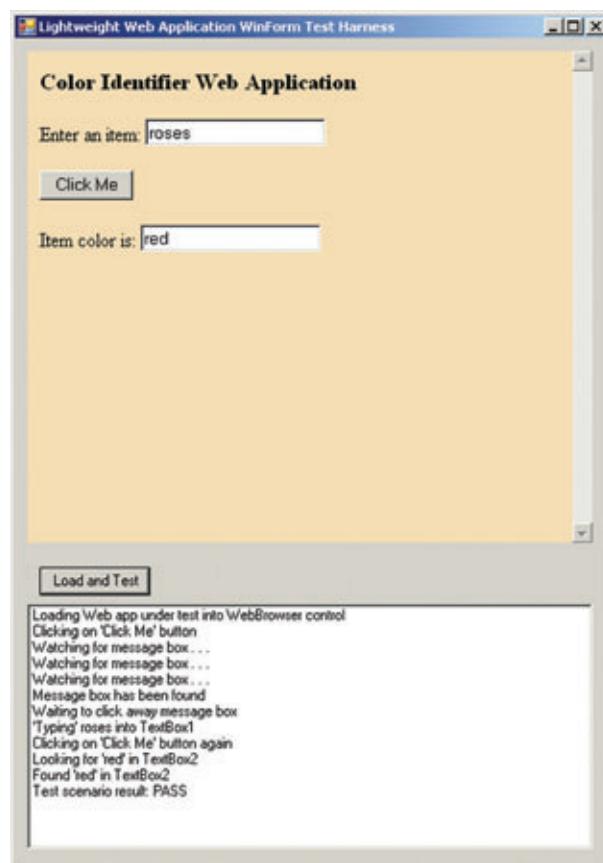


Figure 2 Example Test Run

handler methods. You'll see this same pattern with the WebBrowser control. When using the test automation technique I present in this article, it's useful to have a way to display logging messages, and a ListBox control works well:

```
// listBox1
listBox1 = new ListBox();
listBox1.Location = new Point(10, 460);
listBox1.Size = new Size(460, 200);

Next I set up the WebBrowser control:
// wb
wb = new System.Windows.Forms.WebBrowser();
wb.Location = new Point(10,10);
wb.Size = new Size(460, 400);
wb.DocumentCompleted +=
    new WebBrowserDocumentCompletedEventHandler(ExerciseApp);
```

The key thing to notice here is that I hook up an event handler method to the DocumentCompleted event so that, after the Web application under test is fully loaded into the WebBrowser control, control of execution will be transferred to a program-defined method named ExerciseApp (which I haven't yet coded). This is important because in almost all situations there will be a delay while the Web application is loading, and any attempt to access the Web application in the control before it's fully loaded will throw an exception.

You might have guessed that one way to deal with this is to place a Thread.Sleep statement into the test harness. But because the harness and WebBrowser control are both running in the same thread of execution, the Sleep statement will halt both the test harness and the WebBrowser loading.

I finish the Form1 constructor code by attaching the user controls to the Form object:

```
// Form1
this.Text = "Lightweight Web Application Windows Forms Test Harness";
this.Size = new Size(500, 710);
this.Controls.Add(wb);
this.Controls.Add(button1);
this.Controls.Add(listBox1);
} // Form1()
```

Next I code the event handler method for the button control on the Windows Forms harness that kicks off the test automation:

```
private void button1_Click(object sender, EventArgs e) {
    listBox1.Items.Add(
        "Loading Web app under test into WebBrowser control");
    wb.Url = new Uri(
        "http://localhost/ColorApp/default.html");
}
```

After logging a message, I instruct the WebBrowser control to load the Web app under test by setting the control's Url property. Notice that I've hardcoded the URL of the app under test. The technique I present here is best suited for lightweight, disposable test automation where hardcoded parameter values have fewer disadvantages than in situations where your test automation must be used over a long period of time.

Next I begin the code for the ExerciseApp method, which accepts control of execution when the DocumentCompleted event is fired:

```
private void ExerciseApp(object sender, EventArgs e) {
    Thread thread = new Thread(new
        ThreadStart(WatchForAndClickAwayMessageBox));
    thread.Start();
```

The ExerciseApp method contains most of the actual test harness logic. I begin by spawning a new thread associated with a program-defined method named WatchForAndClickAwayMessageBox. The idea here is that when a modal message box is generated by the Web application under test in the WebBrowser

Figure 3 The Web App

```
<html>
<head>
<title>Item Color Web Application</title>
<script language="JavaScript">
    function processclick() {
        if (document.all['TextBox1'].value == "") {
            alert("You must enter an item into the first box!");
        }
        else {
            var txt = document.all['TextBox1'].value;
            if (txt == "roses")
                document.all['TextBox2'].value = "red";
            else if (txt == "sky")
                document.all['TextBox2'].value = "blue";
            else
                document.all['TextBox2'].value = "I don't know that item";
        }
    }
</script>
</head>
<body bgcolor="#F5DEB3">
    <h3>Color Identifier Web Application</h3>
    <p>Enter an item:</p>
    <input type="text" id="TextBox1" /></p>
    <p><input type="button" value="Click Me"
        id="Button1"
        onclick="processclick()" /></p>
    <p>Item color is:<br/>
        <input type="text" id="TextBox2" /></p>
</body>
</html>
```

control, all test harness execution will halt until that message box is dealt with, meaning that the test harness can't directly deal with the message box. So by spinning off a separate thread that watches for a message box, the harness can indirectly deal with the message box.

Next I log a message and then simulate a user clicking on the Web application's Click Me button:

```
listBox1.Items.Add(
    "Clicking on 'Click Me' button");
HtmlElement btn1 =
    wb.Document.GetElementById("button1");
btn1.InvokeMember("click");
```

The GetElementById method accepts the ID of an HTML element that's part of the document loaded into the WebBrowser control. The InvokeMember method can be used to fire off events such as click and mouseover. Because there's no text in the Web app's TextBox1 control, the Web app will generate the error message box, which will be dealt with by the harness WatchForAndClickAwayMessageBox method, as I'll explain shortly.

Now, assuming that the message box has been dealt with, I continue the test scenario:

```
listBox1.Items.Add("Waiting to click away message box");
listBox1.Items.Add("Typing 'roses' into TextBox1");
HtmlElement tb1 = wb.Document.GetElementById("TextBox1");
tb1.InnerText = "roses";
```

I use the InnerText property to simulate a user typing "roses" into the TextBox1 control. Other useful properties for manipulating the Web application under test are OuterText, InnerHtml, and OuterHtml.

My automation continues by simulating a user click on the Web application's Click Me button:

```
listBox1.Items.Add(
    "Clicking on 'Click Me' button again");
btn1 = wb.Document.GetElementById("button1");
btn1.InvokeMember("click");
```

Unlike the previous simulated click, this time there's text in the TextBox1 control, so the Web application's logic will display some result text in the TextBox2 control, and the test harness can check for an expected result and log a pass or fail message:

```
listBox1.Items.Add("Looking for 'red' in TextBox2");
HtmlElement tb2 = wb.Document.GetElementById("TextBox2");
string response = tb2.OuterHtml;
if (response.IndexOf("red") >= 0) {
    listBox1.Items.Add("Found 'red' in TextBox2");
    listBox1.Items.Add("Test scenario result: PASS");
}
else {
    listBox1.Items.Add("Did NOT find 'red' in TextBox2");
    listBox1.Items.Add("Test scenario result: **FAIL**");
}
```

Notice that the HTML response will look something like <input type="text" value="red" />, so I use the IndexOf method to search the OuterHtml content for the correct expected result.

Here's the definition for the method that will deal with the Web app's modal message box:

```
private void WatchForAndClickAwayMessageBox() {
    IntPtr hMessageBox = IntPtr.Zero;
    bool mbFound = false;
    int attempts = 0;
    string caption = "Message from webpage";
    ...
}
```

I declare a handle to the message box, a Boolean variable to let me know when the message box has been found, a counter variable so I can limit the number of times my harness will look for the message box to prevent an endless loop, and the caption of the message box to look for. Although in this case the message box caption is fairly obvious, you can always use the Spy++ tool to verify the caption property of any window control.

Next I code a watching loop:

```
do {
    hMessageBox = FindWindow(null, caption);
    if (hMessageBox == IntPtr.Zero) {
        listBox1.Items.Add("Watching for message box . . . ");
        System.Threading.Thread.Sleep(100);
        ++attempts;
    }
    else {
        listBox1.Items.Add("Message box has been found");
        mbFound = true;
    }
} while (!mbFound && attempts < 250);
```

I use a do-while loop to repeatedly attempt to get a handle to a message box. If the return from FindWindow is IntPtr.Zero, I delay 0.1 seconds and increment my loop attempt's counter. If the return is not IntPtr.Zero, I know I've obtained a handle to the message box and I can exit the do-while loop. The "attempts < 250" condition will limit the amount of time my harness is waiting for a message box to appear. Depending on the nature of your Web app, you may want to modify the delay time and the maximum number of attempts.

After the do-while loop exits, I finish up the WatchForAndClickAwayMessageBox method by seeing if the exit occurred because a message box was found or because the harness timed out:

```
if (!mbFound) {
    listBox1.Items.Add("Did not find message box");
    listBox1.Items.Add("Test scenario result: **FAIL**");
}
else {
    IntPtr hOkBtn = FindWindowEx(hMessageBox, IntPtr.Zero, null, "OK");
    ClickOn(hOkBtn );
}
```

If the message box wasn't found, I classify this as a test case failure and log that result. If the message box was found, I use

the FindWindowEx method to get a handle to the OK button child control located on the parent message box control and then call a program-defined helper method named ClickOn, which I define as:

```
private void ClickOn(IntPtr hControl) {
    uint WM_LBUTTONDOWN = 0x0201;
    uint WM_LBUTTONUP = 0x0202;
    PostMessage1(hControl, WM_LBUTTONDOWN, 0, 0);
    PostMessage1(hControl, WM_LBUTTONUP, 0, 0);
}
```

The Windows message constants 0201h and 0202h represent left-mouse-button-down and left-mouse-button-up, respectively. I use the PostMessage1 method that's hooked to the Win32 PostMessage API function, which I described earlier.

My test harness ends by defining the harness Main method entry point:

```
[STAThread]
private static void Main() {
    Application.Run(new Form1());
}
```

After saving my test harness as Harness.cs, I used the command-line compiler. I launch the special Visual Studio command shell (which knows where the csc.exe C# compiler is), navigate to the directory holding my Harness.cs file, and issue the command:

```
C:\LocationOfHarness> csc.exe /t:winexe Harness.cs
```

The /t:winexe argument instructs the compiler to generate a Windows Forms executable rather than the default console application executable. The result is a file named Harness.exe, which can be executed from the command line. As I mentioned earlier, you will likely want to use Visual Studio rather than Notepad to create WebBrowser control-based test automation.

Wrapping Up

The example I've presented here should give you enough information to get you started writing your own WebBrower control-based test automation. This technique is best suited for lightweight automation scenarios—situations where you want to get your test automation up and running quickly and where the automation has a short expected lifespan. The strength of this technique is its ability to deal with modal message boxes—something that can be quite tricky when using other UI test-automation approaches. This technique is especially useful when you're testing Web application functionality that's generated primarily by client-side JavaScript.

In situations where Web application functionality is generated by multiple client-server round trips, you'll have to modify the code I've presented, because each time a response is returned from the Web server, the DocumentCompleted event will be fired. One approach for dealing with this is to create and use a variable that tracks the number of DocumentCompleted events and adds branching logic to your harness. ■

DR. JAMES McCAFFREY works for Volt Information Sciences Inc., where he manages technical training for software engineers working at the Microsoft Redmond, Wash., campus. He's worked on several Microsoft products, including Internet Explorer and MSN Search. Dr. McCaffrey is the author of ".NET Test Automation Recipes" (Apress, 2006) and can be reached at jammc@microsoft.com.

THANKS to the following Microsoft technical experts for reviewing this article: Paul Newson and Dan Liebling



Multiparadigmatic .NET, Part 3: Procedural Programming

Last month, the software-design exercise as one of commonality and variability stood as the centerpiece of the discussion (see msdn.microsoft.com/magazine/gg232770). It left us with the idea that software languages such as C# and Visual Basic offer different paradigms for representing these commonality/variability concepts along different dimensions, and that the heart of multiparadigmatic design is in pairing up the demands of the domain with the facilities of the language.

This month, we begin by examining one of the older facilities of programming languages, “procedural programming,” also sometimes known as “structured programming,” though the two are somewhat subtly different. Although commonly seen as “old school” and therefore outdated and useless in modern software design, the procedural design paradigm still shows up in a surprising number of places.

Proceeding It, Old School

For those of us who weren’t alive when structured programming emerged as a new term, its core tenet was to put some definition (structure) around the code being written—at a practical level, this meant “single entry points” and “single exit points” to the blocks of code being written in assembly at the time. The goal here was pretty simple, in retrospect: put some higher-level abstractions around the repetitive bits of code that were floating around.

But frequently, these commands (procedures) needed some variation to them if they were to be at all useful, and parameters—input passed to the procedure to vary its execution—were included as part of the approach, first informally (“pass the character you want to display in the AX register”), then formally (as parameters to functions, as in C/C++/C#/Java/Visual Basic and the like). Procedures often calculate some kind of returned value, sometimes derived from the input passed in, sometimes simply to indicate success or failure (such as in the case of writing data to a file or database); these are also specified and handled by the compiler.

However, all of this is a remedial topic for most readers. What the multiparadigm approach asks of us isn’t to rehash history, but to look at it again through the lens of commonality analysis. What, specifically, is being generalized in the procedural approach, and how do we introduce variability? Once the variability is identified, what kind of variability is it—positive or negative?

With commonality/variability glasses on, the procedural paradigm yields up interesting secrets: Commonality is gathered into procedures, which are essentially named blocks of code that can be

invoked from any given context. (Procedural languages rely heavily on “scope” to isolate work inside the procedure away from the surrounding context.) One way to introduce variability within the procedure is by way of parameters (indicating how to process the remainder of the parameters, for example), which can have either positive or negative variability, depending on how the procedure itself is written. If the source to that procedure is unavailable or shouldn’t be modified for some reason, variability can still be had by creating a new procedure that either calls to the old procedure, or not, depending on whether positive or negative variability is desired.

The procedural design paradigm
still shows up in a surprising
number of places.

Hello Procedures

In essence, the procedure provides common behavior, which can vary based on input. And, ironically enough, the first example we see of the procedural paradigm lies in the very first example most Microsoft .NET Framework programmers ever see:

```
Sub Main()
    Console.WriteLine("{0}, {1}!", "Hello", "world!")
End Sub
```

In the WriteLine implementation, developers pass a format string describing not only what to print out but how to print it, including formatter commands contained within the replacement markers, like so:

```
Sub Main()
    Console.WriteLine("Hello, world, it's {0:hh} o'clock!", Date.Now)
End Sub
```

The implementation of WriteLine is an interesting case study, in that it differs somewhat from its ancient predecessor, printf from the C standard library. Recall that printf took a similar kind of format string using different formatting markers and wrote directly to the console (the STDOUT stream). If a programmer wanted to write formatted output to a file, or to a string, different variations of printf had to be invoked: fprintf in the case of file output, or sprintf in the case of a string. But the actual formatting of the output was common, and often C runtime libraries took advantage of this fact by creating a single generic formatting function before sending the results to the final destination—

a perfect example of commonality. However, this formatting behavior was considered “closed” to the average C developer and couldn’t be extended. The .NET Framework takes one step beyond that, offering developers the chance to create new formatting markers by passing responsibility off to the objects passed in to `WriteLine` after the format string. If the object implements the `IFormattable` interface, it’s given the responsibility for figuring out the formatting marker and returning an appropriately formatted string for processing.

Variability could also hide behind other places in the procedural approach. When sorting values, the `qsort` (a Quicksort implementation) procedure needed help to know how to compare two elements to determine which one was greater or lesser than the other. To require developers to write their own wrappers around `qsort`—the traditional variability mechanism, when the original was untouchable—would’ve been too awkward and difficult. Fortunately, the procedural paradigm offered a different approach, an early variation of what would later become known as Inversion of Control: The C developer passed in a pointer to a function, which `qsort` invoked as part of its operation. This, in essence a variation of the parameters-as-variability approach, offered an open-ended variability approach, in that any procedure (so long as it met the parameter and return type expectations) could be used. Although somewhat rare at first, over time this paradigm’s idiom became more and more commonplace, usually under the general label of “callbacks”; by the time Windows 3.0 was released, it was an accepted core practice and necessary to write Windows programs.

Hello Services

Most interestingly, the place where the procedural paradigm has achieved the most widespread success (if we blithely ignore the unbelievable success and ubiquity of the C standard library, of course) is in the service-oriented realm. (Here, I use the term “service” to mean a wider collection of software, rather than the traditional narrow view of just WS-* or SOAP/Web Services Description Language [WSDL]-based services; REST-based implementations as well as Atom/RSS implementations fit much the same definition.)

According to past literature appearing on msdn.com, such as “Principles of Service-Oriented Design” (msdn.microsoft.com/library/bb972954), services obey four basic tenets:

- Boundaries are explicit.
- Services are autonomous.
- Services share schema and contract, not class.
- Service compatibility is based on policy.

These tenets, perhaps without intending to do so, reinforce the nature of services as belonging to the procedural paradigm of design more than to the object-oriented one. “Boundaries are explicit” reinforces the notion that the service is an entity separate and distinct from the system invoking it; this view is reinforced by the notion that “services are autonomous” and therefore distinct from one another, ideally even at an infrastructure-management level. “Services share schema and contract, not class” speaks to the notion that services are defined in terms of the parameters sent to them, expressed as XML (or JSON) constructs, not specific runtime types from a particular programming language or platform. Finally,

“Service compatibility is based on policy” suggests that services must be compatible based on policy declarations, which provide more of a context around the invocation—this is something that the procedural paradigm historically has assumed from the surrounding environment, and as such, it isn’t necessary to define explicitly.

Developers may be quick to point out that in classic WSDL-based services, it’s more difficult to create variability because the service is tied to the schema definition of the input type. But this is only for the most basic (or code-generative) of services—input and result types can be (and frequently are) reused across service definitions. In fact, if the notion of service is expanded to include REST-based systems, then the service can accept any number of different kinds of input types—essentially the parameters to the procedure are taking on an open-ended and interpretive role not generally seen in traditional statically typed procedures—and behave differently, bringing the variability within that service squarely to the fore once again. Some of that behavior will, of course, need to be validational in nature, because the service’s URL (its name) won’t always be appropriate for every kind of data that can be thrown at it.

Politics aside, the classic service—be it a RESTful one or a SOAP/WSDL-based one—bears a striking resemblance to the classic procedural paradigm.

When services are seen through the lens of a messaging system, such as BizTalk, ServiceBus or some other Enterprise Service Bus, the procedural aspect still holds, though now the entire variability rests with the messages being passed around, because the messages carry the entirety of the call context—not even the name of the procedure to invoke is present. This also implies that the variability mechanism by which we wrap another procedure in a new one—either introducing or restricting variability in doing so—is no longer present, because we typically don’t control how messages are passed around the bus.

Succeeding with Proceeding

The procedural paradigm demonstrates some of the earliest commonality/variability dimensions:

- **Name and behavior.** Names convey meanings. We can use commonality of name to group items (such as procedures/methods) that have the same meaning. In fact, “modern” languages have allowed us to capture this relationship more formally by allowing us to have different methods use the same name, so long as they vary in the number and/or types of parameters; this is method overloading. C++, C# and Visual Basic can also take advantage of appropriately named methods by creating methods whose names are well-understood based on algebra; this is operator



STATEMENT OF OWNERSHIP, MANAGEMENT AND CIRCULATION

(Required by 39 U.S.C. 3685, United States Postal Service)

1. Title of Publication: MSDN Magazine
2. Publication No. 1528-4859
3. Filing Date: 9/30/10
4. Frequency of Issue: Monthly
5. No. of issues published annually: 12
6. Annual Subscription Price: US \$35, International \$60
7. Mailing address of known office of publication: 9201 Oakdale Ave., Ste. 101, Chatsworth, CA 91311
8. Mailing address of the headquarters of general business offices of the publisher: Same as above.
9. Name and complete mailing address of Publisher, Editor, and Managing Editor: Henry Allain, President, 16261 Laguna Canyon Rd., Ste. 130, Irvine, CA 92618 Matt N. Morollo, Publisher, 600 Worcester Rd., Ste. 204, Framingham, MA 01702 Doug Barney, VP/Editorial Director, 600 Worcester Rd., Ste. 204, Framingham, MA 01702 Wendy Gonchar, Managing Editor, 16261 Laguna Canyon Rd., Ste. 130, Irvine, CA 92618
10. Owner(s): 1105 Media, Inc. dba: 101 Communications LLC, 9201 Oakdale Ave., Ste. 101, Chatsworth, CA 91311. Listing of shareholders in 1105 Media, Inc.
11. Known Bondholders, Mortgagees, and Other Security Holders Owning or Holding 1 Percent or more of the Total Amount of Bonds, Mortgages or Other Securities: Nautic Partners V, L.P., 50 Kennedy Plaza, 12th Flr., Providence, RI 02903 Kennedy Plaza Partners III, LLC, 50 Kennedy Plaza, 12th Flr., Providence, RI 02903 Alta Communications 1X, L.P., 1X-B, L.P., Assoc., LLC, 28 State St., Ste. 1801, Boston, MA 02109
12. The tax status has not changed during the preceding 12 months.
13. Publication Title: MSDN Magazine
14. Issue date for Circulation Data Below: September 2010
15. Extent & Nature of Circulation:

	Average No. Copies Each Month During Preceding 12 Months	No. Copies of Single Issue Published Nearest to Filing Date
a. Total Number of Copies (Net Press Run)	80,543	77,948
b. Legitimate Paid/and or Requested Distribution		
1. Outside County Paid/Requested Mail Subscriptions Stated on PS Form 3541	63,268	66,491
2. In-County Paid/Requested Mail Subscriptions Stated on PS Form 3541	1,163	1,073
3. Sales Through Dealers and Carriers, Street Vendors, Counter Sales, and Other Paid or Requested Distribution Outside USPS®	9,680	8,640
4. Requested Copies Distributed by Other Mail Classes Through the USPS	0	0
c. Total Paid and/or Requested Circulation	74,111	76,204
d. Nonrequested Distribution		
1. Outside County Nonrequested Copies Stated on PS Form 3541	4,030	519
2. In-County Nonrequested Copies Distribution Stated on PS Form 3541	0	0
3. Nonrequested Copies Distribution Through the USPS by Other Classes of Mail	0	0
4. Nonrequested Copies Distributed Outside the Mail	2,249	1,036
e. Total Nonrequested Distribution	6,279	1,555
f. Total Distribution	80,390	77,759
g. Copies not Distributed	153	189
h. Total	80,543	77,948
i. Percent paid and/or Requested Circulation	92.19%	98.00%

16. Publication of Statement of Ownership for a Requester Publication is required and will be printed in the November 2010 issue of this publication.

17. I certify that all information furnished on this form is true and complete:
Abraham Langer, Senior Vice President, Audience Development and Digital Media

overloading. F# takes this even further by allowing developers to create new operators.

• **Algorithm.** Algorithms aren't just mathematical calculations, but rather repeated steps of execution. If the entire system (rather than individual layers) is seen in a top-down form, interesting process/code fragments—use cases, in fact—begin to emerge that form families. After these steps (procedures) have been identified, families can form around the variability based on how the algorithm/procedure operates on different kinds of data/parameters. In C#, F# and Visual Basic, these algorithms can be varied by placing them in base classes, then varied by inheriting the base class and replacing the base's behavior; this is method overriding. Algorithmic behavior can also be customized by leaving part of that behavior unspecified and passed in; this is using delegates as Inversion of Control or callbacks.

One final note before we wrap up this piece. The procedural paradigm may not line up one-to-one with the service-oriented world; in fact, many service-oriented architecture evangelists and proponents will reject even the smallest association to the procedural paradigm, for fear that such an association will somehow take the shine off of their vested interest. Politics aside, the classic service—be it a RESTful one or a SOAP/WSDL-based one—bears a striking resemblance to the classic procedural paradigm. As a result, using the same commonality analysis during service design helps create an acceptable level of granularity, though designers must take care to ensure that the (assumed) traversal of the network to execute the service at the service host's location won't be blithely ignored. In particular, naïve implementations of services using the procedural paradigm might attempt to use the “pass a callback” approach to variability, and while this isn't entirely a terrible idea, it could represent a major bottleneck or performance problem.

To this day, the procedural paradigm still appears throughout a great deal of what we do, but it's been lurking under the surface, hiding from developers under an assumed name. Our next subject, object orientation, has no such excuse—it's the perky, outgoing, “Hey, come look at me!” younger sister to its moody, melodramatic and often-ignored procedural older sibling. In next month's piece, we'll start analyzing the commonality/variability dimensions of objects, and some of what we find may prove surprising.

In the meantime, as an intellectual exercise, cast your gaze around the various tools you use and identify which of them use fundamentally procedural tactics. (Hint: Two of them are tools you use every day while writing software: the compiler and MSBuild, the build system hidden away behind the Build button in Visual Studio.)

And, as always, happy coding! ■

TED NEWARD is a principal with Neward & Associates, an independent firm specializing in enterprise Microsoft .NET Framework and Java platform systems. He's written more than 100 articles, is a C# MVP and INETA speaker, and has authored and coauthored a dozen books, including "Professional F# 2.0" (Wrox, 2010). He also consults and mentors regularly. Reach him at ted@tedneward.com with questions or consulting requests, and read his blog at blogs.tedneward.com.

THANKS to the following technical expert for reviewing this article:
Anthony Green

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 +91-80-6785-1111
[@infragistics](http://twitter.com/infragistics)





Web Application Configuration Security Revisited

A few years ago—prior to my time at Microsoft and the Security Development Lifecycle (SDL) team—I wrote an article on the dangers of insecure web.config settings and named the top 10 worst offenders. You can still find this article today—just search for “Top 10 Application Security Vulnerabilities in Web.Config Files” in your favorite search engine. The configuration vulnerabilities I talked about back then are still relevant and serious today, although they probably wouldn’t come as huge surprises to regular readers of *MSDN Magazine*. It’s still important to enable custom errors; it’s still important to disable tracing and debugging before pushing your application to production; and it’s still important to require SSL for authentication cookies.

Even the most securely coded ASP.NET application can be hacked if it isn’t configured correctly.

In this month’s column, I’d like to pick up where that article left off and discuss some of the more obscure but equally serious security misconfigurations. I’m also going to take a look at a new free tool from the Microsoft Information Security Tools team called the Web Application Configuration Analyzer that can help find these problems. Remember, even the most securely coded ASP.NET application can be hacked if it isn’t configured correctly.

EnableEventValidation

One of the more common mistakes I see developers make is that they give users a list of choices and then assume the users will, in fact, choose one of those values. It seems logical enough: If you add a ListBox control to a page and then pre-populate it with the list of all states in the United States, you’d expect to get back “Washington” or “Georgia” or “Texas”; you wouldn’t expect “Foo” or “!@#\$%” or “<script>alert(document.cookie);</script>”. There may not be a way to specify values like this by using the application in the traditional way, with a browser, but there are plenty of ways to access Web applications without using a browser at all! With a Web proxy tool such as Eric Lawrence’s Fiddler (which remains one of my favorite

tools for finding security vulnerabilities in Web applications and can be downloaded from fiddler2.com), you can send any value you want for any form field. If your application isn’t prepared for this possibility, it can fail in potentially dangerous ways.

The EnableEventValidation configuration setting is a defense-in-depth mechanism to help defend against attacks of this nature. If a malicious user tries to send an unexpected value for a control that accepts a finite list of values (such as a ListBox—but not such as a TextBox, which can already accept any value), the application will detect the tampering and throw an exception.

Bad:

```
<configuration>
  <system.web>
    <pages enableEventValidation="false"/>
```

Good:

```
<configuration>
  <system.web>
    <pages enableEventValidation="true"/>
```

PasswordFormat

The membership provider framework supplied as part of ASP.NET (starting in ASP.NET 2.0) is a great feature that keeps developers from having to reinvent the membership-functionality wheel time and time again. In general, the built-in providers are quite good from a security perspective when left in their default settings. However, if the membership configuration settings are changed, they can become significantly less secure.

One good example of this is the PasswordFormat setting, which determines how user passwords are stored. You have three choices: Clear, which stores passwords in plaintext; Encrypted, which encrypts the passwords before storing them; and Hashed, which stores hashes of the passwords instead of the passwords themselves. Of these choices, Clear is clearly the worst. It’s never appropriate to store passwords in plaintext. A much better choice is Encrypted, and the best choice is Hashed, because the best way to store a secret is not to store it at all. However, because there’s no way to retrieve the original password from a hash, if a user forgets his or her password, you won’t be able to recover it for him.

Bad:

```
<configuration>
  <system.web>
    <membership>
      <providers>
        <clear/>
        <add name="AspNetSqlMembershipProvider"
          passwordFormat="Clear"
          ...
        />
```

Better:

```
<configuration>
  <system.web>
    <membership>
      <providers>
        <clear/>
        <add name="AspNetSqlMembershipProvider"
          passwordFormat="Encrypted"
          ...
        />
```

Best:

```
<configuration>
  <system.web>
    <membership>
      <providers>
        <clear/>
        <add name="AspNetSqlMembershipProvider"
          passwordFormat="Hashed"
          ...
        />
```

MinRequiredPasswordLength and MinRequiredNonalphanumericCharacters

There are two values of the membership settings that should be changed from their defaults: the MinRequiredPasswordLength and MinRequiredNonalphanumericCharacters properties. For AspNetSqlMembershipProvider objects, these settings default to a minimum required password length of six characters, with no non-alphanumeric characters required. For better security, these settings should be set much higher. You should require at least a 10-character-long password, with two or more non-alphanumeric characters. A 14-character minimum with four or more non-alphanumeric characters would be better still.

It's true that password length and complexity are dual-edged swords: When you require your users to set longer and more complex passwords, there's less of a chance those passwords will fall to brute-force attacks, but there's also a correspondingly greater chance that your users won't be able to remember their passwords and will be forced to write them down. However, while this sounds like a horrible potential security hole, many security experts believe the benefits outweigh the risks. Noted security guru Bruce Schneier, for one, suggests that users create long, complex passwords and store them in their purse or wallet, as this is a place where people are used to securing small pieces of paper.

Bad:

```
<configuration>
  <system.web>
    <membership>
      <providers>
        <clear/>
        <add name="AspNetSqlMembershipProvider"
          minRequiredPasswordLength="6"
          minRequiredNonalphanumericCharacters="0"
          ...
        />
```

Good:

```
<configuration>
  <system.web>
    <membership>
      <providers>
        <clear/>
        <add name="AspNetSqlMembershipProvider"
          minRequiredPasswordLength="14"
          minRequiredNonalphanumericCharacters="4"
          ...
        />
```

The Microsoft Online Safety site (microsoft.com/protect/fraud/passwords/create.aspx) also suggests that users should write their passwords

down, and it has additional information on creating and securing strong passwords.

ValidateRequest

Cross-site scripting (XSS) continues to be the most common Web vulnerability. A report published by Cenzic Inc. in July found that in the first half of the year, XSS vulnerabilities accounted for 28 percent of all Web attacks. Given the potentially severe consequences of an XSS vulnerability—I've often called XSS "the buffer overflow of the Web" in the past—it's only logical that developers should do whatever they can to help defend their applications against this attack. It's especially nice when you get a defense that basically costs you nothing, and that's what ValidateRequest is.

Bad:

```
<configuration>
  <system.web>
    <pages validateRequest="false" />
```

Good:

```
<configuration>
  <system.web>
    <pages validateRequest="true" />
```

Cross-site scripting (XSS)
continues to be the most
common Web vulnerability.

ValidateRequest works by testing user input for the presence of common attack patterns, such as whether the input string contains angle brackets (<). If it does, the application throws an exception and stops processing the request. While this isn't a complete solution in and of itself—you should also always apply output encoding and input validation/sanitization logic, such as is built into the Microsoft Web Protection Library—ValidateRequest does block many types of popular XSS attacks. It's best to leave ValidateRequest enabled whenever possible.

MaxRequestLength

It's rarely a good idea to allow users to make arbitrarily large HTTP requests to your application. Doing so opens you to denial-of-service (DoS) attacks, where a single attacker could use up all your bandwidth, processor cycles or disk space and make your application unavailable to any of the other legitimate users you're trying to reach.

To help prevent this, you can set the MaxRequestLength property setting to an appropriately small value. The default value is 4096KB (4MB). Because different applications have different requirements as to what their usual and exceptional request sizes are, it's difficult to make a good rule of thumb about what the MaxRequestLength value should be set to. So, instead of giving examples of what "bad" and "good" settings would be, I just suggest that you keep in mind the fact that the higher you set this value, the more you put yourself at risk for a DoS attack:

```
<configuration>
  <system.web>
    <httpRuntime maxRequestLength="4096"/>
```

EnableViewStateMac

I've written previously about the EnableViewStateMac setting in the July 2010 Security Briefs column on view state security (microsoft.com/magazine/ff797918). For those who missed it, EnableViewStateMac is a defense to prevent attackers from tampering with client-side view state. When EnableViewStateMac is enabled, the ASP.NET application adds a cryptographic Message Authentication Code (MAC) to the hidden __VIEWSTATE form value. There's no way for an attacker to determine a valid MAC for an arbitrary attack—to try to poison a victim's view state to inject some malicious JavaScript, for example—so if an attacker tries to tamper with view state in this manner, the MAC will be invalid and the ASP.NET application will block the request.

Bad:

```
<configuration>
  <system.web>
    <pages enableViewStateMac="false"/>
```

Good:

```
<configuration>
  <system.web>
    <pages enableViewStateMac="true"/>
```

If you're deploying your application in a server farm environment, it's also important to remember to manually specify a key for the MAC rather than letting the application auto-generate random keys. (If you don't manually specify keys, each machine in the farm will auto-generate a different key, and the view state MAC created by any of the machines will be considered invalid and will be blocked by any of the other machines.)

It's rarely a good idea to allow users to make arbitrarily large HTTP requests to your application.

There are a few additional guidelines you should follow when manually creating keys to ensure maximum security for your view state. First, be sure to specify one of the SDL-approved cryptographic algorithms. For applications using the Microsoft .NET Framework 3.5 or earlier, this means using either SHA1 (which is the default algorithm) or AES. For applications using the .NET Framework 4, you can also use HMACSHA256, HMACSHA384 or HMACSHA512. Avoid weak algorithms such as MD5.

Bad:

```
<configuration>
  <system.web>
    <machineKey validation="MD5" validationKey="..."/>
```

Good:

```
<configuration>
  <system.web>
    <machineKey validation="AES" validationKey="..."/>
```

It's just as important to choose a strong key as it is to choose a strong algorithm. Use a cryptographically strong random-number generator to generate a 64-byte key (128-byte if you're using HMACSHA384 or HMACSHA512 as your key algorithm).

Reference sample code to generate appropriate keys is provided in the July 2010 Security Briefs column I mentioned earlier.

Bad:

```
<configuration>
  <system.web>
    <machineKey validation="AES" validationKey="12345"/>
```

Good:

```
<configuration>
  <system.web>
    <machineKey validation="AES" validationKey="143a907bb73069a2fe7c..."/>
```

ViewStateEncryptionMode

Just as you should apply a MAC to your application's view state to keep potential attackers from tampering with it, you should also encrypt the view state to keep them from reading it. Unless you're 100 percent sure there's no sensitive information in any of your view state, it's safest to set the ViewStateEncryptionMode property to encrypt and protect it.

Bad:

```
<configuration>
  <system.web>
    <pages ViewStateEncryptionMode="Never"/>
```

Good:

```
<configuration>
  <system.web>
    <pages ViewStateEncryptionMode="Auto"/>
```

Again, just as with EnableViewStateMac, you have your choice of several cryptographic algorithms the application will use to encrypt the view state. However, it's best to stick with AES, which is the only available algorithm currently approved by the SDL Cryptographic Standards.

Bad:

```
<configuration>
  <system.web>
    <machineKey decryption="DES" decryptionKey="" />
```

Good:

```
<configuration>
  <system.web>
    <machineKey decryption="AES" decryptionKey="" />
```

Finally, remember that if you're deploying your application in a server farm, you'll need to manually specify a key. Make sure to set the key value to a 24-byte cryptographically random value.

Bad:

```
<configuration>
  <system.web>
    <machineKey decryption="AES" decryptionKey="12345"/>
```

Good:

```
<configuration>
  <system.web>
    <machineKey decryption="AES" decryptionKey="143a907bb73069a2fe7c..."/>
```

UseUnsafeHeaderParsing

When developers are frustrated enough by a difficult bug, they'll often implement any change they read about that fixes the problem without really understanding what they're doing to their application. The UseUnsafeHeaderParsing setting is a great example of this phenomenon. While the word "unsafe" in the property name alone should be enough to throw up a red flag for most people, a quick Internet search reveals literally thousands of results suggesting developers enable this property. If you do enable UseUnsafeHeaderParsing, your application will ignore many of the HTTP RFC specifications and attempt to parse malformed requests. While

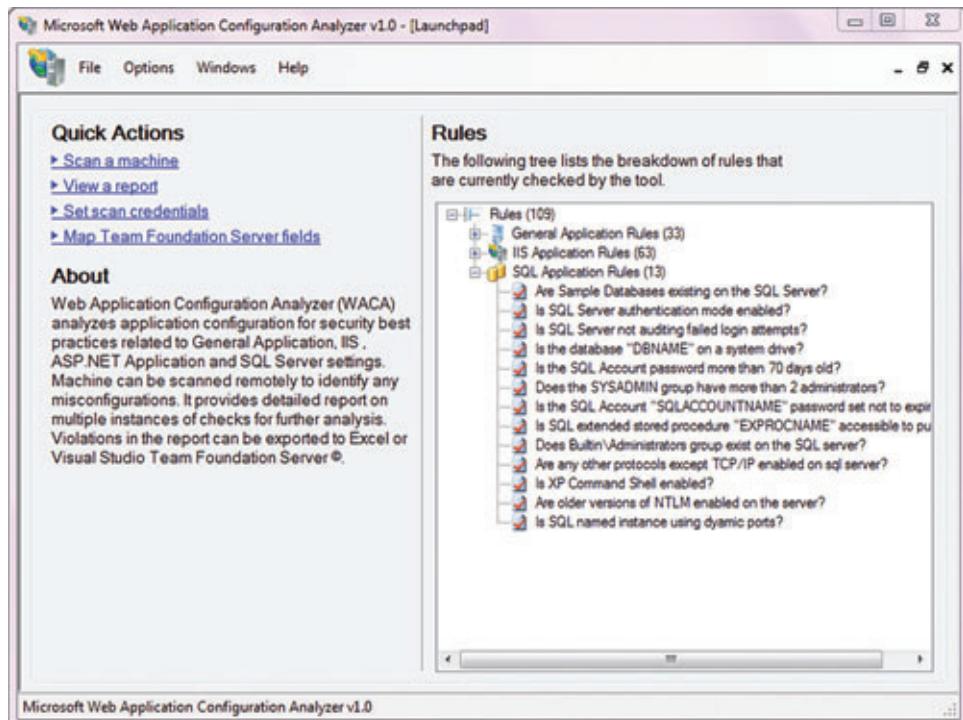


Figure 1 Web Application Configuration Analyzer Rules

doing so can allow your application to work with HTTP clients that disobey HTTP standards (which is why so many people suggest it as a problem fix), it can also open your application to malformed header attacks. Play it safe and leave this setting disabled.

Bad:

```
<configuration>
  <system.net>
    <settings>
      <httpWebRequest
        useUnsafeHeaderParsing=
        "true"/>
```

Good:

```
<configuration>
  <system.net>
    <settings>
      <httpWebRequest
        useUnsafeHeaderParsing=
        "false"/>
```

Web Application Configuration Analyzer (WACA)

Now that we've taken a look at some dangerous configuration settings, let's take a look at a tool that can help automate finding these settings in your code. After all, while manual code review can be useful, automated analysis can be more thorough and more consistent. You'll also save yourself from the drudgery of hand-reviewing XML files and leave yourself more time to solve more-interesting problems!

The Microsoft Information Security Tools team has released some excellent security tools, including two—AntiXSS/Web Protection Library and CAT.NET—that we've made mandatory for all internal .NET Framework Microsoft products and services as part of the Microsoft SDL. Its latest release, WACA, is designed to detect potentially dangerous misconfigurations, such as the ones I talked about in this article and in my earlier article on the top 10 most common web.config vulnerabilities. Some examples of WACA checks include:

- Is tracing enabled?
- Is MaxRequestLength too large?
- Are HttpOnly cookies disabled?
- Is SSL required for forms authentication login?
- Is EnableViewStateMac attribute set to false?

In addition, WACA can also check for misconfigurations in IIS itself,

as well as SQL database misconfigurations and even system-level issues. Some examples include:

- Is the Windows Firewall service disabled?
- Is the local admin named “Administrator”?
- Is the IIS log file on the system drive?
- Is execute enabled on the application virtual directory?

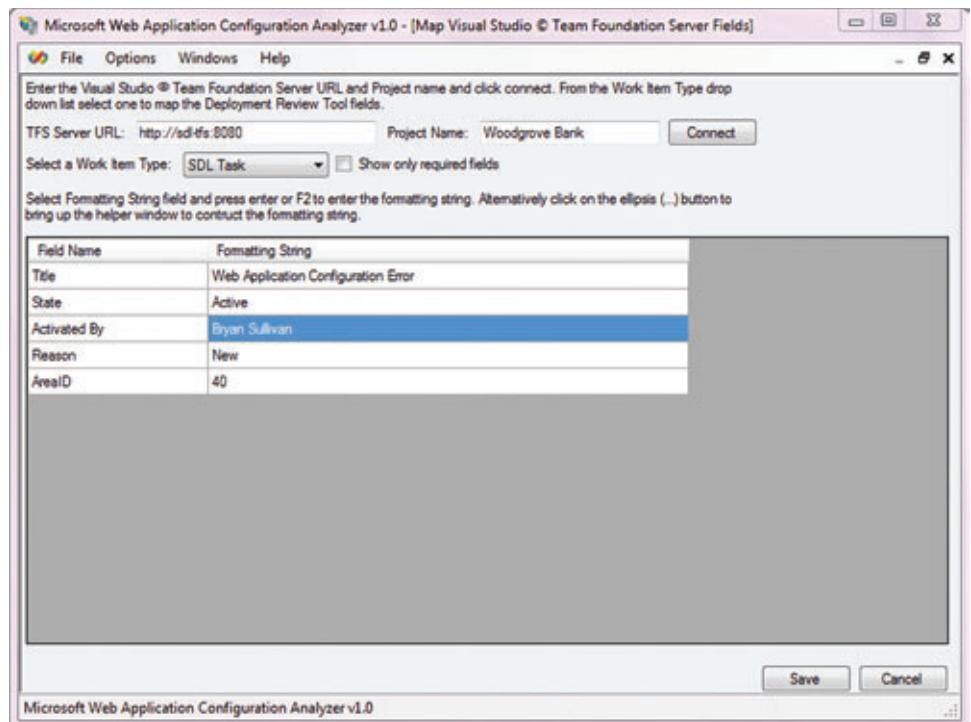


Figure 2 WACA Team Foundation Server Integration

- Are sample databases present on the SQL server?
- Is xp_cmdshell enabled on the SQL server?

While developers and testers will probably use WACA mostly for checking their applications' configuration settings, systems administrators and database administrators will find value in using WACA to check IIS, SQL and system settings (see **Figure 1**). In all, there are more than 140 checks in WACA derived from SDL requirements and patterns & practices coding guidelines.

One more really handy feature of WACA is that you can automatically create work items or bugs in Team Foundation Server (TFS) team projects from WACA scan results. This is especially useful when you use it with a team project created from either the SDL process template or the MSF-Agile+SDL process template. From the WACA TFS setup page, map the template field "Origin" to the value "Web Application Configuration Analyzer." Now when you view your bug reports and trend charts, you'll be able to filter and drill down into the WACA results to see how effective it's been at detecting potential vulnerabilities (see **Figure 2**).

You can read more about WACA on the Microsoft IT InfoSec group's page (msdn.microsoft.com/security/dd547422); watch a video demonstration of the tool presented by Anil Revuru, program manager for the WACA project (msdn.microsoft.com/security/ee909463); or, best of all, download the tool and try it for yourself (tinyurl.com/3x7bgfd).

Always Check Your Settings

It's frustrating to think that you could develop your application following every secure development guideline and best practice and still end up hacked because of a simple mistake in a web.config configuration file. It's even more frustrating when you realize that web.config files are designed to be changed at any time and that the configuration mistake could come years after you've finished coding the application and moved it to production. It's important to always check your configuration settings—not just by manual inspection, but with automated tools, and not just during the development lifecycle, but also in production.

Follow-up on Regular Expression DoS Attacks

On a completely different topic: In the May 2010 Security Briefs column (msdn.microsoft.com/magazine/ff646973), I wrote about the

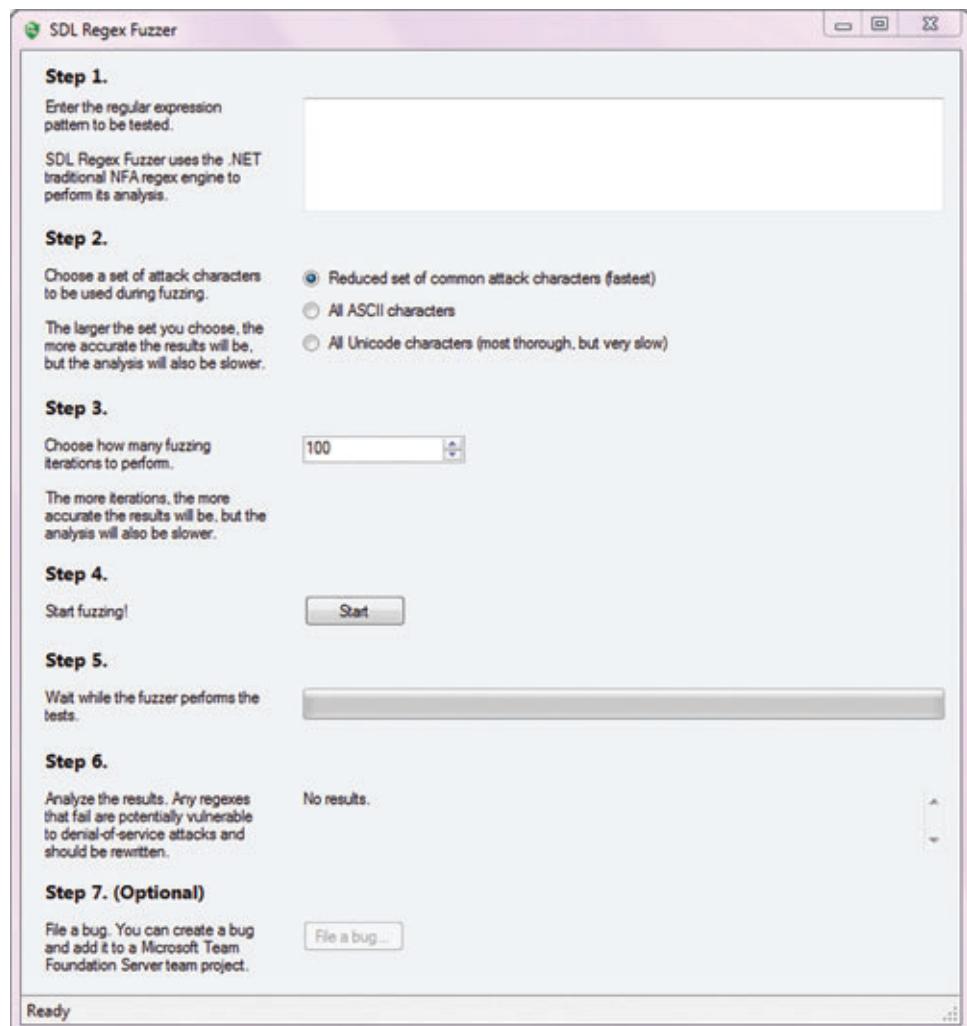


Figure 3 SDL Regex Fuzzer

regular expression DoS attack demonstrated by Checkmarx at the OWASP Israel conference in September 2009. In that column, I also provided code for a regex DoS fuzzer based on the Visual Studio Database Projects Data Generation Plan functionality. Although this approach was technically sound and worked well to detect regex vulnerabilities, it was admittedly somewhat tedious to generate the test data, and it did require you to own a license of Visual Studio Database Projects. So I'm happy to report that the SDL team has released a new, freely downloadable tool to fuzz for regex vulnerabilities that takes care of the data generation details for you. The tool has no external dependencies (other than .NET Framework 3.5). It's shown in **Figure 3**.

You can download SDL Regex Fuzzer from microsoft.com/sdl. Give it a try and let us know what you think. ■

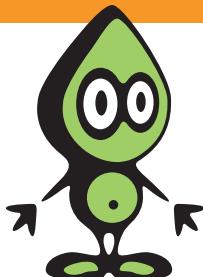
BRYAN SULLIVAN is a security program manager for the Microsoft Security Development Lifecycle team, where he specializes in Web application and Microsoft .NET Framework security issues. He's the author of "Ajax Security" (Addison-Wesley, 2007).

THANKS to the following technical expert for reviewing this article:
Anil Revuru

Know it all.

We've got all the answers.

- Thousands of code samples
- Over 29,000 articles
- Practical answers in active forums
- MFC/Microsoft® Visual C++®, Visual C#®, Visual Basic® .NET, ASP.NET, SQL Server®, Windows® Phone, and more!
- QuickAnswers
- Over 7.3 million members
- Start your FREE membership TODAY!



THE CODE PROJECT™
WWW.CODEPROJECT.COM



The Intricacies of Touch Controls

I was the kind of kid who took apart toasters to find out how they worked. Much later in life, I graduated to disassembling operating systems and application programs.

I don't do a lot of disassembling these days. But sometimes when I see a particularly interesting UI, I try to figure out how to code it on my own. It's an engineering exercise, of course, but I also like to think of myself as an art student who goes to the museum to paint copies of existing masterworks. In my own code, I strive for simplicity, of course, and to make use of pre-existing elements and controls. But mostly I like to give myself a challenge and hope I learn something new from it.

Recently I've been exploring Windows Phone 7, and I became intrigued by the pages used to set date and time, as shown in **Figure 1**. These controls (which are not publicly available) struck me as interesting touch interfaces. [*The controls were released in a Silverlight for Windows Phone Toolkit following completion of this article. It's available at silverlight.codeplex.com/releases/view/52297.—Ed.*]

I started wondering: How would I code these controls? In the past several issues, I've been exploring multi-touch in the Windows Presentation Foundation (WPF), so I decided to target that platform for my first shot at duplicating them. If successful, I could think about moving the code to Silverlight.

Exploring the Controls

When you first navigate to one of the date or time pages on Windows Phone 7, you see the current setting in gray squares centered in the middle of the page. Touch one of these squares and a list of other choices pops up on the top and bottom. Often this list is circular; as you can see in **Figure 1**, the month of December is followed by January. Circular lists are also used for the days of the month, hours and minutes. Non-circular lists are used for the year—the list ranges from 1601 to 3000—or (as you can see) to select AM or PM.

How different from a ListBox! A conventional ListBox displays the currently selected item with a highlight, but as you scroll through the ListBox, that selected item is sometimes scrolled completely out of view. In contrast, these date and time controls always tend to display the selected item in the center, and I began to think of



Figure 1 Windows Phone 7 Date and Time Pickers

these center areas as a type of "window," with the list functioning similar to the mechanical reels of old-fashioned slot machines, which in my mind I began referring to as "bands."

These controls seem to respond to touch in three distinct ways:

- If you simply tap another visible item in the band (such as the month of December in **Figure 1**), the item becomes highlighted when your finger leaves the screen, and that item shifts into the center window area.
- Instead of tapping, you can move one of the bands of items up or down with your finger. During that time, nothing in that band is highlighted. As soon as you lift your finger, the item closest to the window becomes highlighted and it shifts into the center.
- The third type of interface involves inertia. If the band of items is moving when your finger leaves the screen, it will continue moving while slowing down. As it gets close to stopping, the item closest to the window becomes highlighted and moves to the center. If necessary, sometimes the band reverses direction right at the end before stopping. That little effect—very natural, I had to admit—was something I knew would be one of the more "interesting" challenges in duplicating these controls.

Overall Architecture

Another "interesting" challenge involved the circular list of items, where December is followed by January and 12:00 is followed by 1:00. These circular bands are crucial aspects of the design—particularly in combination with inertia. The bands can be flicked in either direction, and inertia carries them forward to any item in the band without reaching a dead end.

After rolling around ideas in my mind for several days, I simply couldn't think of a better solution to the circular list than a custom panel, and the name WrappableStackPanel suggested itself. Such a panel would sometimes position its children from top to bottom,

Code download available at code.msdn.microsoft.com/mag20101UIFrontiers.

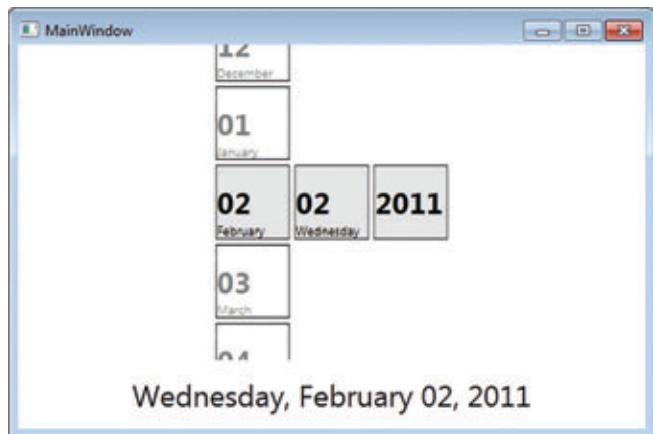


Figure 2 The TouchDatePicker Control in Action

and sometimes start with a child other than the first, and position the early children after the latter children. Of course, rendering more than one version of a particular child is prohibited in WPF, so the stack would always have a definite end.

The WrappableStackPanel would need a property (called StartIndex, for example) to denote the index of the child that should be displayed at the top of the panel, with the rest of the children following sequentially and looping back to child zero. In the general case, no child would be aligned precisely at the top of the panel. The topmost child would usually be partially above the top of the panel. This implied that the StartIndex property would be a floating point value rather than an integer.

But that meant the overall control would probably function in two distinct ways. One mode requires a WrappableStackPanel to host the items, where the WrappableStackPanel is fixed in position relative to the control, and the panel handles the positioning of children relative to itself. The other mode is for cases when a regular StackPanel is adequate (such as for the Year or the AM/PM band); in this mode, the children are fixed relative to the StackPanel, and the StackPanel is then scrolled relative to the control.

Would this control derive from ListBox? I decided it would not. ListBox incorporates its own selection logic, and the control I wanted has a rather different type of selection logic. If I derived from ListBox, I'd probably find it to be hindering rather than assisting me as I wrestled with the existing selection logic.

But I knew I wanted the control to maintain its own collection of items and—particularly—to let me define a template in XAML for displaying these items. This need suggested that an ItemsControl would be involved. ItemsControl is the parent class to Selector, from which ListBox and ComboBox derive, and is the obvious choice for displaying collections where no selection logic is necessary, or where the programmer will be handling customized selection logic. This was me.

The default control template for a ListBox includes a ScrollViewer; the ItemsControl doesn't. If you need to scroll the items displayed by an ItemsControl, you put the whole ItemsControl itself in a ScrollViewer.

I knew I couldn't use the existing ScrollViewer for this job. Although the existing ScrollViewer handles inertia, it doesn't

have any logic to orient a particular item in the center. My first stab at actual coding was to create an alternative to ScrollViewer called WindowedScrollView, which hosted an ItemsControl containing either the months, the days of the months or the years. In its MeasureOverride method, this WindowedScrollView could easily obtain the full size of the ItemsControl and the number of items being displayed—and therefore the uniform height of the individual items—so it could use this information to move the ItemsControl relative to itself based on the Manipulation events.

This approach worked fine when the ItemsControl didn't need to display wrappable bands of items. For those cases, I needed to set the ItemsPanel property of the ItemsControl to an ItemsPanelTemplate referencing the WrappableStackPanel. The big problem involved the WindowedScrollView communicating with the WrappableStackPanel through the ItemsControl sitting in the middle of the visual tree.

This seemed difficult. Moreover, I also slowly came to realize that my WindowedScrollView was unlike the regular ScrollViewer in that it had no visuals of its own. It was doing nothing more than sliding the ItemsControl relative to itself.

I decided to abandon that approach and instead derive from ItemsControl and do everything in there. I called it WindowedItemsControl class, and it implements selection logic, manipulation handling, scrolling and communicating with an optional WrappableStackPanel.

The downloadable source code for this article is a solution named TouchDatePickerDemo with a project of that name and a library project called Petzold.Controls. The library project includes WindowedItemsControl (divided into three files), WrappableStackPanel and a UserControl derivative named TouchDatePicker. The TouchDatePicker combines three WindowedItemsControls (for month, day and year) with a DatePresenter class and a property named DateTime.

After rolling around ideas
in my mind for several days, I
simply couldn't think of a better
solution to the circular list than
a custom panel.

Figure 2 shows the TouchDatePicker in action. A TextBlock is bound to the DateTime property to display the currently selected date.

The TouchDatePicker is basically a Grid with three columns for the month, day and year. **Figure 3** shows the XAML for the first WindowedItemsControl to handle the month. The DataContext is an object of type DatePresenter, which has properties AllMonths and SelectedMonth referenced by the WindowedItemsControl tag itself. The AllMonths property is a collection of MonthInfo objects, and SelectedMonth is also of type MonthInfo. The MonthInfo class has properties named MonthNumber and MonthName that you'll see referenced in the DataTemplate. The WrappableStackPanel is referenced down at the bottom.

Selection Logic

The Items collection maintained by ItemsControl is defined to accept items of type object. Each of these items is rendered based on a DataTemplate set to the ItemTemplate property of ItemsControl.

The ListBox does a little more. The ListBox wraps each of its items in a ListBoxItem control for the purpose of implementing selection logic. It's the ListBoxItem that has an IsSelected property along with the Selected and Unselected events, and which is responsible for visually indicating that an item is selected.

My goal with WindowedItemsControl was to implement selection logic without any kind of wrappers, and implement it in such

Figure 3 One-Third of the TouchDatePicker Control

```
<local:WindowedItemsControl x:Name="monthControl"
    Grid.Column="0"
    ItemsSource="{Binding AllMonths}"
    SelectedItem="{Binding SelectedMonth, Mode=TwoWay}"
    IsActiveChanged="OnWindowedItemsControlIsActiveChanged">

    <local:WindowedItemsControl.ItemTemplate>
        <DataTemplate DataType="local:MonthInfo">
            <Border Width="60" Height="60"
                BorderThickness="1"
                BorderBrush="{Binding ElementName=monthControl,
                    Path=Foreground}"
                Margin="2">
                <Grid>
                    <Rectangle Fill="{DynamicResource
                        {x:Static SystemColors.ControlLightBrushKey}}"
                        <Rectangle.Visibility>
                            <MultiBinding Converter="{StaticResource multiConverter}">
                                <Binding />
                                <Binding ElementName="monthControl" Path="SelectedItem" />
                            </MultiBinding>
                        </Rectangle.Visibility>
                </Rectangle>
            </Border>
            <TextBlock Text="{Binding MonthNumber, StringFormat=D2}"
                VerticalAlignment="Center"
                FontSize="24"
                FontWeight="Bold" />

            <TextBlock Text="{Binding MonthName}"
                VerticalAlignment="Bottom"
                FontSize="10" />

            <Rectangle Fill="#80FFFFFF">
                <Rectangle.Visibility>
                    <MultiBinding Converter="{StaticResource multiConverter}"
                        ConverterParameter="True">
                        <Binding />
                        <Binding ElementName="monthControl" Path="SelectedItem" />
                    </MultiBinding>
                </Rectangle.Visibility>
            </Rectangle>
        </Grid>

        <Border.Visibility>
            <MultiBinding Converter="{StaticResource multiConverter}">
                <Binding />
                <Binding ElementName="monthControl" Path="SelectedItem" />
                <Binding ElementName="monthControl" Path="IsActive" />
            </MultiBinding>
        </Border.Visibility>
    </Border>
</DataTemplate>
</local:WindowedItemsControl.ItemTemplate>

<local:WindowedItemsControl.ItemsPanel>
    <ItemsPanelTemplate>
        <local:WrappableStackPanel IsItemsHost="True" />
    </ItemsPanelTemplate>
</local:WindowedItemsControl.ItemsPanel>
</local:WindowedItemsControl>
```

a way so I could define the visuals of the selected item entirely in XAML. To help, the WindowedItemsControl has a SelectedIndex property (which it uses internally to determine where items should be positioned) and a SelectedItem property, which is the particular object in its Items collection that corresponds to the SelectedIndex.

In the XAML in **Figure 3**, this SelectedItem property is referenced in the DataTemplate property like so:

```
<Binding ElementName="monthControl" Path="SelectedItem" />
```

Within that same DataTemplate, the item itself can be referenced with a much simpler Binding expression:

```
<Binding />
```

If the objects referenced by these two bindings are equal, then the DataTemplate should have some special markup to indicate a selected item—in this case, gray background shading and non-dimmed text.

Normally in XAML you can't determine whether two bindings reference the same object, but I wrote a multi-binding converter specifically for that purpose. It's called EqualsToVisibilityMultiConverter and returns Visibility.Visible or Visibility.Hidden based on whether two objects are equal. Here's how it's used in the DataTemplate for the gray background:

```
<Rectangle Fill="{DynamicResource {x:Static SystemColors.
    ControlLightBrushKey}}"
    <Rectangle.Visibility>
        <MultiBinding Converter="{StaticResource multiConverter}">
            <Binding />
            <Binding ElementName="monthControl" Path="SelectedItem" />
        </MultiBinding>
    </Rectangle.Visibility>
</Rectangle>
```

And it worked!

I was able to implement selection visuals entirely in XAML without the use of any wrappers, and that made me happy.

Unfortunately, this binding converter got more complex as I needed it to perform other duties. I wanted another Rectangle to dim unselected items, and I wanted this Rectangle to be hidden for the selected item, so I added a converter parameter. When this parameter is set to "true," the Visibility return values from the multi-binding converter are swapped.

But I also needed to switch between showing just the selected item and showing the whole band of items I associated with an IsActive property. If IsActive is true, all items need to be displayed; if IsActive is false, only the selected item need to be displayed. I added a facility to the multi-binding converter for a third object of type bool. If true, then the multi-binding converter always returns Visibility.Visible (unless the parameter is set to "true," in which case it returns Visibility.Hidden). This facility was used to make the entire item visible or hidden:

```

<Border.Visibility>
    <MultiBinding Converter="{StaticResource multiConverter}">
        <Binding />
        <Binding ElementName="monthControl" Path="SelectedItem" />
        <Binding ElementName="monthControl" Path="IsActive" />
    </MultiBinding>
</Border.Visibility>

```

Although the actual multi-binding converter isn't complex, it's a rather messy array of functionality. But I was able to implement selection visuals entirely in XAML without the use of any wrappers, and that made me happy.

Communicating with the Panel

The WindowedItemsControl needs to implement different scrolling logic if the panel set through its ItemsPanel property is a WrappableStackPanel. How can it tell? And how can it funnel information to this panel?

Determining the type of panel is fairly easy: Override the OnItemsPanelChanged property. Whenever the ItemsPanel property changes, this method is called with the old ItemsPanelTemplate and the new ItemsPanelTemplate. Call LoadContent on this new template and you get an instance of the panel in the template.

However, the panel returned from LoadContent isn't the same instance as the panel actually being used by the ItemsControl! This technique is suitable only for determining if the panel is of a particular type, not for communicating with that panel.

If the ItemsControl wishes to set properties on that panel, another technique is required. It could find the actual panel by traversing the visual tree, or it could use an inheritable event.

I chose the latter technique. In WindowedItemsControl, I defined a StartIndex property backed by a dependency property, to which I set the Inherits flag of FrameworkPropertyMetadataOptions (it's also well-known that to persuade this Inherits flag to work, you should register an attached property rather than a normal dependency property).

Here it is:

```

public static readonly DependencyProperty StartIndexProperty =
    DependencyProperty.RegisterAttached("StartIndex",
        typeof(double),
        typeof(WindowedItemsControl),
        new FrameworkPropertyMetadata(0.0,
            FrameworkPropertyMetadataOptions.Inherits));

```

When the ItemsPanel is a WrappableStackPanel, the WindowedItemsControl implements scrolling simply by setting a value to this StartIndex property.

The WrappableStackPanel adds an owner to the StartIndex attached property and sets the FrameworkPropertyMetadataOptions flag AffectsArrange. The ArrangeOverride method uses the value of the StartIndex property inherited from the ItemsControl to determine what item should appear at the top of the panel, and how much of it should be above the panel.

The Manipulation Events

As I suspected from the very outset, implementing the actual Manipulation events would be the hardest part of the whole job. It turned out that my analysis of the three types of touch events was right on target. They all had to be handled pretty much separately. (The code is in the WindowedItemsControl.Manipulation.cs file.)

Much of the determination of the three types of touch events occurs in the OnManipulationInertiaStarting override. This event indicates that the user's finger has left the screen.

If the ManipulationInertiaStarting event follows the ManipulationStarting event without any intervening ManipulationDelta events, that's a tap. The code in the ManipulationInertiaStarting event determines how far the tapped item needs to travel to move to the center, and then determines the velocity necessary to accomplish this in a fixed period of time (set at 250 milliseconds). It then initializes the DesiredDisplacement and InitialVelocity properties of the TranslationBehavior property of the event arguments for that amount of inertia.

Keep in mind that the user just tapped the item. The user didn't move the item, hence there's no actual velocity or inertia! But the ManipulationInertiaStarting event allows setting inertia parameters to force an object to move even if it hasn't been moving. In the context of handling the Manipulation events, this approach is much easier than using animation or CompositionTarget.Rendering for scrolling.

I must admit that the resultant control seems more like a "first draft" than a final version.

The logic is quite similar if the user has moved the band manually, but the new selected item is obvious because there's insufficient velocity to move the band beyond that item. Again, all that needs to be done is to set the DesiredDisplacement and InitialVelocity properties to slide it into place.

The really messy code occurs when there's actual inertia, and the new selected item won't be known until the velocity decreases and the scrolling almost stops. This is where velocity must be analyzed to examine if the new selected item will actually scroll beyond the center, and to reverse the direction if necessary. Some early code actually reversed the scrolling several times, back and forth, and that wasn't desirable at all.

How'd It Turn Out?

I must admit that the resultant control seems more like a "first draft" than a final version. It doesn't seem as smooth or as natural as the version implemented in Windows Phone 7, and it lacks some amenities. For example, when you activate the control by pressing on it, the bands in the Windows Phone 7 version fade into view and later fade out. Mine just pop.

But I'm happy for the experience, and I'm convinced even more that good multi-touch coding is a lot harder than it might at first seem. ■

CHARLES PETZOLD is a longtime contributing editor to MSDN Magazine. His new book, "Programming Windows Phone 7," will be published as a free downloadable ebook in the fall of 2010.

THANKS to the following technical expert for reviewing this column:
Doug Kramer



A Real Pain in the Neck

Not too long ago, the Microsoft.com homepage, shown here, was inflicting physical pain on its users. This pain was caused by deliberate decisions of professionals whose job it is to know better. Plattska hyperbole? Read and wince.

As on many homepages, the central display frame cycles through several highlight screens. The tabs beside the screen provide links to the highlights not currently displayed. The basic idea is sound, but this implementation stinks on several levels.

First, the tabs containing the links have text running vertically, which the reader has to tilt his head to read. Furthermore, the tabs and display pane scramble around as they cycle through the different displays: sometimes both tabs on the left, sometimes one on each side, sometimes both on the right.

This design violates what I call Plattska's Law of Minimum Chiropractic, which states that inflicting on your user an injury requiring chiropractic care will not make him happy; therefore you should do it as seldom as possible. The peer-reviewed "Journal of the Canadian Chiropractic Association" published a study (see tinyurl.com/2972ea5) on the prevalence of neck pain in the general population, in which 54 percent of all adults reported experiencing neck pain within the past six months, with almost 5 percent of all adults reporting neck pain of a disabling level. Microsoft should put themselves in the shoes—or neck braces—of its users, and stop this nonsense.

One Microsoft Web designer said to me, "It's just like a book. It looks cool. What's the problem?" Here's the problem: Book titles read this way because that's the most convenient storage of the books themselves. It's a compromise forced by the physical

medium. Don't inflict it on the virtual world where those physical constraints don't apply.

Besides neck pain, the tabs and display pane jump around confusingly as they cycle through the display topics. The display area starts on the left with two tabs on its right. After seven seconds, it jumps to the right with a tab on each side. After another seven seconds it jumps again to the right, now having two tabs on its left. Finally it jumps a double step left, back to the starting position.

The user has to visually reacquire the display pane every time the image changes. If he wants to click a link, sometimes that link is on the left; but a few seconds later that same link is on the right. This major navigation structure is a Whac-A-Mole game.

Why did Microsoft Web designers do this? Probably the usual suspect: the toolkit contained a pre-fabricated component that worked this way, and the designer just picked it out of the toolkit because it existed. It's like a 6 year old who got an Erector Set for Christmas, saying, "Look Ma, see what I can do! Isn't it cool?"

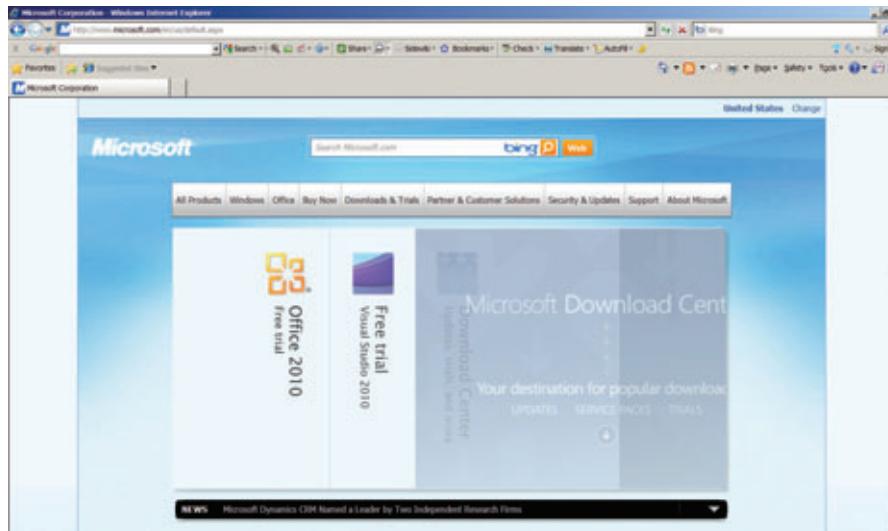
Inflicting physical pain on grownups is not cool. It's juvenile at best—malpractice at worst. It's the toy image that Microsoft acquired in the early 1990s and today is desperately trying to shed. This doesn't help.

The goal of rotating highlights with links can be accomplished far less painfully and more pleasingly, as in the MSN Lifestyle page. Users don't tilt their heads. The links always occupy the same location, as does the display area. It's easy to use, but apparently not cool enough for the main site designers.

Late update: The week this article was filed (Sept. 20), Microsoft unveiled a much-improved design. The highlight topics are shown all at once in separate panels. There's no motion, no need for any

links to display hidden topics. Maybe this column leaked and the designers realized their sins. They annoyed and confused many users in the meantime, though—I first saw the jumping, neck-bending design in April 2009. But now I know you readers are on the job, ready to pounce if they backslide. ■

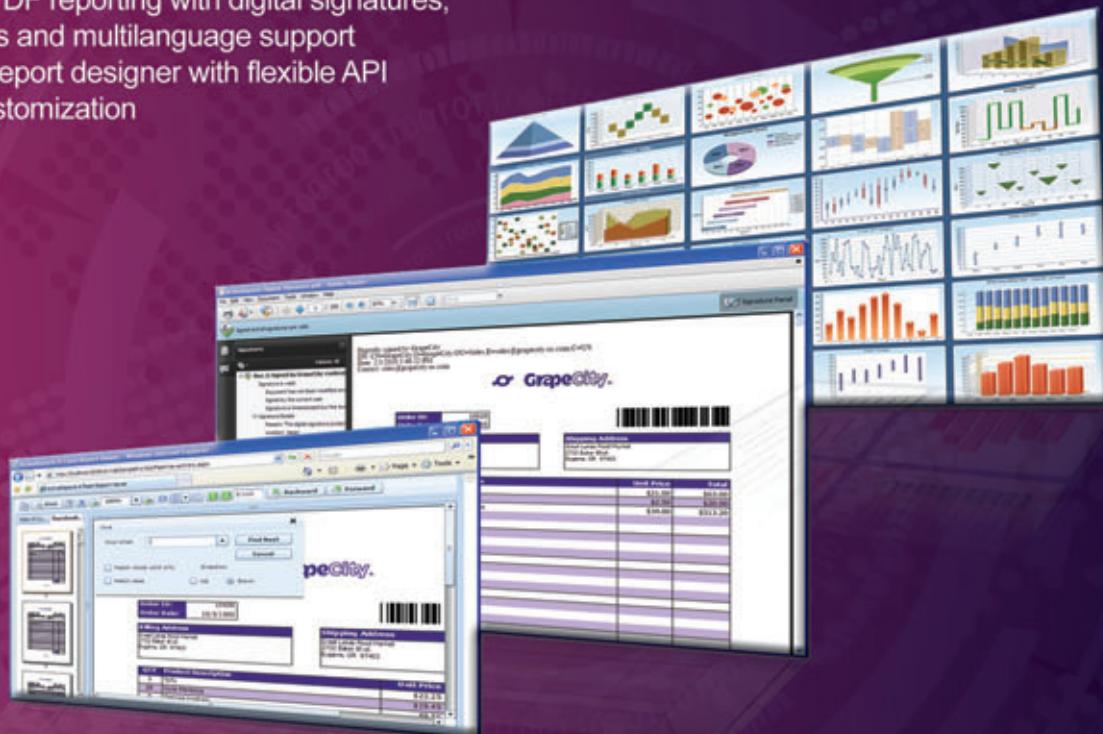
DAVID S. PLATT teaches Programming .NET at Harvard University Extension School and at companies all over the world. He's the author of 11 programming books, including "Why Software Sucks" (Addison-Wesley Professional, 2006) and "Introducing Microsoft .NET" (Microsoft Press, 2002). Microsoft named him 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



ACTIVE REPORTS



ACTIVE REPORTS

SPREAD

DATA DYNAMICS REPORTS

ACTIVE ANALYSIS

GrapeCity PowerTools
Report • Analyze • Excel

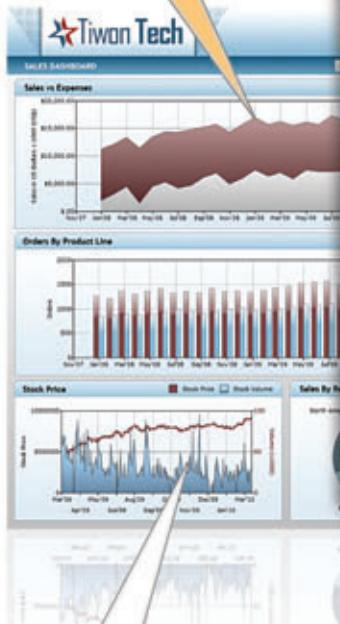
www.GCPowerTools.com



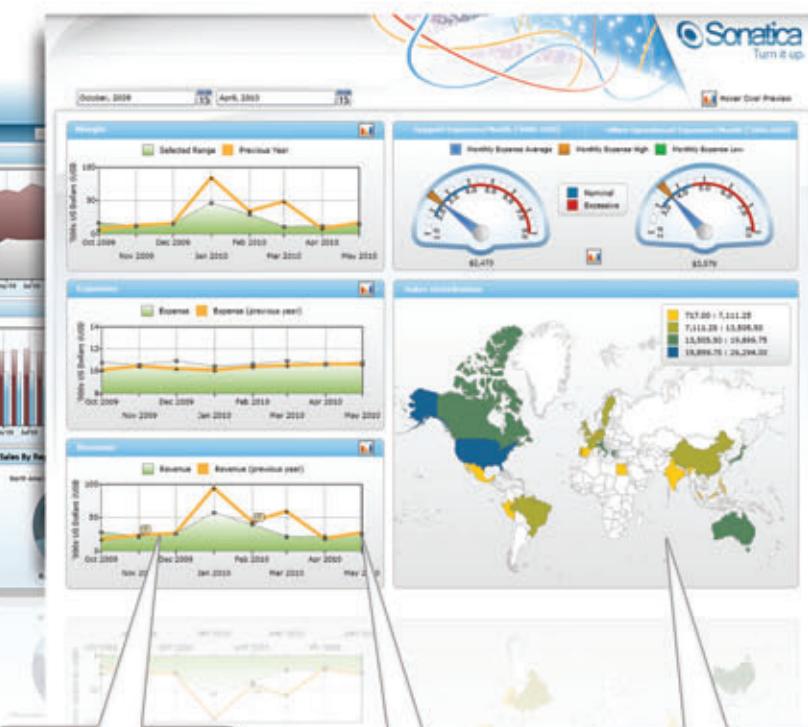
The Dashboard Platform for Developers like you

Build more powerful and effective dashboards, faster.

Support For Numerous Data Sources

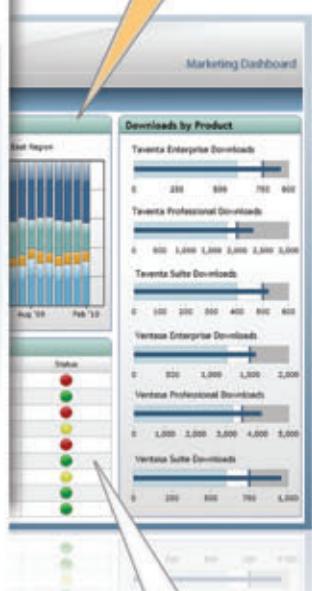


Rapid Dashboard Development



Full Scripting Capabilities With DundasScript™

Leverages The Latest Silverlight Technology



OLAP Support

Extensible And Customizable With An Open API

Dundas Dashboard is a ground-breaking, extensible solution that uses a revolutionary approach to dashboard creation, providing you with a unified view of key metrics and a new level of strategic insight and decision-making.



Powered by
Microsoft Silverlight



www.dundas.com/dashboard
(416) 467-5100 • (800) 463-1492

Silverlight is a trademark of Microsoft Corporation in the United States and/or other countries.