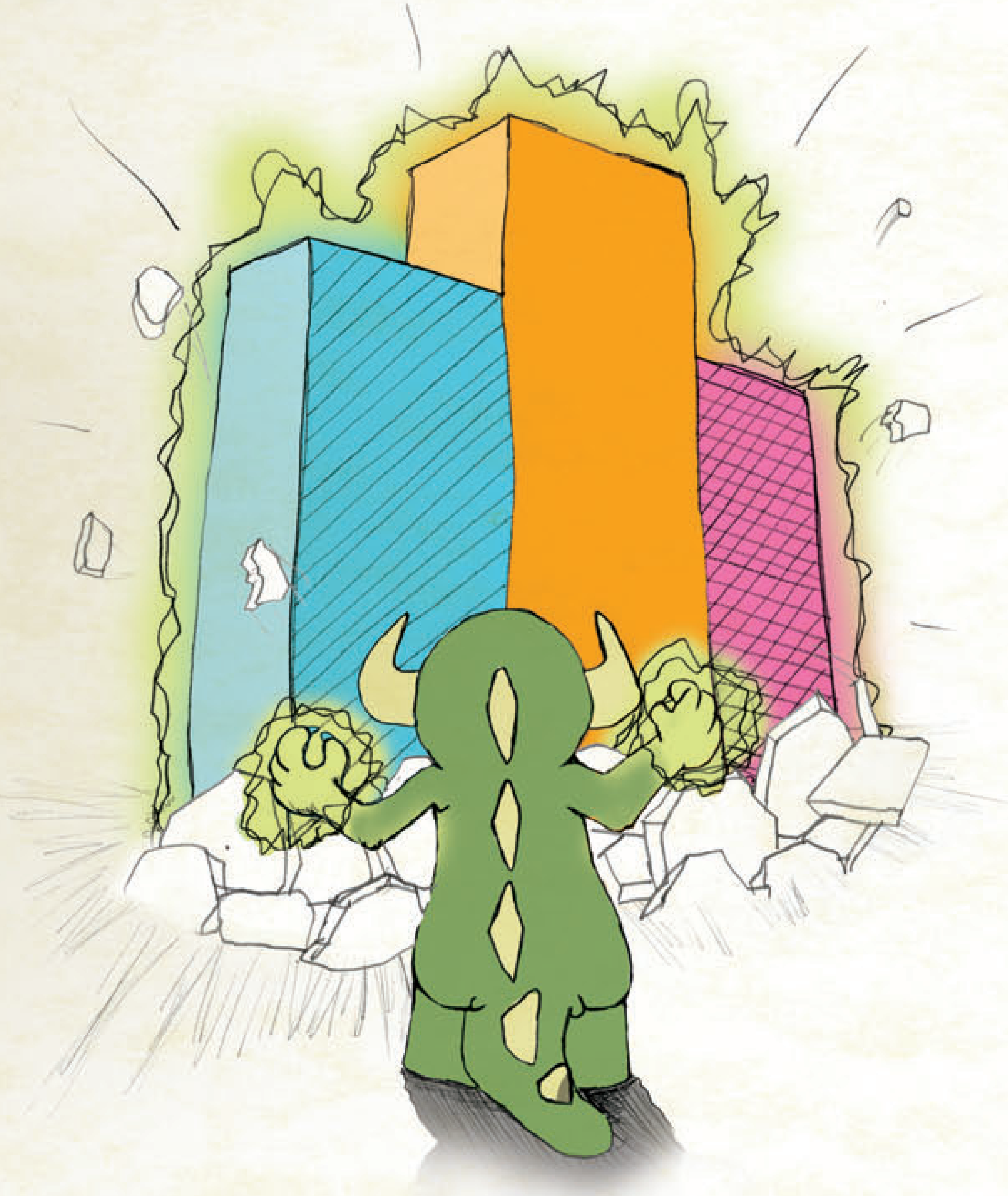
THE MICROSOFT JOURNAL FOR DEVELOPERS

JULY 2010 VOL 25 NO 7

msdn magazine

Microsoft



msdn® magazine

JULY 2010 VOLUME 25 NUMBER 7

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
MARTI LONGWORTH Associate Managing Editor

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

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

RedmondMediaGroup

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

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

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

1105 MEDIA INC

Neal Vitale President & Chief Executive Officer
Richard Vitale Senior Vice President & Chief Financial Officer
Michael J. Valenti Executive Vice President

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

Jeffrey S. Klein Chairman of the Board

MSDN Magazine (ISSN 1528-4859) is published monthly by 1105 Media, Inc., 9201 Oakdale Avenue, Ste. 101, Chatsworth, CA 91311. Periodicals postage paid at Chatsworth, CA 91311-9998, and at additional mailing offices. Annual subscription rates payable in U.S. funds: U.S. $35; Canada $45; International $60. Single copies/back issues: U.S. $10, all others $12. Send orders with payment to: MSDN Magazine, P.O. Box 3167, Carol Stream, IL 60132, e-mail MSDNmag@1105service.com or call 847-763-9560. **POSTMASTER:** Send address changes to *MSDN Magazine*, P.O. Box 2166, Skokie, IL 60076. Canada Publications Mail Agreement No: 40612608. Return Undeliverable Canadian Addresses to Circulation Dept. or IMS/NJ. Attn: Returns, 310 Paterson Plank Road, Carlstadt, NJ 07072.

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

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

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

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

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

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

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

Microsoft®

BPA WORLDWIDE℠

Printed in the USA

Your best source for software development tools!

programmer's paradise®

LEADTOOLS Recognition SDK

NEW RELEASE!

by LEAD Technologies

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

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

Paradise #
L05 26301A01

$3,214.99

programmers.com/LEAD

Multi-Edit^x

NEW RELEASE!

by Multi Edit Software

Multi-Edit^x is "The Solution" for your editing needs with support for over 50 languages. Edit plain text, ANY Unicode, hex, XML, HTML, PHP, Java, Javascript, Perl and more! No more file size limitations, unlimited line length, any file, any size Multi-Edit^x is "The Solution"!

Pre-Order Your Copy and Save!

1-49 Users
Paradise #
A30Z10101A01

$223.20

programmers.com/multiedit

VMware vSphere

Put time back into your day.

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

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

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

VMware
Advanced
Acceleration Kit
for 6 processors
Paradise #
V55 78101A01

$9,234.99

programmers.com/vSphere

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

Certified for Windows 7/2008R2

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

Paradise #
P35 04201A01

$550.99

programmers.com/pragma

InstallShield Professional for Windows

FLEXERA SOFTWARE™
InstallShield®

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.

Upgrade from
Active IS Pro +
IS Pro Silver Mtn
Paradise #
I21 02301S01

$1,399.00

programmers.com/flexera

VMware ThinApp Windows 7 Migration Promotion

SAVE 75%!

by VMware

Migration to the new Windows 7 OS is forcing companies to rethink their desktop delivery. VMware ThinApp is the easiest and most cost effective way to prepare for your Win 7 journey. By virtualizing your applications first with ThinApp, you will mitigate risk, dramatically speed up the migration process—and save money when you migrate to Windows 7!

Client Licenses
Minimum 500 with
Basic Level Support
Paradise #
V55 MIGRATION

LIMITED TIME OFFER!

$16.99

programmers.com/vmware

ActiveReports 6

NEW VERSION 6!

by GrapeCity

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

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

Professional Ed.
Paradise #
D03 04301A01

$1,310.99

programmers.com/grapecity

TX Text Control 15.1

NEW RELEASE!

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

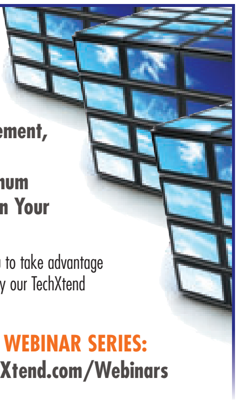
Professional Edition
Paradise #
T79 02101A02

$1,220.99

Download a demo today.

programmers.com/theimagingsource

BUILD ON VMWARE ESXi AND VSPHERE

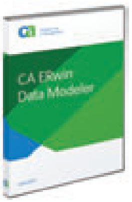


for Centralized Management, Continuous Application Availability, and Maximum Operational Efficiency in Your Virtualized Datacenter.

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

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

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



by CA

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

Paradise #
P26 04201E01

$3,931.99

programmers.com/ca

Intel Parallel Studio

by Intel

Intel Parallel Studio is a comprehensive Windows parallelism toolset designed for Microsoft Visual Studio C/C++ developers. Parallel Studio is interoperable with the widely used Microsoft Visual Studio, supports higher-level parallelism abstractions to simplify and speed development such as Intel Threading Building Blocks and Open MP, is fully supported, and provides an immediate opportunity to realize the benefits of multicore platforms. Tools are designed so novices can learn as they go, and professional developers can more easily bring parallelism to existing and new projects. Create optimized, innovative parallel applications and compete in a multicore industry.

Single User
Commercial
Paradise #
I23 63101A04

$753.99

programmers.com/intel

Microsoft Office Professional 2010

NEW RELEASE!

by Microsoft

Organize projects, manage finances and build a better way to do business with tools from Microsoft® Office Professional 2010. This software suite includes 2010 versions of Word, Excel®, PowerPoint®, OneNote®, Outlook®, Publisher® and Access®. It offers a Backstage™ view which replaces the traditional File menu to give you one go-to spot to conveniently save, open and print documents. Additionally, the server integration capabilities make it easier to track. Besides this, the Office Professional Plus 2010 also offers complete package through familiar intuitive tools.

Complete
DVD Windows
32/64 bit
Paradise #
M47 2130A01

$447.99

programmers.com/microsoft

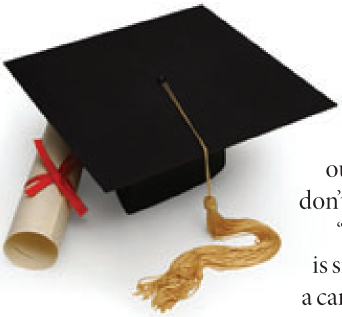
866-719-1528 **programmersparadise.com**

Prices subject to change. Not responsible for typographical errors.

EDITOR'S NOTE

KEITH WARD

Over-Educated, Yet Under-Qualified?



As we get going on the latest mini-bounce-back on what looks like an extremely long road to economic recovery, there is some good news: It looks like the tech sector may have a quicker—and higher—bounce than other industries. We're finally getting some news that shows solid, sustained job growth in all areas of IT, including software development.

But as Adrian Monk, my favorite TV detective, would say, "Here's the thing …" Is this great news for you if you're hiring coders but can't pay them a lot yet, which might mean plucking fresh fruit off the college tree? Because I've been reading some worrisome stuff about the quality of education computer science grads are getting, and the heartburn it's causing both the grads and potential employers.

My concern was piqued by an article I saw on the InfoWorld Web site, "The sad standards of computer-related college degrees" (bit.ly/bIp267). A concerned father wrote in about his daughter's lack of preparedness for the world of real work. He writes: "Imagine my surprise (and, as it turned out, her relief) that she could get a four-year undergraduate degree in "data processing" without having to write a single program in any language!

"This seems to be a trend," the writer continues. "In an effort to widen and deepen my own skill set, I have had occasion to examine computer science course material available online from a number of top-tier colleges and some from the lower rungs. In most instances, what I remember from my nearly 40-year-old computer science education still places me far ahead of what they are now teaching." And he concludes: "We've had trouble finding qualified U.S. job applicants who want to do the work we need done. I wonder if there's a connection."

The comments from readers accompanying the article support the writer's contention, for the most part. Here's a sampling:

From "rsr," who claims to be a former computer science professor: "Computer Science (and related computer program) enrollments have greatly declined, and schools are trying to reverse the trend. This includes making the programs easier so there will be fewer drop-outs and it will be more attractive to students who don't want to work hard but still get a degree."

"Woking," a manager at a "Fortune 500 company," is similarly unimpressed. "I have never interviewed a candidate right out of college who I would hire. No recent graduate that I have interviewed has had sufficient understanding of real-world problems to be useful to me, at least for the salary that the interviewees were expecting."

Woking gives a specific example: "Several years ago I interviewed candidates for an open position as a data modeler. None of the recent college graduates who had even covered Entity Relationship Diagramming in their programs had created a data model with more than five entities." Woking says that they have better success hiring candidates with three to five years work experience, even if the applicant lacks a college degree. That's a pretty damning statement.

"Beney," with 20-plus years experience and no IT degree, puts it succinctly: "Maybe if IT students had to actually write code rather than manipulate IDEs, they'd at least be able to handle the real world when they get out into the job market."

Pretty discouraging stuff. What I'd like to do is use the power of the MSDN network to help determine if we're facing a crisis when it comes to teaching college kids proper software development skills. If you're a computer science professor, recent computer science graduate, hiring manager or anyone else with insight into this issue, let me know your thoughts at mmeditor@microsoft.com. If you agree that this is a general failing of the education system, explain how you'd change things: What are the top two or three things you'd do? I'm looking forward to reading your responses. After all, if there's a job to be filled, it makes sense that it be filled with a developer who can actually do that job.

Keith Ward

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

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.

FREE! 1-User License

Introducing

OnTime 2010

NEW! Scrum Planning Board that Changes Everything!

OnTime EXPRESS
OnTime ENTERPRISE
axosoft
OnTime 2010
Burn-Down
Planning Board
Expand login help screen
Expand product class to store
Add hosted features for east coast users
Add new features to product pages
create new user rights for
Create new product comparison pages
Add HTML text control to 'About Me' pages
User would like to able to search the handbook
Update financial picklists for optional entry
create new security right for admin of success stories
Add ability to use backforward buttons within the browser
Add search to company publications
remove antiquated entry builder
Create new security rolls
Update request module to allow multiple entries
create screenshots for Quickhelp

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

Project Management • Bug Tracking • Scrum

Sign up for *OnTime Now* (in the Cloud) or install locally!

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

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

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

Planning Board
The NEW planning board makes this both the most powerful & easiest to use version of OnTime EVER!

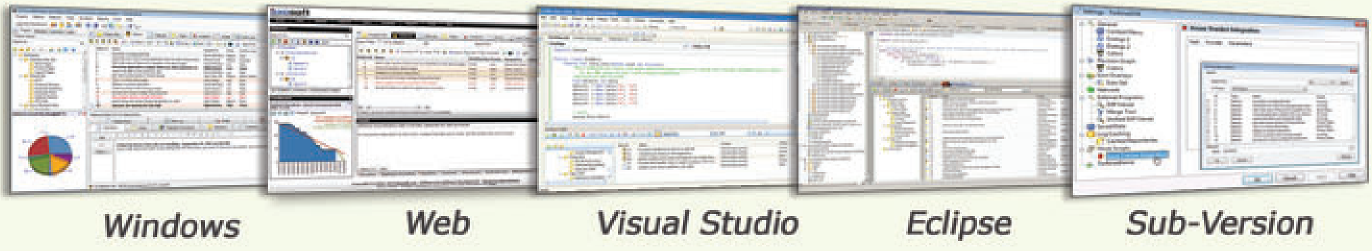
Workflow
Define a hyper-fast, lightweight, agile workflow or a robust process-enforcement system.

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

Tortoise SVN
Perfect for Subversion Teams! The new TortoiseSVN plug-in for OnTime 2010 provides tight integration.

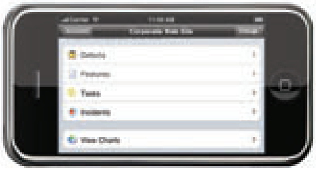
Help Desk
Use OnTime for tracking Help Desk incidents - even monitor email accounts to generate help tickets.

OnTime Now!
Cloud project management done right! During signup, speed-test multiple data centers & pick the quickest.



axosoft

FREE 1-User License • FREE 30-day Team Trials



800.653.0024
www.axosoft.com

CUTTING EDGE

DINO ESPOSITO

Expando Objects in C# 4.0

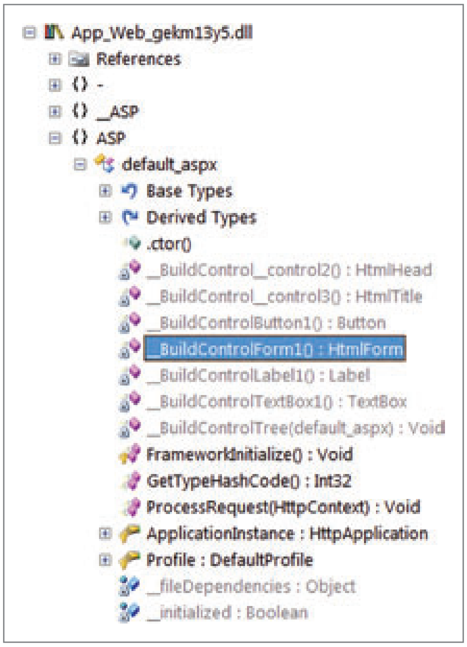
Most of the code written for the Microsoft .NET Framework is based on static typing, even though .NET supports dynamic typing via reflection. Moreover, JScript had a dynamic type system on top of the .NET 10 years ago, as did Visual Basic. Static typing means that every expression is of a known type. Types and assignments are validated at compile time and most of the possible typing errors are caught in advance.

The well-known exception is when you attempt a cast at run time, which may sometimes result in a dynamic error if the source type is not compatible with the target type.

Static typing is great for performance and for clarity, but it's based on the assumption that you know nearly everything about your code (and data) beforehand. Today, there's a strong need for relaxing this constraint a bit. Going beyond static typing typically means looking at three distinct options: dynamic typing, dynamic objects, and indirect or reflection-based programming.

In .NET programming, reflection has been available since the .NET Framework 1.0 and has been widely employed to fuel special frameworks, like Inversion of Control (IoC) containers. These frameworks work by resolving type dependencies at run time, thus enabling your code to work against an interface without having to know the concrete type behind the object and its actual behavior. Using .NET reflection, you can implement forms of indirect programming where your code talks to an intermediate object that in turn dispatches calls to a fixed interface. You pass the name of the member to invoke as a string, thus granting yourself the flexibility of reading it from some external source. The interface of the target object is fixed and immutable—there's always a well-known interface behind any calls you place through reflection.

Dynamic typing means that your compiled code ignores the static structure of types that can be detected at compile time. In fact, dynamic typing delays any type checks until run time. The interface you code against is still fixed and immutable, but the value you use may return different interfaces at different times.



```
⊟ App_Web_gekm13y5.dll
    ⊞ References
    ⊞ {} -
    ⊞ {} _ASP
    ⊟ {} ASP
        ⊟ default_aspx
            ⊞ Base Types
            ⊞ Derived Types
              .ctor()
              __BuildControl__control2() : HtmlHead
              __BuildControl__control3() : HtmlTitle
              __BuildControlButton1() : Button
              __BuildControlForm1() : HtmlForm
              __BuildControlLabel1() : Label
              __BuildControlTextBox1() : TextBox
              __BuildControlTree(default_aspx) : Void
              FrameworkInitialize() : Void
              GetTypeHashCode() : Int32
              ProcessRequest(HttpContext) : Void
            ⊞ ApplicationInstance : HttpApplication
            ⊞ Profile : DefaultProfile
              __fileDependencies : Object
              __initialized : Boolean
```

Figure 1 **The Structure of a Dynamically Created Web Forms Class**

The .NET Framework 4 introduces some new features that enable you to go beyond static types. I covered the new dynamic keyword in the May 2010 issue (msdn.microsoft.com/magazine/ee336309). In this article, I'll explore the support for dynamically defined types such as expando objects and dynamic objects. With dynamic objects, you can define the interface of the type programmatically instead of reading it from a definition statically stored in some assemblies. Dynamic objects wed the formal cleanliness of static typed objects with the flexibility of dynamic types.

Scenarios for Dynamic Objects

Dynamic objects are not here to replace the good qualities of static types. Static types are, and will remain for the foreseeable future, at the foundation of software development. With static typing, you can find type errors reliably at compile time and produce code that, because of this, is free of runtime checks and runs faster. In addition, the need to pass the compile step leads developers and architects to take care in the design of the software and in the definition of public interfaces for interacting layers.

There are, however, situations in which you have relatively well-structured blocks of data to be consumed programmatically. Ideally, you'd love to have this data exposed through objects. But, instead, whether it reaches you over a network connection or you read it from a disk file, you receive it as a plain stream of data. You have two options to work against this data: using an indirect approach or using an ad hoc type.

In the first case, you employ a generic API that acts as a proxy and arranges queries and updates for you. In the second case, you have a specific type that perfectly models the data you're working with. The question is, who's going to create such an ad hoc type?

In some segments of the .NET Framework, you already have good examples of internal modules creating ad hoc types for

Code download available at code.msdn.microsoft.com/mag201007CutEdge.

LEADTOOLS®

The World Leader in Imaging Development SDKs

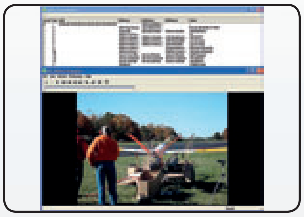
Silverlight, .NET, WPF, WCF, WF, C API, C++ Class Lib, COM & more!

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.**

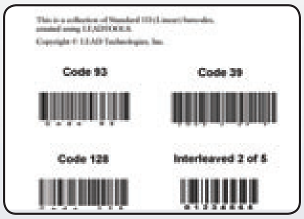
LEADTOOLS provides developers easy access to decades of expertise in color, grayscale, document, medical, vector and multimedia imaging development. Install LEADTOOLS to eliminate months of research and programming time while maintaining high levels of quality, performance and functionality.

- **•Image Formats & Compression:** Supports 150+ image formats and compressions including TIFF, EXIF, PDF, JPEG2000, JBIG and CCITT.
- **•Display Controls:** ActiveX, COM, Win Forms, Web Forms, WPF and Silverlight.
- **•Image Processing:** 200+ filters, transforms, color conversion and drawing functions supporting region of interest and extended grayscale data.
- **•OCR/ICR/OMR:** Full page or zonal recognition for multithreaded 32 and 64 bit development.
- **•Forms Recognition and Processing:** Automatically identify forms and extract user filled data.
- **•Barcode:** Detect, read and write 1D and 2D barcodes for multithreaded 32 and 64 bit development.
- **•Document Cleanup/Preprocessing:** Deskew, despeckle, hole punch, line and border removal, inverted text correction and more.
- **•PDF and PDF/A:** Read and write searchable PDF with text, images and annotations.
- **•Annotations:** Interactive UI for document mark-up, redaction and image measurement (including support for DICOM annotations).
- **•Medical Web Viewer Framework:** Plug-in enabled framework to quickly build high-quality, full-featured, web-based medical image delivery and viewer applications.
- **•Medical Image Viewer:** High level display control with built-in tools for image mark-up, window level, measurement, zoom/pan, cine, and LUT manipulation.
- **•DICOM:** Full support for all IOD classes and modalities defined in the 2008 DICOM standard (including Encapsulated PDF/CDA and Raw Data).
- **•PACS Communications:** Full support for DICOM messaging and secure communication enabling quick implementation of any DICOM SCU and SCP services.
- **•JPIP:** Client and Server components for interactive streaming of large images and associated image data using the minimum possible bandwidth.
- **•Scanning:** TWAIN 2.0 and WIA (32 and 64-bit), autodetect optimum driver settings for high speed scanning.
- **•DVD:** Play, create, convert and burn DVD images.
- **•DVR:** Pause, rewind and fast-forward live capture and UDP or TCP/IP streams.
- **•Multimedia:** Capture, play, stream and convert MPEG, AVI, WMV, MP4, MP3, OGG, ISO, DVD and more.
- **•Enterprise Development:** Includes WCF services and WF activities to create scalable, robust enterprise applications.

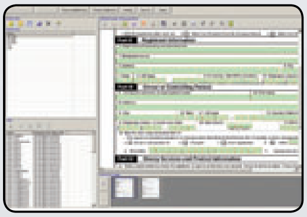
Multimedia



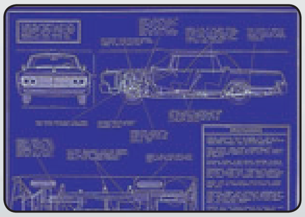
Barcode



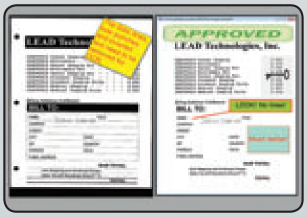
Form Recognition & Processing



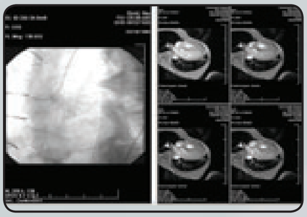
Vector



Document



DICOM Medical



High Level Design • Low Level Control

LEAD TECHNOLOGIES INCORPORATED

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

Figure 2 **Using LINQ-to-XML to Load Data into a Person Object**

```
var persons = GetPersonsFromXml(file);
foreach(var p in persons)
  Console.WriteLine(p.GetFullName());

// Load XML data and copy into a list object
var doc = XDocument.Load(@"..\..\sample.xml");
public static IList<Person> GetPersonsFromXml(String file) {
  var persons = new List<Person>();

  var doc = XDocument.Load(file);
  var nodes = from node in doc.Root.Descendants("Person")
              select node;

  foreach (var n in nodes) {
    var person = new Person();
    foreach (var child in n.Descendants()) {
      if (child.Name == "FirstName")
        person.FirstName = child.Value.Trim();
      else
        if (child.Name == "LastName")
          person.LastName = child.Value.Trim();
    }
    persons.Add(person);
  }

  return persons;
}
```

specific blocks of data. One obvious example is ASP.NET Web Forms. When you place a request for an ASPX resource, the Web server retrieves the content of the ASPX server file. That content is then loaded into a string to be processed into an HTML response. So you have a relatively well-structured piece of text with which to work.

To do something with this data, you need to understand what references you have to server controls, instantiate them properly and link them together into a page. This can be definitely done using an XML-based parser for each request. In doing so, though, you end up paying the extra costs of the parser for every request, which is probably an unacceptable cost.

Due to these added costs of parsing data, the ASP.NET team decided to introduce a one-time step to parse the markup into a class that can be dynamically compiled. The result is that a simple chunk of markup like this is consumed via an ad hoc class derived from the code-behind class of the Web Forms page:

```
<html>
<head runat="server">
  <title></title>
</head>
<body>
  <form id="Form1" runat="server">
    <asp:TextBox runat="server" ID="TextBox1" />
    <asp:Button ID="Button1" runat="server" Text="Click" />
    <hr />
    <asp:Label runat="server" ID="Label1"></asp:Label>
  </form>
</body>
</html>
```

Figure 1 shows the runtime structure of the class created out of the markup. The method names in gray refer to internal procedures used to parse elements with the runat=server elements into instances of server controls.

You can apply this approach to nearly any situation in which your application receives external data to process repeatedly. For example, consider a stream of XML data that flows into the application. There are several APIs available to deal with XML data, from XML DOM to LINQ-to-XML. In any case, you have to either proceed indirectly by querying the XML DOM or LINQ-to-XML API, or use the same APIs to parse the raw data into ad hoc objects.

In the .NET Framework 4, dynamic objects offer an alternative, simpler API to create types dynamically based on some raw data. As a quick example, consider the following XML string:

```
<Persons>
  <Person>
    <FirstName> Dino </FirstName>
    <LastName> Esposito </LastName>
  </Person>
  <Person>
    <FirstName> John </FirstName>
    <LastName> Smith </LastName>
  </Person>
</Persons>
```

To transform that into a programmable type, in the .NET Framework 3.5 you'd probably use something like the code in **Figure 2**.

The code uses LINQ-to-XML to load raw content into an instance of the Person class:

```
public class Person {
  public String FirstName { get; set; }
  public String LastName { get; set; }
  public String GetFullName() {
    return String.Format("{0}, {1}", LastName, FirstName);
  }
}
```

The .NET Framework 4 offers a different API to achieve the same thing. Centered on the new ExpandoObject class, this API is more direct to write and doesn't require that you plan, write, debug, test and maintain a Person class. Let's find out more about ExpandoObject.

Dynamic typing delays any type checks until run time.

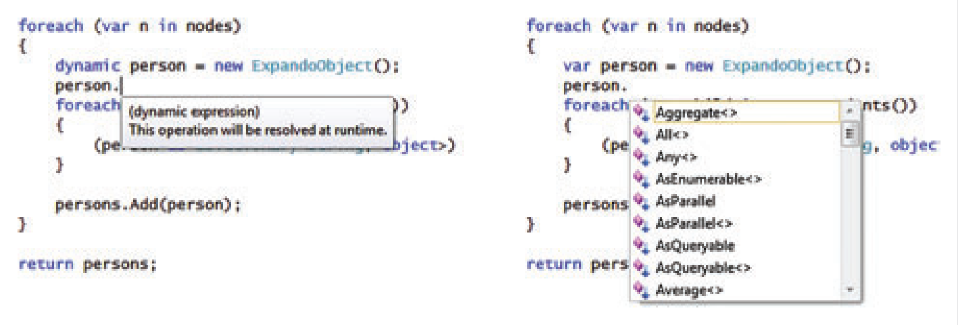
Using the ExpandoObject Class

Expando objects were not invented for the .NET Framework; in fact, they appeared several years before .NET. I first heard the term used to describe JScript objects in the mid-1990s. An expando is a sort of inflatable object whose structure is entirely defined at run time. In the .NET Framework 4, you use it as if it were a classic managed object except that its structure is not read out of any assembly, but is built entirely dynamically.

Figure 3 **Using LINQ-to-XML to Load Data into an Expando Object**

```
public static IList<dynamic> GetExpandoFromXml(String file) {
  var persons = new List<dynamic>();

  var doc = XDocument.Load(file);
  var nodes = from node in doc.Root.Descendants("Person")
              select node;
  foreach (var n in nodes) {
    dynamic person = new ExpandoObject();
    foreach (var child in n.Descendants()) {
      var p = person as IDictionary<String, object>);
      p[child.Name] = child.Value.Trim();
    }

    persons.Add(person);
  }

  return persons;
}
```

DXGrid
Card Templates
ASPxPivotGrid
Custom Chart Data
DevExpress



```
foreach (var n in nodes)
{
    dynamic person = new ExpandoObject();
    person.
    foreach                                        ))
    {
        (pe                                        bject>)
    }

    persons.Add(person);
}

return persons;
```

(dynamic expression)
This operation will be resolved at runtime.

```
foreach (var n in nodes)
{
    var person = new ExpandoObject();
    person.
    foreach                          nts())
    {
        (pe                          g, objec
    }

    persons
}

return pers
```

Figure 4 **Visual Studio 2010 IntelliSense and Expando Objects**

An expando object is ideal to model dynamically changing information such as the content of a configuration file. Let's see how to use the ExpandoObject class to store the content of the aforementioned XML document. The full source code is shown in **Figure 3**.

The function returns a list of dynamically defined objects. Using LINQ-to-XML, you parse out the nodes in the markup and create an ExpandoObject instance for each of them. The name of each node below <Person> becomes a new property on the expando object. The value of the property is the inner text of the node. Based on the XML content, you end up with an expando object with a FirstName property set to Dino.

In **Figure 3**, however, you see an indexer syntax used to populate the expando object. That requires a bit more explanation.

Inside the ExpandoObject Class

The ExpandoObject class belongs to the System.Dynamic namespace and is defined in the System.Core assembly. ExpandoObject represents an object whose members can be dynamically added and removed at run time. The class is sealed and implements a number of interfaces:

```
public sealed class ExpandoObject :
  IDynamicMetaObjectProvider,
  IDictionary<string, object>,
  ICollection<KeyValuePair<string, object>>,
  IEnumerable<KeyValuePair<string, object>>,
  IEnumerable,
  INotifyPropertyChanged;
```

As you can see, the class exposes its content using various enumerable interfaces, including IDictionary<String, Object> and IEnumerable. In addition, it also implements IDynamicMetaObjectProvider. This is the standard interface that enables an object to be shared within the Dynamic Language Runtime (DLR) by programs written in accordance with the DLR interoperability model. In other words, only objects that implement the IDynamicMetaObjectProvider interface can be shared across .NET dynamic languages. An expando object can be passed to, say, an IronRuby component. You can't do that easily with a regular .NET managed object. Or, rather, you can, but you just don't get the dynamic behavior.

The ExpandoObject class also implements the INotifyPropertyChanged interface. This enables the class to raise a PropertyChanged event when a member is added or modified. Support of the INotifyPropertyChanged interface is key to using expando objects in Silverlight and Windows Presentation Foundation application front ends.

You create an ExpandoObject instance as you do with any other .NET object, except that the variable to store the instance is of type dynamic:

```
dynamic expando = new ExpandoObject();
```

At this point, to add a property to the expando you simply assign it a new value, as below:

```
expando.FirstName = "Dino";
```

It doesn't matter that no information exists about the FirstName member, its type or its visibility. This is dynamic code; for this reason, it makes a huge difference if you use the var keyword to assign an ExpandoObject instance to a variable:

```
var expando = new ExpandoObject();
```

This code will compile and work just fine. However, with this definition you're not allowed to assign any value to a FirstName property. The ExpandoObject class as defined in System.Core has no such member. More precisely, the ExpandoObject class has no public members.

This is a key point. When the static type of an expando is dynamic, the operations are bound as dynamic operations, including looking up members. When the static type is ExpandoObject, then operations are bound as regular compile-time member lookups. So the compiler knows that dynamic is a special type, but does not know that ExpandoObject is a special type.

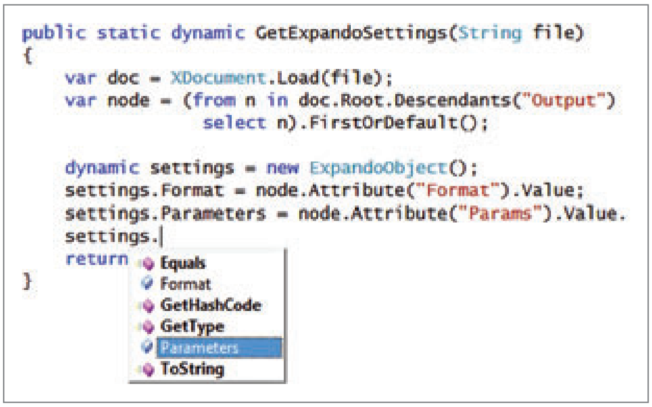
In **Figure 4**, you see the Visual Studio 2010 IntelliSense options when an expando object is declared as a dynamic type and when it's treated as a plain .NET object. In the latter case, IntelliSense shows you the default System.Object members plus the list of extension methods for collection classes.

It should also be noted that some commercial tools in some circumstances go beyond this basic behavior. **Figure 5** shows ReSharper 5.0, which captures the list of members currently defined on the object. This doesn't happen if members are added programmatically via an indexer.

To add a method to an expando object, you just define it as a property, except you use an Action<T> or Func<T> delegate to express the behavior. Here's an example:

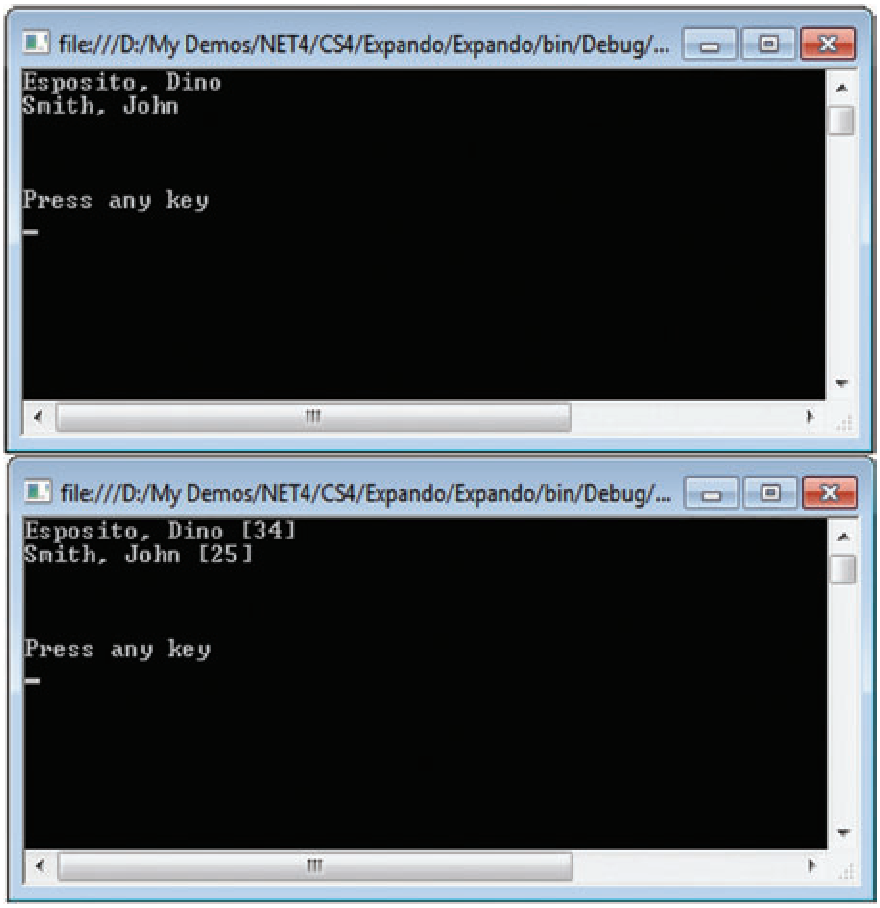
```
person.GetFullName = (Func<String>)(() => {
  return String.Format("{0}, {1}",
    person.LastName, person.FirstName);
});
```

The method GetFullName returns a String obtained by combining the last name and first name properties assumed to be available on



```
public static dynamic GetExpandoSettings(String file)
{
    var doc = XDocument.Load(file);
    var node = (from n in doc.Root.Descendants("Output")
                select n).FirstOrDefault();

    dynamic settings = new ExpandoObject();
    settings.Format = node.Attribute("Format").Value;
    settings.Parameters = node.Attribute("Params").Value.
    settings.
    return
}
```

Figure 5 **The ReSharper 5.0 IntelliSense at Work with Expando Objects**



file:///D:/My Demos/NET4/CS4/Expando/Expando/bin/Debug/...

```
Esposito, Dino
Smith, John


Press any key
_
```

file:///D:/My Demos/NET4/CS4/Expando/Expando/bin/Debug/...

```
Esposito, Dino [34]
Smith, John [25]


Press any key
_
```

Figure 6 **Two Sample Console Applications Driven by an XML File**

the expando object. If you attempt to access a missing member on expando objects, you'll receive a RuntimeBinderException exception.

XML-Driven Programs

To tie together the concepts I've shown you so far, let me guide you through an example where the structure of the data and the structure of the UI are defined in an XML file. The content of the file is parsed to a collection of expando objects and processed by the application. The application, however, works only with dynamically presented information and is not bound to any static type.

The code in **Figure 3** defines a list of dynamically defined person expando objects. As you'd expect, if you add a new node to the XML schema, a new property will be created in the expando object. If you need to read the name of the member from an external source, you should employ the indexer API to add it to the expando. The ExpandoObject class implements the IDictionary<String, Object> interface explicitly. This means you need to segregate the ExpandoObject interface from the dictionary type in order to use the indexer API or the Add method:

```
(person as IDictionary<String, Object>)[child.Name] = child.Value;
```

Because of this behavior, you just need to edit the XML file to make a different data set available. But how can you consume this dynamically changing data? Your UI will need to be flexible enough to receive a variable set of data.

Let's consider a simple example where all you do is display data through the console. Suppose the XML file contains a section that describes the expected UI—whatever that means in your context. For the purpose of example, here's what I have:

```
<Settings>
  <Output Format="{0}, {1}"
    Params="LastName,FirstName" />
</Settings>
```

This information will be loaded into another expando object using the following code:

```
dynamic settings = new ExpandoObject();
settings.Format =
  node.Attribute("Format").Value;
settings.Parameters =
  node.Attribute("Params").Value;
```

The main procedure will have the following structure:

```
public static void Run(String file) {
  dynamic settings = GetExpandoSettings(file);
  dynamic persons = GetExpandoFromXml(file);
  foreach (var p in persons) {
    var memberNames =
      (settings.Parameters as String).
      Split(',');
    var realValues =
      GetValuesFromExpandoObject(p,
      memberNames);
    Console.WriteLine(settings.Format,
      realValues);
  }
}
```

The expando object contains the format of the output, plus the names of the members whose values are to be displayed. Given the person dynamic object, you need to load the values for the specified members, using code like this:

```
public static Object[] GetValuesFromExpandoObject(
  IDictionary<String, Object> person,
  String[] memberNames) {

  var realValues = new List<Object>();
  foreach (var m in memberNames)
    realValues.Add(person[m]);
  return realValues.ToArray();
}
```

Because an expando object implements IDictionary<String, Object>, you can use the indexer API to get and set values.

Finally, the list of values retrieved from the expando object are passed to the console for actual display. **Figure 6** shows two screens for the sample console application, whose only difference is the structure of the underlying XML file.

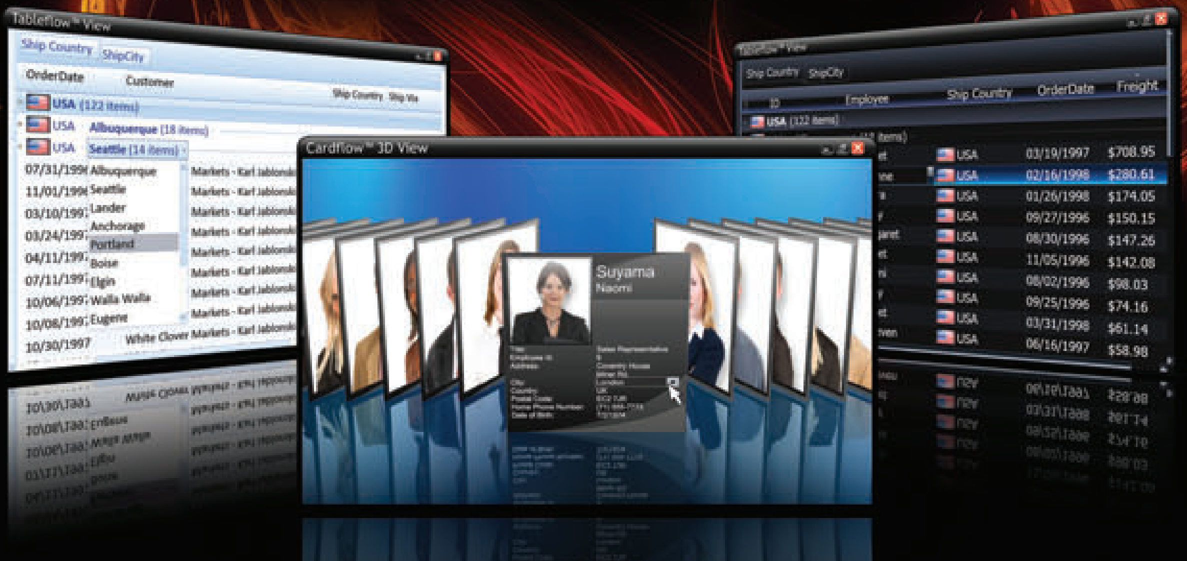
Admittedly, this is a trivial example, but the mechanics required to make it work are similar to that of more interesting examples. Try it out and share your feedback! ■

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

THANKS to the following technical expert for reviewing this article: *Eric Lippert*

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



Tableflow™ View

Ship Country | ShipCity

OrderDate | Customer | Ship Country | Ship Via

USA (122 items)

USA Albuquerque (18 items)

USA Seattle (14 items)

07/31/199	Albuquerque	Markets - Karl Jablonsk
11/01/199	Seattle	Markets - Karl Jablonsk
03/10/199	Lander	Markets - Karl Jablonsk
03/24/199	Anchorage	Markets - Karl Jablonsk
04/11/199	Portland	Markets - Karl Jablonsk
07/11/199	Boise	Markets - Karl Jablonsk
10/06/199	Elgin	Markets - Karl Jablonsk
10/08/199	Walla Walla	Markets - Karl Jablonsk
	Eugene	Markets - Karl Jablonsk
10/30/1997	White Clover	Markets - Karl Jablonsk

Cardflow™ 3D View

Suyama
Naomi

Title:	Sales Representative
Employee ID:	6
Address:	Coventry House Miner Rd.
City:	London
Country:	UK
Postal Code:	EC2 7JR
Home Phone Number:	(71) 555-7773
Date of Birth:	[illegible]

Tableflow™ View

Ship Country | ShipCity

USA (122 items)

ID	Employee	Ship Country	OrderDate	Freight
	et	USA	03/19/1997	$708.95
	ne	USA	02/16/1998	$280.61
		USA	01/26/1998	$174.05
		USA	09/27/1996	$150.15
	aret	USA	08/30/1996	$147.26
	et	USA	11/05/1996	$142.08
	ni	USA	08/02/1996	$98.03
		USA	09/25/1996	$74.16
	et	USA	03/31/1998	$61.14
	even	USA	06/16/1997	$58.98

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

Windows Azure Table Storage—Not Your Father's Database

Windows Azure Table storage causes a lot of head scratching among developers. Most of their experience with data storage is with relational databases that have various tables, each containing a predefined set of columns, one or more of which are typically designated as identity keys. Tables use these keys to define relationships among one another.

Windows Azure stores information a few ways, but the two that focus on persisting structured data are SQL Azure and Windows Azure Table storage. The first is a relational database and aligns fairly closely with SQL Server. It has tables with defined schema, keys, relationships and other constraints, and you connect to it using a connection string just as you do with SQL Server and other databases.

Windows Azure Table storage, on the other hand, seems a bit mysterious to those of us who are so used to working with relational databases. While you'll find many excellent walk-throughs for creating apps that use Windows Azure Table storage, many developers still find themselves forced to make leaps of faith without truly understanding what it's all about.

This column will help those stuck in relational mode bridge that leap of faith with solid ground by explaining some core concepts of Windows Azure Table storage from the perspective of relational thinking. Also, I'll touch on some of the important strategies for designing the tables, depending on how you expect to query and update your data.

> Windows Azure Table storage seems a bit mysterious to those of us who are so used to working with relational databases.

Storing Data for Efficient Retrieval and Persistence

By design, Windows Azure Table services provides the potential to store enormous amounts of data, while enabling efficient access and persistence. The services simplify storage, saving you from jumping through all the hoops required to work with a relational database—constraints, views, indices, relationships and stored procedures. You just deal with data, data, data. Windows Azure Tables use keys that enable efficient querying, and you can employ one—the PartitionKey—for load balancing when the table service decides it's time to spread your table over multiple servers. A table doesn't have a specified schema. It's simply a structured container of rows (or entities) that doesn't care what a row looks like. You can have a table that stores one particular type, but you can also store rows with varying structures in a single table, as shown in **Figure 1**.

It All Begins with Your Domain Classes

Our typical development procedure with databases is to create them, define tables in them and then, for every table, define a particular structure—specific columns, each with a specified data type—as well as relationships to other tables. Our applications then push data into and pull data out of the tables.

With Windows Azure Table services, though, you don't design a database, just your classes. You define your classes and a container (table) that one or more classes belong to, then you can save instantiated objects back to the store as rows.

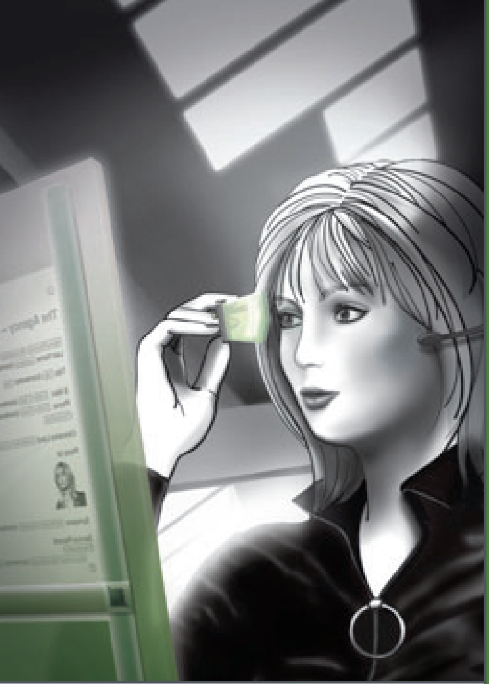
In addition to the properties you need in your classes, each class must have three properties that are critical in determining how Windows Azure Table services do their job: PartitionKey, RowKey and TimeStamp. PartitionKey and RowKey are both strings, and there's an art (or perhaps a science) to defining them so you get the best balance of query and transaction efficiency along with scalability at run time. For a good understanding of how to define PartitionKeys and RowKeys for the most benefit, I highly recommend the PDC09 session "Windows Azure Tables and Queues Deep Dive," presented by Jai Haridas, which you can watch at microsoftpdc.com/sessions/svc09.

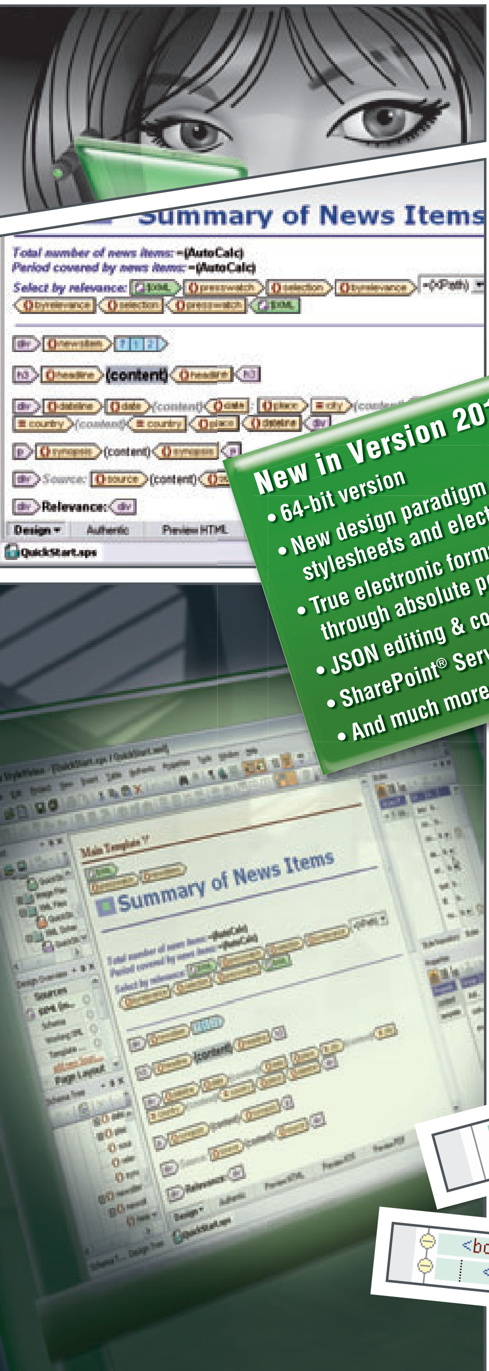
PartitionKeys and RowKeys Drive Performance and Scalability

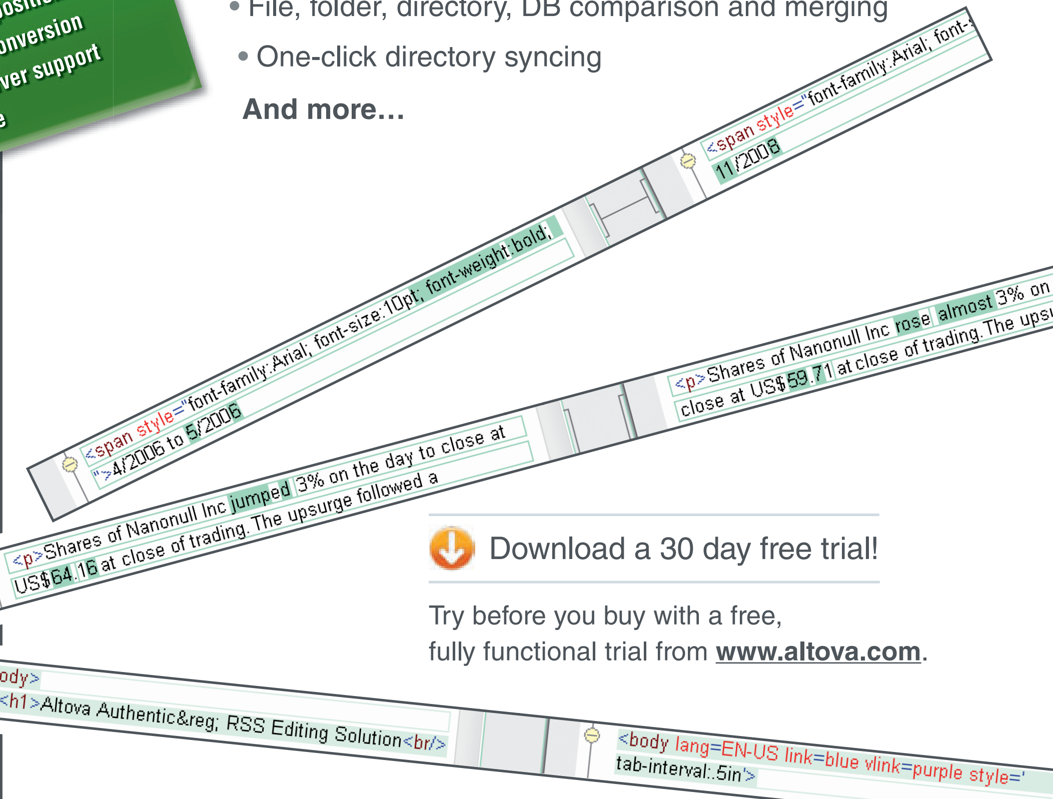
Many developers are used to a system of primary keys, foreign keys and constraints between the two. With Windows Azure Table storage, you have to let go of these concepts or you'll have difficulty grasping its system of keys.

In Windows Azure Tables, the string PartitionKey and RowKey properties work together as an index for your table, so when defining them, you must consider how your data is queried. Together, the properties also provide for uniqueness, acting as a primary key for the row. Each entity in a table must have a unique PartitionKey/RowKey combination.

Code download available at code.msdn.microsoft.com/mag201007DataPoints.




Summary of News Items



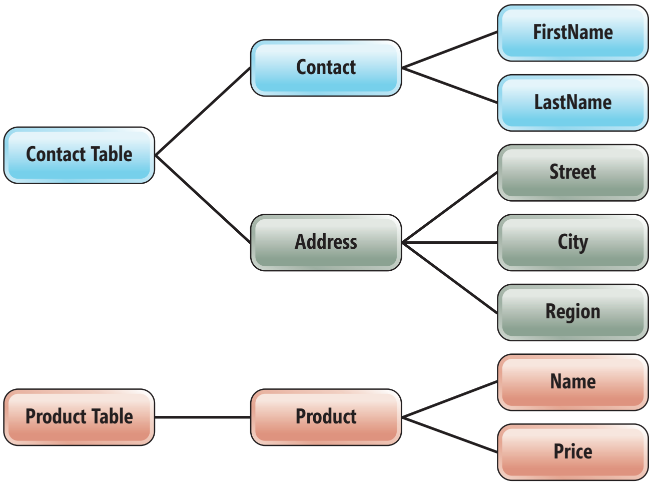


Figure 1 **A Single Windows Azure Table Can Contain Rows Representing Similar or Different Entities**

But you need to consider more than querying when defining a PartitionKey, because it's also used for physically partitioning the tables, which provides for load balancing and scalability. For example, consider a table that contains information about food and has PartitionKeys that correspond to the food types, such as Vegetable, Fruit and Grain. In the summer, the rows in the Vegetable partition might be very busy (becoming a so-called "hot" partition). The service can load balance the Food table by moving the Vegetable partition to a different server to better handle the many requests made to the partition.

If you anticipate more activity on that partition than a single server can handle, you should consider creating more-granular partitions such as Vegetable_Root and Vegetable_Squash. This is because the unit of granularity for load balancing is the PartitionKey. All the rows with the same PartitionKey value are kept together when load balancing. You could even design your table so that every single entity in the table has a different partition.

> Windows Azure Tables live in the cloud, but for me they began in a fog.

Digging Deeper into PartitionKeys and Querying

Notice that when I suggested fine-tuning the Vegetable PartitionKeys, I placed Vegetable at the beginning of the key, not the end. That's another mechanism for enabling more efficient queries. Queries to Windows Azure Tables from the Microsoft .NET Framework use LINQ to REST and a context that derives from the WCF Data Services System.Data.Services.Client.DataServiceContext. If you want to find any green squash, you can search in the Vegetable_Squash partition without wasting resources to search the entire table:

```
var query = _serviceContext.FoodTable.AsTableServiceQuery()
.Where(c => c.PartitionKey=="Vegetable_Squash"&& c.Color == "Green");
```

A big difference between querying OData (returned by WCF Data Services) and querying against Windows Azure Tables is that string functions are not supported. If you want to search part of a string, you must use String.CompareTo to inspect the beginning characters of the string. If you want to query the entire Vegetable category, however, you can use the CompareTo method to do a prefix search over the start of the PartitionKey:

```
var query = _serviceContext.FoodTable.AsTableServiceQuery()
            .Where(c => c.PartitionKey.CompareTo("Vegetable")>=0
            && c.PartitionKey.CompareTo("Vegetablf")<0
            && c.Color == "Green");
```

This would limit the search to only partitions that begin with Vegetable—nothing less and nothing more. (Using Vegetablf rather than Vegetable in the second predicate defines the upper bound, preventing foods in partitions such as Yogurt or VegetableLike from being returned.) In the code sample accompanying this article, you'll see how I've done this replacement dynamically.

Parallel Querying for Full Table Scans

What if you were searching for all green food, regardless of type? Windows Azure would have to scan through the entire table. If it's a large table, Windows Azure throws in another wrench: It can return only 1,000 rows at a time (or process for 5 seconds). Windows Azure will return those results along with a *continuation key*, then go back for more. This can be a tedious synchronous process.

Instead you could execute a number of queries, perhaps iterating through a known list of categories, then building each query:

```
_serviceContext.FoodTable.AsTableServiceQuery()
.Where(c => c.PartitionKey == _category && c.Color == "Green");
```

Then you can send off all the queries to run in parallel.

More Design Considerations for Querying

The RowKey property serves a number of purposes. In combination with PartitionKey, it can define uniqueness within a table for each row. For example, I know another Julie Lerman (truly I do). So the RowKey will be critical in differentiating us when we share a PartitionKey of lerman_julie. You can also use RowKey to help with sorting, because it acts as part of an index. So then, what would be useful RowKeys for Julie Lerman the elder (that's me) and Julie Lerman the younger? A GUID will certainly do the trick for identity, but it does nothing for searches or sorting. In this case, a combination of values would probably be best.

What else differentiates us? We live on opposite ends of the United States, but locations can change so that's not useful for a key. Certainly our date of birth is different (by more than 20 years) and that's a static value. But there's always the chance that another Julie Lerman with my birth date exists somewhere in the world and could land in my database—highly implausible, but not impossible. After all of the deliberation I might go through, birth date may still not be a value on which my application is searching or sorting. So in this case, RowKey might not be part of queries, and a plain-old GUID would suffice. You'll have to make these kinds of decisions for all of your Windows Azure Tables.

There's much more to learn about defining keys, and factors such as retrieving data, storing data, scalability and load balancing all come into play.

Rethinking Relationships

In a relational database, we rely on foreign keys and constraints to define relationships. We certainly could define a foreign key property in a class that refers to another class, but there's nothing in Windows Azure Table storage to enforce relationships. Your code will still be responsible for that.

This impacts how you perform queries and updates (including inserts and deletes) from tables.

When querying, you can't perform joins across tables. And when persisting data, you can't have transacted commands that span partitions or tables. There is, however, a mechanism for working with data in graphs, which is something I pointed out at the beginning of this column—you can store rows of varying schemas in a single table.

If your application requires that users work with contacts and addresses together, you could store the addresses in the same table as the contacts. It would be critical to ensure that the addresses have the same PartitionKey—for example, "lerman_julie." Also, the RowKey should contain a value that specifies the type or kind of entity, such as "address_12345," so you can easily differentiate between contact types and address types when querying.

The common PartitionKey ensures that the rows will always stay together to take advantage of a feature called Entity Group Transactions (EGT). This allows a single transaction to carry out operations atomically across multiple entities as long as all the entities have the same PartitionKey value. One of the benefits of EGT with respect to related data is that you can perform a transacted update on all the entities in a single transaction.

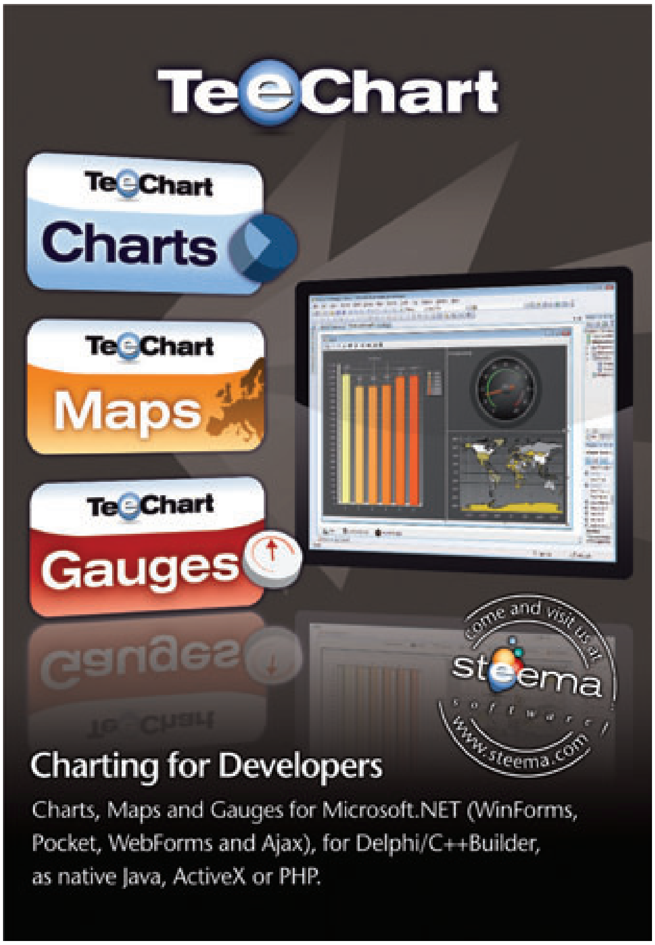
A Base of Understanding from Which to Learn More

Windows Azure Tables live in the cloud, but for me they began in a fog. I had a lot of trouble getting my head wrapped around them because of my preconceived understanding of relational databases. I did a lot of work (and pestered a lot of people) to enable myself to let go of the RDBMS anchors so I could embrace and truly appreciate the beauty of Windows Azure Tables. I hope my journey will make yours shorter.

There's so much more to learn about Windows Azure Table services. The team at Microsoft has some great guidance in place on MSDN. In addition to the PDC09 video mentioned earlier, check this resource page on the Windows Azure Storage team blog at blogs.msdn.com/windowsazurestorage/archive/2010/03/28/windows-azure-storage-resources. The team continues to add detailed, informative posts to the blog, and I know that in time, or even by the time this column is published, I'll find answers to my myriad questions. I'm looking forward to providing some concrete examples in a future Data Points column. ■

JULIE LERMAN *is a Microsoft MVP, .NET mentor and consultant who lives in the hills of Vermont. You can find her presenting on data access and other Microsoft .NET topics at user groups and conferences around the world. Lerman blogs at thedatafarm.com/blog and is the author of the highly acclaimed book, "Programming Entity Framework" (O'Reilly Media, 2009). Follow her on Twitter.com: julielerman.*

THANKS to the following technical experts for reviewing this article: *Brad Calder and Jai Haridas*



3 Solutions for Accessing SharePoint Data in Office 2010

Donovan Follette and Paul Stubbs

Millions of people use the Microsoft Office client applications in support of their daily work to communicate, scrub data, crunch numbers, author documents, deliver presentations and make business decisions. In ever-increasing numbers, many are interacting with Microsoft SharePoint as a portal for collaboration and as a platform for accessing shared data and services.

Some developers in the enterprise have not yet taken advantage of the opportunity to build custom functionality into Office applications—functionality that can provide a seamless, integrated experience for users to directly access SharePoint data from within familiar productivity applications. For enterprises looking at ways to improve end-user productivity, making SharePoint data available directly within Office applications is a significant option to consider.

With the release of SharePoint 2010, there are a number of new ways available to access SharePoint data and present it to the Office user. These range from virtually no-code solutions made possible via SharePoint Workspace 2010 (formerly known as Groove), direct synchronization between SharePoint and Outlook, the new SharePoint REST API and the new client object model. Just as in Microsoft Office SharePoint Server (MOSS) 2007, a broad array of Web services is available in SharePoint 2010 for use as well.

In this article, we'll describe a couple of no-code solutions and show you how to build a few more-complex solutions using these new features in SharePoint 2010.

This article discusses:

- Using external data sources
- Building a Word add-in
- Using the client object model
- Web services as social services

Technologies discussed:

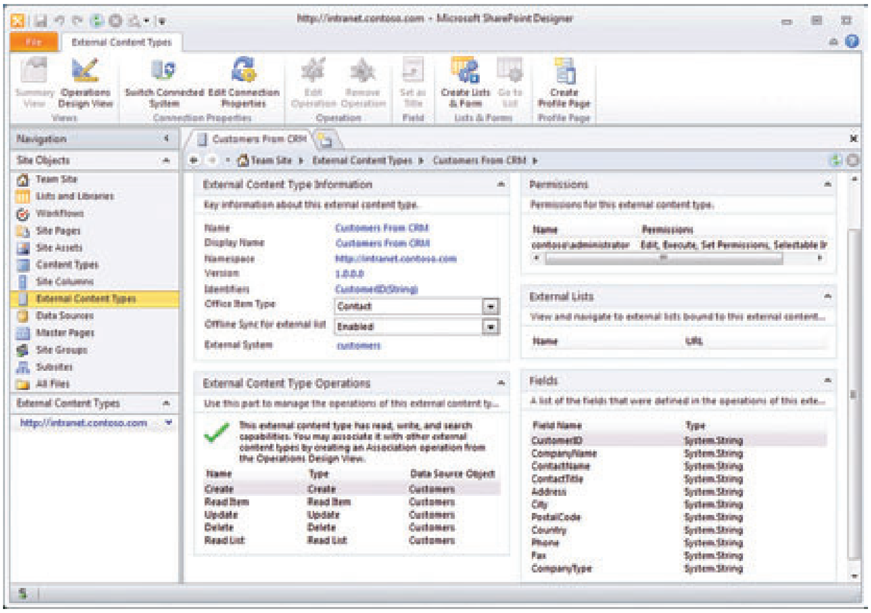
Office 2010, SharePoint 2010, Windows Communication Foundation

External Data Sources

Let's start by taking a quick look at the SharePoint list types you can employ as data sources.

One particularly useful data source is an external list that displays data retrieved via a connection to a line-of-business (LOB) system. MOSS 2007 let you to connect to LOB data using the Business Data Catalog (BDC), which provided read-only access to back-end systems. SharePoint 2010 provides Business Connectivity Services (BCS), which is an evolution of the BDC that supports full read/write access to your LOB data.

Why would you want to bring LOB data into SharePoint? Consider the use case where you have a customer relationship management (CRM) system that only a limited number of people in the organization can access directly. However, there's a customer table in the database with name and address data that could be used by many others if it were available. In real-life, you probably end up with users copying this information from various



External Content Type Information

Key information about this external content type.

Name	Customers From CRM
Display Name	Customers From CRM
Namespace	http://intranet.contoso.com
Version	1.0.0.0
Identifiers	CustomerID(String)
Office Item Type	Contact
Offline Sync for external list	Enabled
External System	customers

External Content Type Operations

Use this part to manage the operations of this external content ty...

✓ This external content type has read, write, and search capabilities. You may associate it with other external content types by creating an Association operation from the Operations Design View.

Name	Type	Data Source Object
Create	Create	Customers
Read Item	Read Item	Customers
Update	Update	Customers
Delete	Delete	Customers
Read List	Read List	Customers

Permissions

Permissions for this external content type.

Name	Permissions
contoso\administrator	Edit, Execute, Set Permissions, Selectable Ir

External Lists

View and navigate to external lists bound to this external content...

Name	URL

Fields

A list of the fields that were defined in the operations of this exte...

Field Name	Type
CustomerID	System.String
CompanyName	System.String
ContactName	System.String
ContactTitle	System.String
Address	System.String
City	System.String
PostalCode	System.String
Country	System.String
Phone	System.String
Fax	System.String
CompanyType	System.String

Figure 1 **ECT Configuration for Accessing External CRM Data**

non-authoritative sources and pasting it into their Office documents. It would be better to access this customer data from the authoritative CRM system and expose it in SharePoint as an external list that Office clients can access.

SharePoint Designer 2010 is the tool used for configuring access to a LOB system and making its data available in a SharePoint external list. There are a couple steps required to do this.

The first step is to create a new External Content Type (ECT). The ECT contains metadata describing the structure of the back-end data, such as the fields and CRUD methods that SharePoint will use to interact with it. Once the ECT has been created, an external list can be generated from it on any site within SharePoint. External lists look and act like any other standard list in SharePoint, but the external list data is not stored in SharePoint. Instead, it's retrieved via the ECT when accessed by an end user.

SharePoint Designer includes default support for connecting to external data sources including SQL Server, Windows Communication Foundation (WCF) and the Microsoft .NET Framework. Therefore, an ECT can be easily created for connecting to any SQL Server database table or view, WCF service or Web service. Custom .NET solutions can be built in Visual Studio 2010 using the new SharePoint 2010 Business Data Connectivity Model project template.

For the purposes of this article, the SQL Server data source type was used to create an ECT for a database table. Then the ECT was used to create an External List. **Figure 1** shows the resulting "Customers From CRM" ECT after completing the configuration in SharePoint Designer.

SharePoint Designer 2010 is the tool used for configuring access to a LOB system.

There are a couple things to call out here. First, notice in the External Content Type Information panel that the Office Item Type property value is set to Contact. During the configuration process, you can map the external data fields to a corresponding Office item type like Contact. This isn't a requirement, but because the name and address data from the CRM database can be mapped nicely to an Outlook Contact, this designation was chosen. You'll be able to use the result of this configuration option in Outlook later.

Second, notice in the External Content Type Operations panel that full CRUD methods have been enabled for this ECT. This was due to the selections made in the configuration wizard. However, there certainly may be business reasons to limit the LOB system operations to read-only. In that case, you can simply select the Read List and Read Item operations during configuration. These are the only two operations required to create an ECT.

Once the ECT is created, it's a simple step to create an external list from it. You can do this by creating a new external list from within SharePoint or SharePoint Designer.

SharePoint Standard Lists

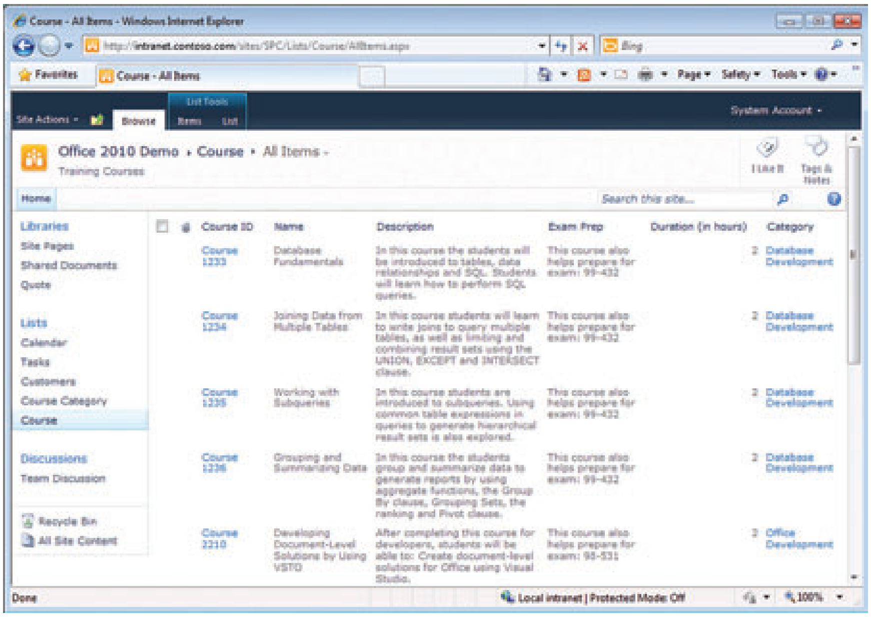
Of course, you can employ standard SharePoint lists to display business data. For example, say your department manages training-course content. You maintain two SharePoint lists: Course Category and Course. These lists contain the course information that employees on other teams use to create customer correspondence, brochures or advertising campaigns. So the data is maintained by a small team, but must be readily available for use by many people across the company.

SharePoint 2010 has a new capability whereby lookups form relationships between lists. When creating a new column on a list, one of the options is to make the column a lookup type, then indicate another list within the site as its source. SharePoint supports single-value lookups for one-to-many relationships or multi-value lookups for many-to-many relationships. If you choose, SharePoint will also maintain referential integrity between the lists supporting restricted or cascading deletes. This provides a number of options in how you set up and use lists in SharePoint.

Going back to our example, you could easily create a Course list lookup column named Category that's sourced from the Course Category list as shown in **Figure 2**.

Bringing SharePoint List Data to Office

So far, we've looked at how to surface external data as SharePoint lists using the new BCS features in SharePoint 2010. Users can access the data via the browser on a PC or a mobile device, but users will probably appreciate the rich experience of the full Office



Course ID	Name	Description	Exam Prep	Duration (in hours)	Category
Course 1233	Database Fundamentals	In this course the students will be introduced to tables, data relationships and SQL. Students will learn how to perform SQL queries.	This course also helps prepare for exam: 99-432	2	Database Development
Course 1234	Joining Data from Multiple Tables	In this course students will learn to write joins to query multiple tables, as well as limiting and combining result sets using the UNION, EXCEPT and INTERSECT clause.	This course also helps prepare for exam: 99-432	2	Database Development
Course 1235	Working with Subqueries	In this course students are introduced to subqueries. Using common table expressions in queries to generate hierarchical result sets is also explored.	This course also helps prepare for exam: 99-432	2	Database Development
Course 1236	Grouping and Summarizing Data	In this course the students group and summarize data to generate reports by using aggregate functions, the Group By clause, Grouping Sets, the ranking and Pivot clause.	This course also helps prepare for exam: 99-432	2	Database Development
Course 2210	Developing Document-Level Solutions by Using VSTO	After completing this course for developers, students will be able to: Create document-level solutions for Office using Visual Studio.	This course also helps prepare for exam: 98-531	3	Office Development

Figure 2 **Using a Lookup List to Source Course Category Data**

client application. Let's now turn our attention to using the SharePoint list data on the client in two ways. First, we'll see how you can access data without writing any code by employing SharePoint Workspace and Outlook.

When developing our sample CRM solution, there are two Connect & Export buttons in the SharePoint ribbon for the external customers list: Sync to SharePoint Workspace and Connect to Outlook (see **Figure 3**). If SharePoint Workspace 2010 is installed on the client computer, Sync to SharePoint Workspace lets you synchronize lists and document libraries to the client with a single click. A local cached copy of the content is then available to the user in SharePoint Workspace whether the user is online or offline. When the user is in an offline state and modifies a list item or document and saves it locally, the list item or document will be synchronized with SharePoint automatically when the user is back online again.

You can employ standard SharePoint lists to display business data.

This is a no-code-required solution. Data is made accessible in the SharePoint Workspace client application shown in **Figure 4**. And because full CRUD methods were defined in the ECT, any changes made to the customer data in SharePoint Workspace will be updated in the CRM database as well.

Because we mapped the CRM database fields to the Contact Office item type during ECT configuration, SharePoint can provide our external list data to Outlook as native Contact Items. By clicking the Connect to Outlook button on the ribbon, SharePoint will synchronize this external list directly to Outlook. Again, no code required, with SharePoint data landing in the Office client.

Using the REST API

No-code solutions, such as those enabled through SharePoint Workspaces and Outlook list connectivity, are great, but there are some user experiences that require a more-customized solution. To accommodate these, we need to provide access to the list data in the Office applications in a way that permits us to further tailor the solution.

Possibly one of the easiest ways for a developer to access SharePoint list and document library data is via the new REST API (listdata.svc). Most of the data in SharePoint is exposed as a RESTful endpoint. The standard location for SharePoint services is _vti_bin, so if you simply type into your browser the URL to your site and append /_vti_bin.listdata.svc, you will get back a standard ATOM services document that describes the collections available on the site (see **Figure 5**).

Notice that the Course and CourseCategory lists are present. By further appending /Course to the URL, you can retrieve all the courses in the list or you can retrieve any one specific course by appending a number. For example, this will return the third course:

```
http://intranet.contoso.com/sites/SPC/_vti_bin/listdata.svc/Course(3)
```

You can do more advanced queries by appending the following property filter:

```
?$filter=startswith(propertyname,'value')
```

But an advanced query that's important here is one that can return the Courses with their associated CourseCategory data. By appending the following to the site URL, you can retrieve the combined structure of Course and CourseCategory in a single payload:

```
/_vti_bin.listdata.svc/Course?$expand=Category
```

You'll see this implemented in a Word add-in in the next section.

Building a Word Add-In

Once you know how to leverage the REST APIs to acquire access to the data, you can surface the data in the client applications where users have a rich authoring experience. For this example, we'll build a Word add-in and present this data to the user in a meaningful way. This application will have a dropdown list for the course categories, a listbox that loads with courses corresponding to the category selection and a button to insert text about the course into the Word document.

In Visual Studio 2010, create a new Office 2010 Word add-in project in C#.

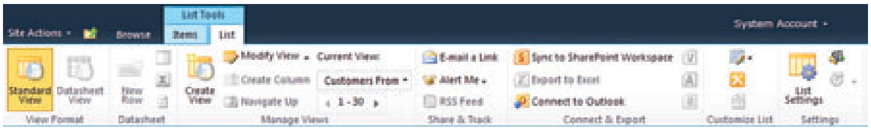


Figure 3 **Connect & Export Options in the SharePoint Ribbon**

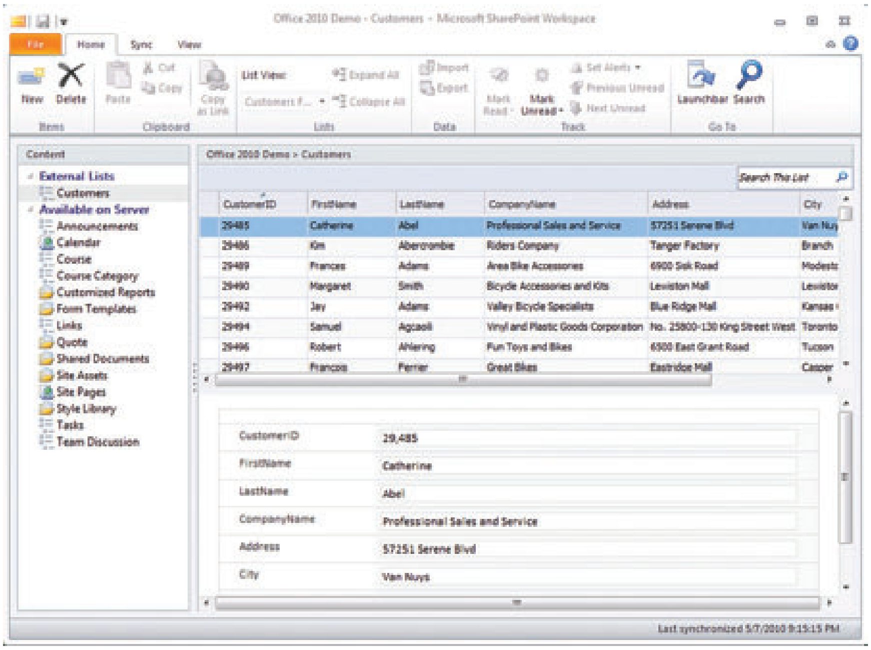
GET THE FASTEST CONTROLS...

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

Infragistics®

Killer Apps. No Excuses.

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



Office 2010 Demo - Customers - Microsoft SharePoint Workspace

Content
- External Lists
  - Customers
- Available on Server
  - Announcements
  - Calendar
  - Course
  - Course Category
  - Customized Reports
  - Form Templates
  - Links
  - Quote
  - Shared Documents
  - Site Assets
  - Site Pages
  - Style Library
  - Tasks
  - Team Discussion

Office 2010 Demo > Customers

CustomerID	FirstName	LastName	CompanyName	Address	City
29485	Catherine	Abel	Professional Sales and Service	57251 Serene Blvd	Van Nuy
29486	Kim	Abercrombie	Riders Company	Tanger Factory	Branch
29489	Frances	Adams	Area Bike Accessories	6900 Sisk Road	Modest
29490	Margaret	Smith	Bicycle Accessories and Kits	Lewiston Mall	Lewistor
29492	Jay	Adams	Valley Bicycle Specialists	Blue Ridge Mall	Kansas
29494	Samuel	Agcaoili	Vinyl and Plastic Goods Corporation	No. 25800-130 King Street West	Toronto
29496	Robert	Ahlering	Fun Toys and Bikes	6500 East Grant Road	Tucson
29497	Francois	Ferrier	Great Bikes	Eastridge Mall	Casper

CustomerID	29,485
FirstName	Catherine
LastName	Abel
CompanyName	Professional Sales and Service
Address	57251 Serene Blvd
City	Van Nuys

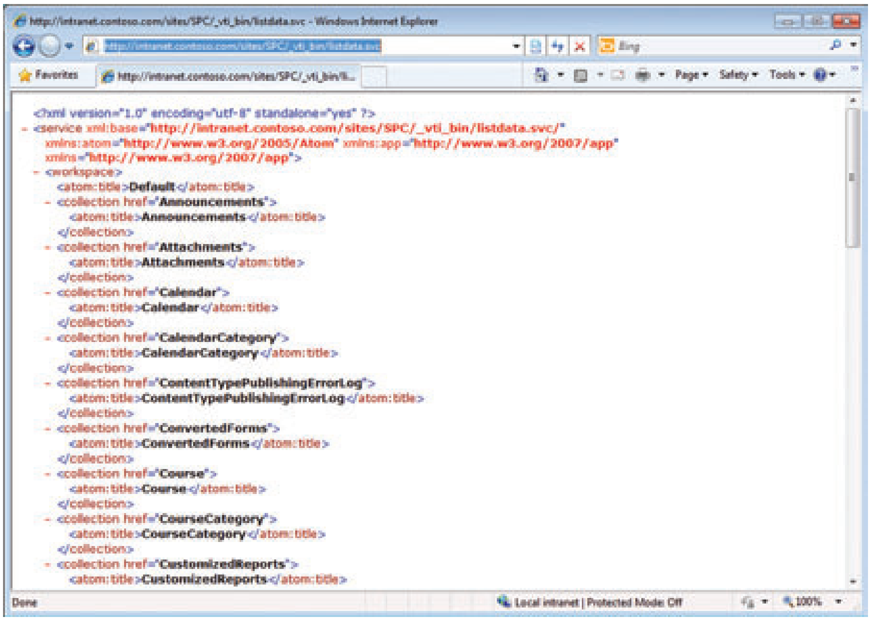
Last synchronized 5/7/2010 8:15:15 PM

Figure 4 **Accessing External List Data in a SharePoint Workspace**

Now add a new service data source. On the Add Service Reference panel in the wizard, enter the URL for your SharePoint site and append /_vti_bin/listdata.svc to it. For example:

```
http://intranet.contoso.com/_vti_bin/listdata.svc
```

After entering the URL, click Go. This retrieves the metadata for the SharePoint site. When you click OK, WCF Data Services will generate strongly typed classes for you by using the Entity Framework. This completely abstracts away the fact that the data source is SharePoint or an OData producer that provides data via the Open Data Protocol. From this point forward, you simply work with the data as familiar .NET classes.



http://intranet.contoso.com/sites/SPC/_vti_bin/listdata.svc - Windows Internet Explorer

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
- <service xml:base="http://intranet.contoso.com/sites/SPC/_vti_bin/listdata.svc/"
    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="Announcements">
        <atom:title>Announcements</atom:title>
      </collection>
    - <collection href="Attachments">
        <atom:title>Attachments</atom:title>
      </collection>
    - <collection href="Calendar">
        <atom:title>Calendar</atom:title>
      </collection>
    - <collection href="CalendarCategory">
        <atom:title>CalendarCategory</atom:title>
      </collection>
    - <collection href="ContentTypePublishingErrorLog">
        <atom:title>ContentTypePublishingErrorLog</atom:title>
      </collection>
    - <collection href="ConvertedForms">
        <atom:title>ConvertedForms</atom:title>
      </collection>
    - <collection href="Course">
        <atom:title>Course</atom:title>
      </collection>
    - <collection href="CourseCategory">
        <atom:title>CourseCategory</atom:title>
      </collection>
    - <collection href="CustomizedReports">
        <atom:title>CustomizedReports</atom:title>
```

Done | Local intranet | Protected Mode: Off

Figure 5 **ATOM Services Document**

For the UI, you will create a custom task pane, which provides a UI in Office applications that can be docked on the top, bottom, left or right of the application. Task panes can have Windows Forms controls added to them, including the Windows Presentation Foundation (WPF) user control that will be used here.

Add a WPF user control to the project using the Add New Item dialog and name it CoursePicker. When the designer opens, replace the Grid element with the XAML snippet shown in **Figure 6**. This simply adds the ComboBox, Button and ListBox and sets some properties. You will add a couple events later.

Open the CoursePicker.xaml.cs file. Immediately following the namespace, you'll add two using statements, one for your service reference, ServiceReference1, and one for System.Net:

```
namespace Conf_DS {
  using ServiceReference1;
  using System.Net;
```

In the CoursePicker Class, the first order of business is to instantiate the data context object. Here, you pass in the URL to your site, again appended by the _vti_bin/listdata.svc designation:

```
public partial class CoursePicker : UserControl {
  Office2010DemoDataContext dc = new Office2010DemoDataContext(
    new Uri("http://intranet.contoso.com/sites/spc/_vti_bin/listdata.svc"));
```

Next you'll have a List class-level variable to cache the retrieved course items and save round-trips to the server:

```
List<CourseItem> courses = null;
```

The code to retrieve the Courses and CourseCategory data is in the OnInitialized override method. First, you designate your logged-in credentials to pass to the server. Then the course categories are retrieved via the data context object and bound to the category ComboBox. Finally, using the expand option, courses are returned with their associated category and loaded into the courses list object. This will cache the courses locally for better performance:

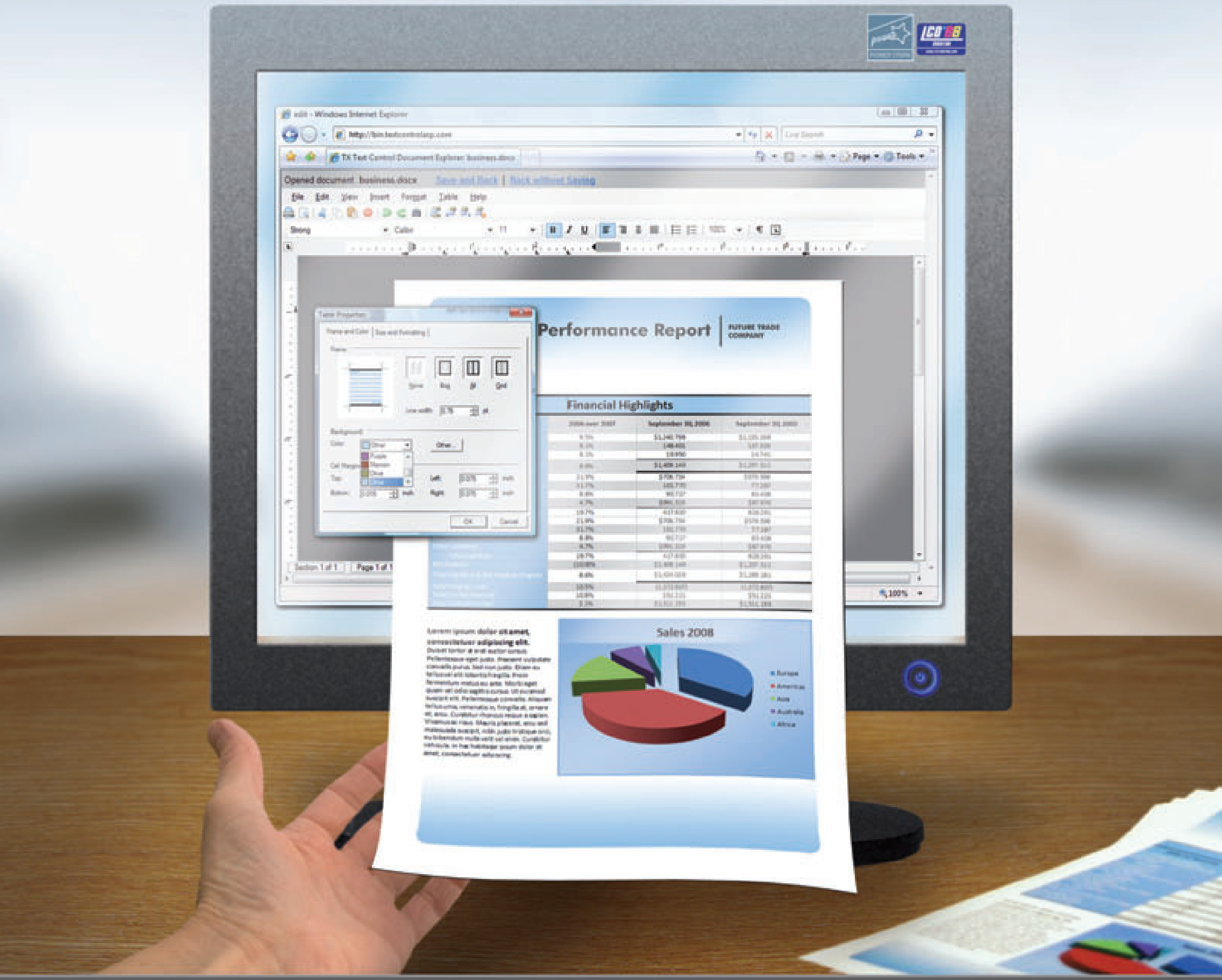
```
protected override void OnInitialized(EventArgs e) {
  dc.Credentials = CredentialCache.
DefaultCredentials;

  // Load Category dropdown list
  cboCategoryLookup.DataContext =
    dc.CourseCategory;
  cboCategoryLookup.SelectedIndex = 0;

  // To cache data locally for courses
  // Expand to retrieve the Category as well.
  courses = dc.Course.Expand("Category").ToList();

  base.OnInitialized(e);
}
```

Now you need to add a couple events. Return to the CoursePicker designer and double-click the button to create the button click event. Next, click on the ComboBox and in the properties menu, click the Events tab and double-click the SelectionChanged

WINDOWS FORMS / WPF / ASP.NET / ACTIVEX

WORD PROCESSING COMPONENTS

(WHAT YOU SEE IS WHAT YOU GET)



TX TEXTCONTROL®
word processing components

Microsoft® Visual Studio®
PARTNER

Word Processing Components
for Windows Forms & ASP.NET

www.textcontrol.com

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

event. Add code to your SelectionChanged event handler so it looks like this:

```
private void cboCategoryLookup_SelectionChanged(
  object sender, SelectionChangedEventArgs e) {

  courseListBox.DataContext =
    from c in courses
    where c.Category.CategoryName ==
      cboCategoryLookup.SelectedValue.ToString()
    orderby c.CourseID
    select c;
}
```

Here, a simple LINQ query searches the courses list object (the one loaded with data retrieved using the expand option) to find all the courses that have a category name that matches the name of the course category selected in the ComboBox. It also orders the results to provide a clean user experience.

Finally, add code to the button event handler to cast the selected listbox item into a CourseItem object. Then you take the various data elements you want to present to the user and place them in the document at the location of the insertion point:

```
private void button1_Click(
  object sender, RoutedEventArgs e) {

  CourseItem course =
    (CourseItem)courseListBox.SelectedItem;
  Globals.ThisAddIn.Application.Selection.InsertAfter(
    String.Format("{0}: {1} \n{2}\n", course.CourseID,
    course.Name, course.Description));
}
```

And that's it—really simple code for accessing the data in SharePoint via WCF Data Services.

Now open the ThisAddIn.cs file. This is the main entry point for all add-ins for Office. Here you add the code to instantiate the task pane:

```
private void ThisAddIn_Startup(
  object sender, System.EventArgs e) {

  UserControl wpfHost = new UserControl();
  ElementHost host = new ElementHost();
  host.Dock = DockStyle.Fill;
  host.Child = new CoursePicker();
  wpfHost.Controls.Add(host);
  CustomTaskPanes.Add(
    wpfHost, "Training Courses").Visible = true;
}
```

One of the easiest ways to access SharePoint list and document library data is via the new REST API.

The CoursePicker WPF user control can't be directly added to the custom task pane objects collection. It must be hosted in an ElementHost control, which provides the bridge between WPF controls and Windows Forms controls. Notice that the CoursePicker object is added as a child of the ElementHost object and then the ElementHost object is added to the custom task pane object collection. An Office application can have more than one custom task pane installed and available to the user at any given time, so the task pane for this add-in will just be one in the collection. **Figure 7** shows the completed add-in.

Figure 6 **Word Add-In UI Markup**

```
<StackPanel>
  <ComboBox
    Name="cboCategoryLookup" Width="180" Margin="5"
    HorizontalAlignment="Center" IsEditable="False"
    ItemsSource="{Binding}"
    DisplayMemberPath="CategoryName"
    SelectedValuePath="CategoryName" />
  <Button Name="button1"
    Content="Insert Course Information" Margin="5" />
  <ListBox Name="courseListBox" ItemsSource="{Binding}">
    <ListBox.ItemTemplate>
      <DataTemplate>
        <StackPanel>
          <StackPanel Orientation="Horizontal">
            <TextBlock Text="{Binding Path=CourseID}"
              FontWeight="Bold" />
            <TextBlock Text=": " FontWeight="Bold" />
            <TextBlock Text="{Binding Path=Name}" />
          </StackPanel>
          <TextBlock Text="{Binding Path=Description}"
            Margin="5 0 0 0" />
        </StackPanel>
      </DataTemplate>
    </ListBox.ItemTemplate>
  </ListBox>
</StackPanel>
```

With the data appearing in the Office application, you can take the solution further by adding code that interacts with the Word APIs. For example, you can add code so that when a user selects a course, the information is inserted and formatted in the document. The Office application APIs are rich and allow you to add more features to your custom solution that can make users even more productive. Next, we'll see an example of this with Word content controls connected to a client-side SharePoint object model.

Using the Client Object Model

Using the REST APIs to gain access to the data is one among a few options available to you. For example, there are also three new APIs available for SharePoint 2010 that provide a consistent programming model across the JavaScript, .NET managed applications and Silverlight clients. These three client object models interact with SharePoint using a subset of the server object model capabilities and essentially interoperate with SharePoint at the site collection level and below: webs, lists, listitems, content types, fields and external lists. If you're familiar with the server object model, you'll be familiar with the client object model.

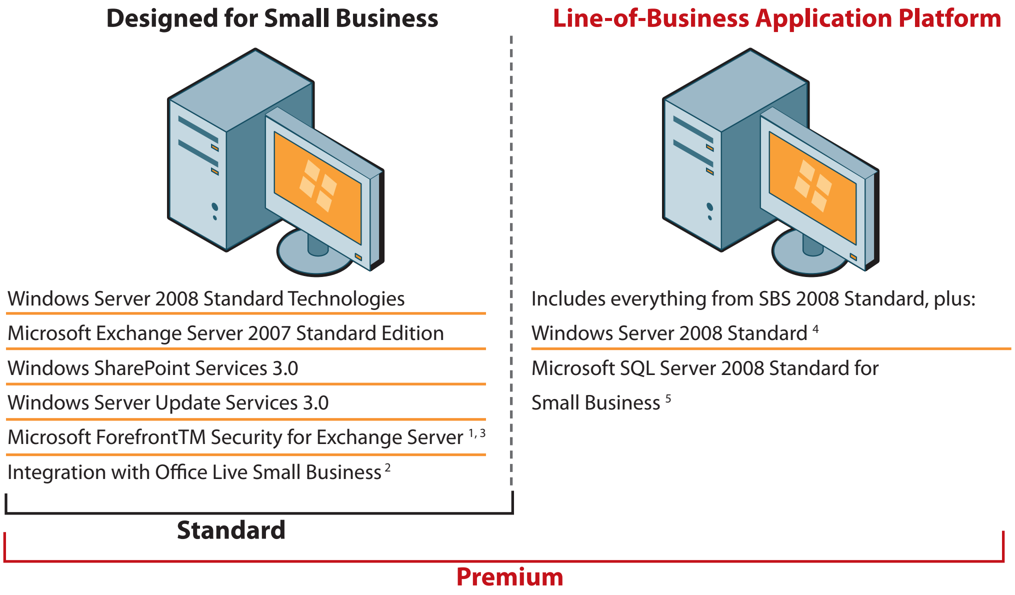
To demonstrate using the client object model, we'll use the external list containing the CRM customers to build a document-level Word add-in where the action pane is loaded with the customers. This is a case where you'll need to use the client object model because the List Data Service doesn't provide access to external lists. In this example, the user can select a customer and insert his name and address information into content controls in a quote document template.

The previous Course and Category example was an application-level add-in. An application-level Word add-in will be present every time Word is started. Document-level add-ins, however, are bound to a document and will only load if a document of a certain type is

Windows Small Business Server 2008

Enhance productivity, access your business desktop virtually anytime, anywhere.

Windows Small Business Server 2008 delivers a range of features and capabilities for small businesses. Business owners and employees will benefit from built-in antivirus and anti-spam protection, integration with Microsoft®
Office Live Small Business and Windows SharePoint® Services 3.0, and support for Windows Mobile® devices. IT managers and technology consultants will appreciate more flexible and costeffective licensing, a more secure infrastructure, and being able to run Microsoft SQL Server® 2008 Standard Edition for Small Business and Windows Server 2008 Standard technologies on a second hardware server (with SBS Premium Edition).



Designed for Small Business

- Windows Server 2008 Standard Technologies
- Microsoft Exchange Server 2007 Standard Edition
- Windows SharePoint Services 3.0
- Windows Server Update Services 3.0
- Microsoft ForefrontTM Security for Exchange Server [1, 3]
- Integration with Office Live Small Business [2]

Standard

Line-of-Business Application Platform

Includes everything from SBS 2008 Standard, plus:

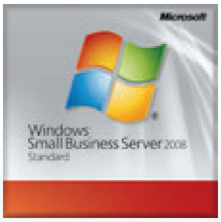
- Windows Server 2008 Standard [4]
- Microsoft SQL Server 2008 Standard for Small Business [5]

Premium

Save up to $100 on Windows Server Solutions!

Intel® SR1630HGP 1U Barebone Server
Microsoft Windows Server
Standard 2008 32Bit/x64

Your Price: $1079.98
Combo Discount: $100



Intel® SR1630HGP 1U Barebone Server
Microsoft Windows Small Business Server
2008 SP2 Standard Edition

Your Price: $1159.98
Combo Discount: $100

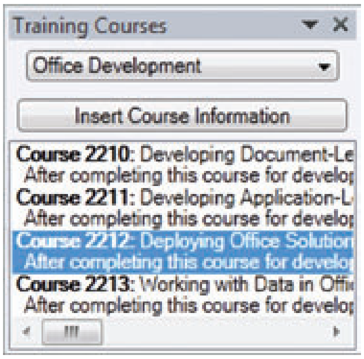
Go to:
www.neweggbusiness.com/msserver

ONCE YOU KNOW, YOU NEWEGG.®

neweggbusiness℠

Follow us on:

opened. In this case, the external customers list will only be presented to the user when working on a quote document.



Course 2210: Developing Document-Le
After completing this course for develo
Course 2211: Developing Application-L
After completing this course for develo
Course 2212: Deploying Office Solution
After completing this course for develo
Course 2213: Working with Data in Offi
After completing this course for develo

Figure 7 **The Word Add-In at Work**

In Visual Studio, start by creating a new Word 2010 document project. In the wizard, you'll need to select either a default document or a document that you've already saved. In my case, I used a quote document I had already saved. The document opens inside Visual Studio and Word becomes the document designer.

You can use the toolbox to place controls directly on the document surface as you would a Windows Forms application. Here you add Word content controls for the name and address information. These content controls will be populated with data from the user's customer selection at run time.

To add a content control to the document, select the text on the document that you want to wrap in the content control. Then drag a RichTextContentControl from the Word Controls in the toolbox and drop it on the selected text. Then provide a Name for the control and a Text value in Properties. Do this for customer and company name, address, city and customer ID so your document looks like **Figure 8**.

Because the client object model does not provide strongly typed data from the server, you need to add a Customer class to the project. The Customer class will be used to map data returned from the client object model:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace CSOM_Quote {
  public class Customer    {
    public string CustomerID { get; set; }
    public string CompanyName { get; set; }
    public string ContactName { get; set; }
    public string Address { get; set; }
    public string City { get; set; }
  }
}
```

To use the client object model you need to reference Microsoft.SharePoint.Client and Microsoft.SharePoint.Client.Runtime.

As with the previous example, data retrieval takes place in the OnIntitialized override method. There are a couple of major differences between coding against the client object model and WCF Data Services. First, the client object model expects that you have familiarity with SharePoint and its structure. With WCF Data Services, that's abstracted away and you work with the data only. Second, with the client object model, the returned data is not strongly typed. You're responsible for getting the data into objects that you can use for LINQ queries and data binding.

The data access code is shown in **Figure 9**. The client context is the central object here. Pass the site URL to create a new instance of the Client Context. Then you can start creating SharePoint objects using the following steps:

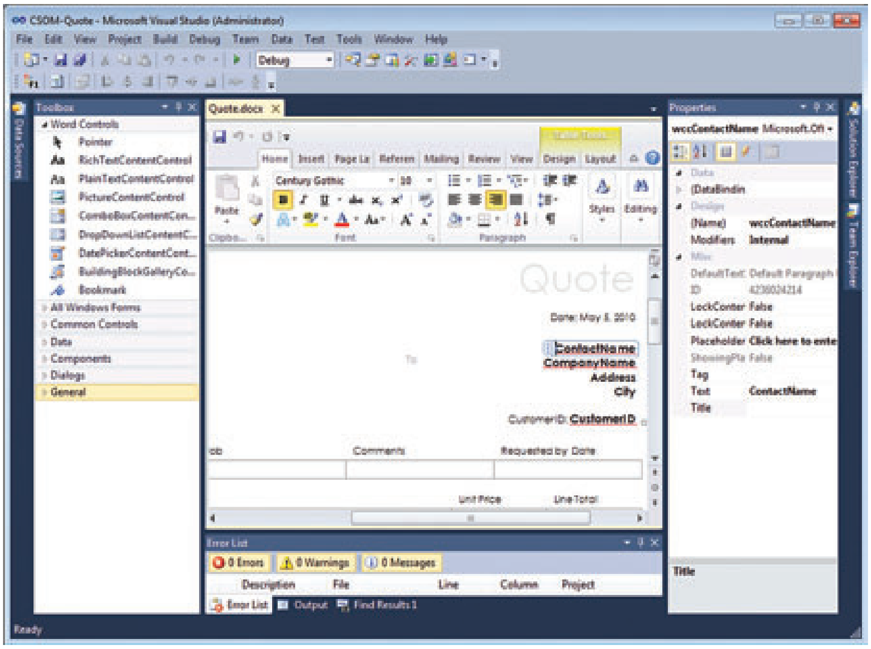
1. Create a site
2. Create a collection of site lists
3. Get a list with a specific name
4. Get all the items in the list
5. Load the query into the context
6. Execute the query

Before calling the ExecuteQuery method, all the previous statements are queued and then only sent to the server when execute query is called. This way, you're in control of the bandwidth and payloads. Once the query returns with its results, the remaining code maps the data into a customers list object that can be bound to the customer listbox control.

A WPF user control is used for this example as well. Because the XAML is similar to the previous example, it isn't shown here. However, the code to instantiate a document-level action pane rather than an application-level task pane is a bit different, as you can see here:

```
public partial class ThisDocument {
  private CustomersCRM CustomerActionPane =
    new CustomersCRM();

  private void ThisDocument_Startup(
    object sender, System.EventArgs e) {

    ElementHost host = new ElementHost();
    host.Dock = DockStyle.Fill;
    host.Child = new CustomerPicker();
    CustomerActionPane.Controls.Add(host);
    this.ActionsPane.Controls.Add(CustomerActionPane);
  }
...
```

Notice that the customer picker WPF user control is added to the ElementHost, the ElementHost object is added to the customer action pane controls collection, and then the customer action pane is added to the actions pane controls collection.



(Name) wccContactName
Modifiers Internal
DefaultText Default Paragraph
ID 4236024214
LockConter False
LockConter False
Placeholder Click here to ente
ShowingPla False
Tag
Text ContactName
Title

Figure 8 **Creating the Quote Document**

Now v4.0!

Why is Amyuni PDF so interesting?

Proven

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

High-Performance

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

Rapid Integration

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

Expertise

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

OEM Licenses

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

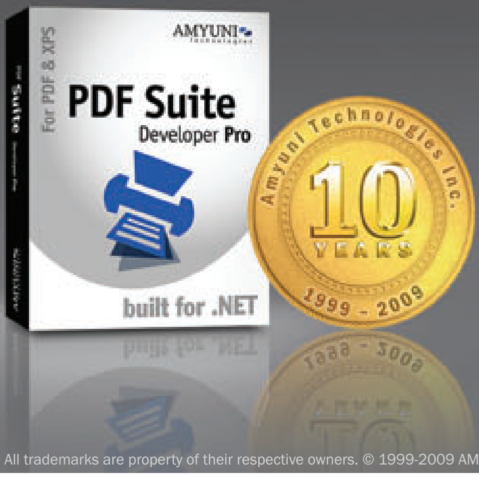
Customization

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

SD Times 2009 100

Windows 7

Certified for Windows Vista



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

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

www.amyuni.com

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

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

AMYUNI Technologies

Figure 9 **CRM Add-In Data Access Code**

```
protected override void OnInitialized(EventArgs e) {
  SPClientOM.ClientContext context =
    new ClientContext("http://intranet.contoso.com/sites/spc");
  SPClientOM.Web site = context.Web;
  SPClientOM.ListCollection lists = site.Lists;
  var theBCSList = lists.GetByTitle("Customers");
  SPClientOM.CamlQuery cq = new SPClientOM.CamlQuery();
  IQueryable<SPClientOM.ListItem> bcsListItems =
    theBCSList.GetItems(cq);
  bcsList = context.LoadQuery(bcsListItems);
  context.ExecuteQuery();

  var bcsCustomerData =
    from cust in bcsList
    select new Customer {
      CustomerID = cust.FieldValues.ElementAt(1).Value.ToString(),
      ContactName = cust.FieldValues.ElementAt(2).Value.ToString()
        + “ “
        + cust.FieldValues.ElementAt(3).Value.ToString(),
      CompanyName = cust.FieldValues.ElementAt(4).Value.ToString(),

      Address = cust.FieldValues.ElementAt(5).Value.ToString(),
      City = cust.FieldValues.ElementAt(6).Value.ToString(),  };

  foreach (var x in bcsCustomerData)  {
    Customer tempCustomer = new Customer();
    tempCustomer.CustomerID = x.CustomerID;
    tempCustomer.CompanyName = x.CompanyName;
    tempCustomer.ContactName = x.ContactName;
    tempCustomer.Address = x.Address;
    tempCustomer.City = x.City;

    customers.Add(tempCustomer);
  }

  customerListBox.DataContext = customers;
  base.OnInitialized(e);
}
```

The last step is to add the button click event to populate the Word content controls with the appropriate name and address information, as shown in **Figure 10**.

First, you cast the selected listbox item to a customer object. Then data from the customer object is used to populate the content controls. The results will look like **Figure 11**.

Web Services as Social Services

So far you've seen a number of ways you can access SharePoint data from Office client applications. The final technique we'll look at is using Web services. SharePoint offers Web services as the primary way to access SharePoint data remotely. Web services in SharePoint 2010 gives you access to nearly all of the functionality in SharePoint Server. Unlike some of the other data technologies you've seen, such as REST and the client object model, Web services covers both accessing data and accessing administrative functionality.

All of the Web services you love are still in there, such as the Lists.asmx and Search.asmx services. SharePoint Web services are implemented as ASP.NET Web services with the .asmx extension, and most of the new services in SharePoint 2010 are also written as ASMX services. This was mainly done to have the broadest compatibility with other products and tools.

A new focus of SharePoint Web services is social services. The center of all social applications is the user. SharePoint has a UserProfileService that allows you to access all of the profile information about a user. UserProfileService includes the standard properties such as name and address, but also includes other properties such as hobbies, skills, schools and colleagues. Colleagues (or friends as they're called in other public social sites) are a key feature in the SharePoint social structure.

Another important aspect of social applications is what people think about content they encounter. SharePoint has a SocialDataService that enables users to tag, rate and comment on data, documents and pages within your sites.

The third important social aspect of SharePoint is publishing activities and subscribing to activities that your colleagues generate. SharePoint provides an ActivityFeed and APIs to publish activities as a feed.

> Take the solution further by adding code that interacts with the Word APIs.

Because this isn't an article on the new social features in SharePoint, we won't go into more detail on these, but they do provide some important context for the examples later in this article. See the SharePoint Developer Center (msdn.microsoft.com/sharepoint) or the “Managing Social Networking with Microsoft Office SharePoint Server 2007” white paper (technet.microsoft.com/library/cc262436(office.12)) for more details.

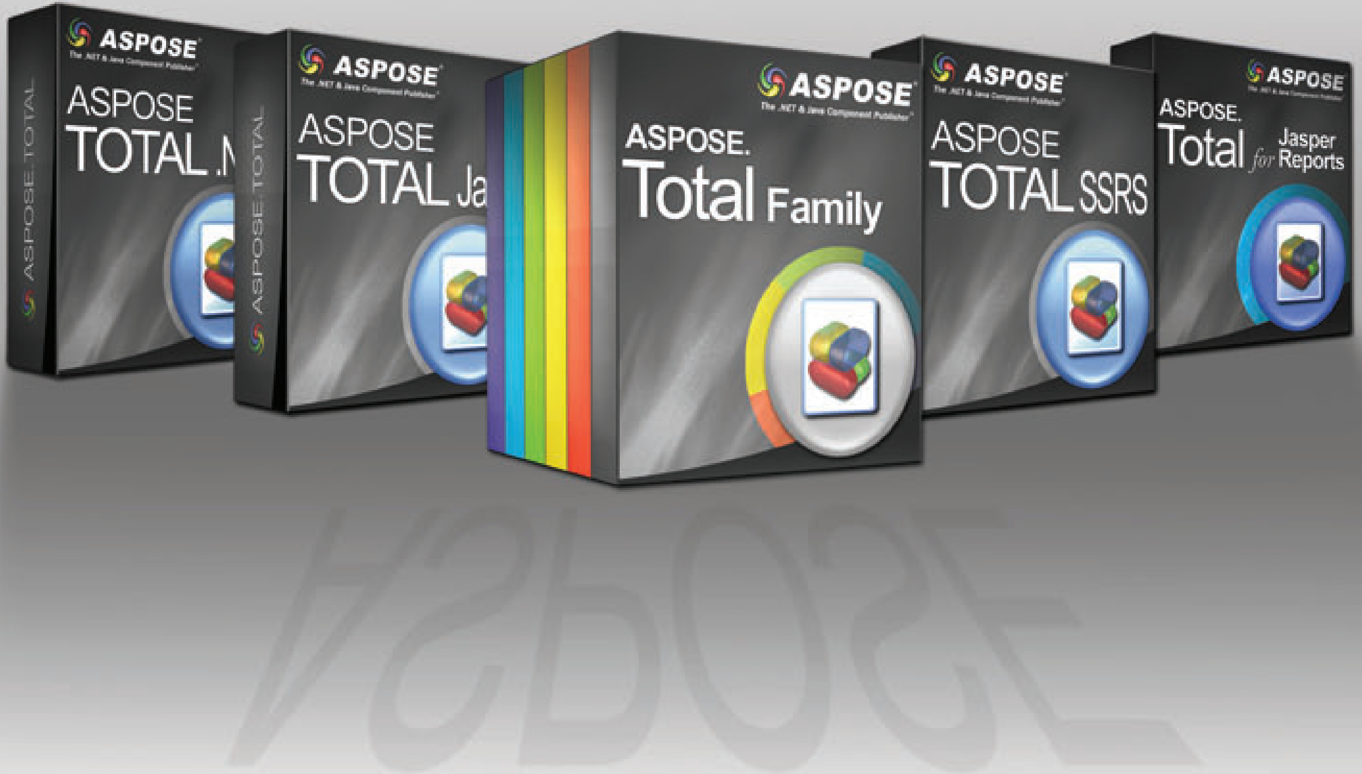
Extending Outlook with Web Services

We've seen how SharePoint and Office provide a lot of choices when you're determining the best way to access data for use in Office applications. Another way includes consuming SharePoint Web services. In this example, we'll create a new Outlook Ribbon that lets you pull all of your SharePoint colleagues into Outlook as contact items. You're even able to surface the user's profile picture into Outlook, just as you're accustomed to seeing with contacts provided by Microsoft Exchange.

Start by creating a new Outlook add-in in Visual Studio 2010. We're going to write it in C#, but you could use Visual Basic if you prefer. In previous versions, Visual Basic had a slight advantage with support for features such as optional parameters, but C# now supports them, too.

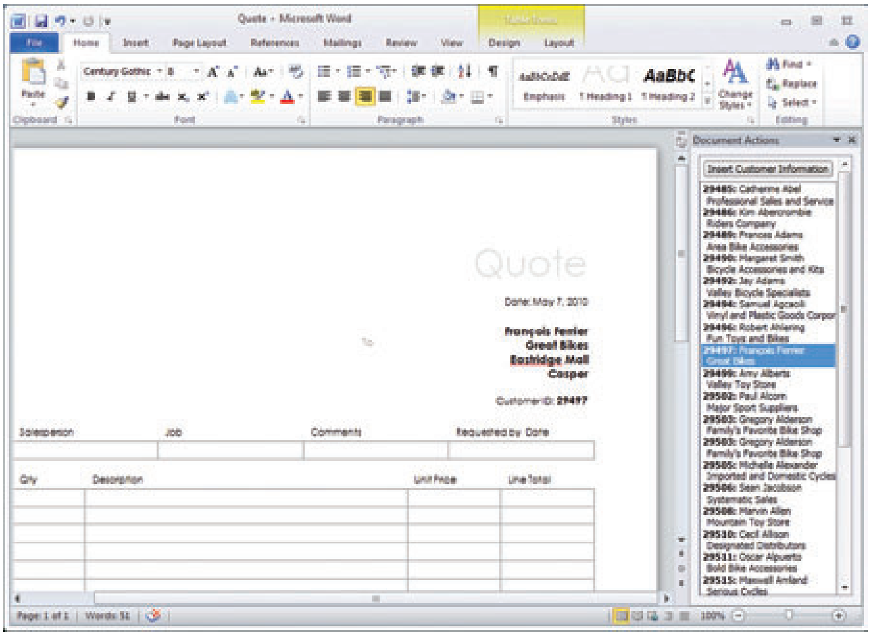
Figure 10 **Adding the Button Click Event to Word Content Controls**

```
private void button1_Click(
  object sender, RoutedEventArgs e) {
  Customer customer =
    (Customer)customerListBox.SelectedItem;

  Globals.ThisDocument.wccContactName.Text =
    customer.ContactName;
  Globals.ThisDocument.wccCompanyName.Text =
    customer.CompanyName;
  Globals.ThisDocument.wccAddress.Text =
    customer.Address;
  Globals.ThisDocument.wccCity.Text =
    customer.City;
  Globals.ThisDocument.wccCustomerID.Text =
    customer.CustomerID;
}
```


ASPOSE
The .NET & Java Component Publisher
ASPOSE TOTAL .N
ASPOSE.TOTAL
ASPOSE TOTAL Ja
ASPOSE. Total Family
ASPOSE TOTAL SSRS
ASPOSE. Total for Jasper Reports

The Ribbon provides a consistent and easy way to interact with all of the Office applications. Outlook 2010 now includes a Ribbon for the main screen. In this example, you'll add a new Ribbon here. Visual Studio 2010 makes it easy to create Office Ribbons with a visual Ribbon Designer. You can simply drag controls from the toolbox on the left and drop them onto the design surface.

In this example, you just need to set some properties, such as the label for the tab and group. Next add a button control onto the surface. Once you have a button added to your Ribbon group, you can set the size to large and set an image for the button. Your Ribbon will look similar to **Figure 12**.



Quote

Date: May 7, 2010

François Ferrier
Great Bikes
Eastridge Mall
Casper

CustomerID: 29497

Salesperson	Job	Comments	Requested by Date

Qty	Description	Unit Price	Line Total

29485: Catherine Abel
Professional Sales and Service
29486: Kim Abercrombie
Riders Company
29489: Frances Adams
Area Bike Accessories
29490: Margaret Smith
Bicycle Accessories and Kits
29492: Jay Adams
Valley Bicycle Specialists
29494: Samuel Agcaoili
Vinyl and Plastic Goods Corpor
29496: Robert Ahlering
Fun Toys and Bikes
29497: François Ferrier
Great Bikes
29499: Amy Alberts
Valley Toy Store
29502: Paul Alcorn
Major Sport Suppliers
29503: Gregory Alderson
Family's Favorite Bike Shop
29503: Gregory Alderson
Family's Favorite Bike Shop
29505: Michelle Alexander
Imported and Domestic Cycles
29506: Sean Jacobson
Systematic Sales
29508: Marvin Allen
Mountain Toy Store
29510: Cecil Allison
Designated Distributors
29511: Oscar Alpuerto
Bold Bike Accessories
29515: Maxwell Amland
Serious Cycles

Figure 11 **The CRM Add-In Within Word**

The last thing to do is set the property to determine when the Ribbon will be displayed. By default, the Ribbon is displayed on the Mail Explorer. This is the window you see when you open a mail item. In this sample, you want the Ribbon to display on the main screen. Select the Ribbon and set the RibbonType property to Microsoft.Outlook.Explorer. You can see there are a number of places where the Ribbon may appear, including the Mail and Contact Explorers.

Next, double-click on your Ribbon button to create a code-behind click event handler. This is the event you'll use to create the Outlook contact.

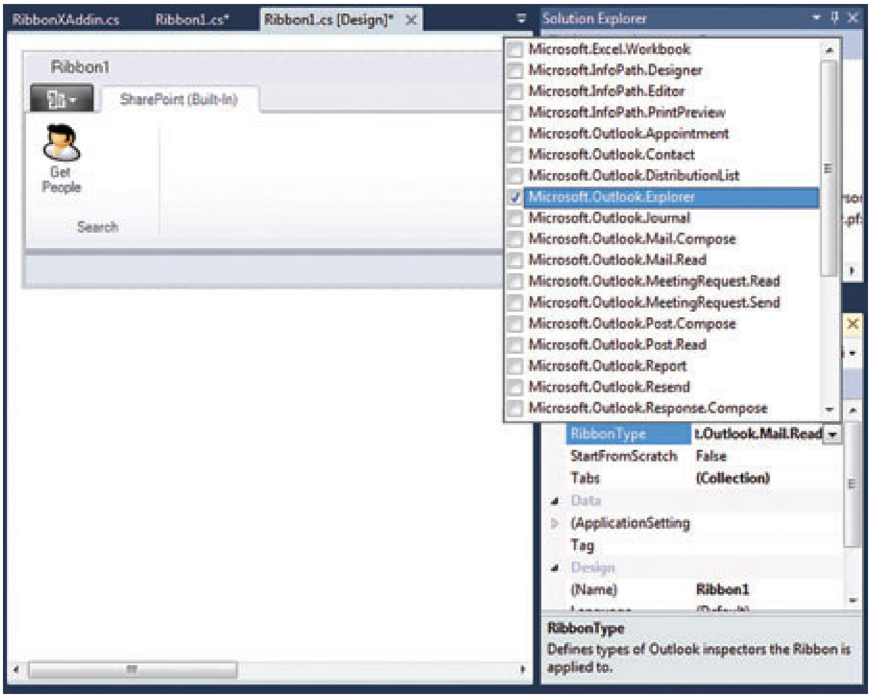
You're now ready to add the code that creates a contact in Outlook. Visual Studio 2010 makes this easy to do. I find it easier to break the problem into multiple smaller parts. First, you created the Outlook add-in, then you created the Ribbon. After each of these steps, make sure you press F5 to compile and run your application. Now you can create an Outlook contact using hard-coded values. After you verify that this is working, you can add the code that calls SharePoint. Again, at each step check that everything is working correctly before moving on to the next step.

Figure 13 shows the code to create a new hard-coded contact. This uses the CreateItem method to create a new ContactItem object. Then you can set the properties of the ContactItem and call the Save method to commit the changes.

The only really challenging piece is that the way to set the contact picture is to call the AddPicture method, which takes a path to a picture on disk. This is problematic because you want to pull images from SharePoint. You'll see how to do this in the next section. Once you verify that the code works and a contact is created in Outlook, you're ready to call SharePoint and add real contacts.

Employing User Profile Service

UserProfileService is a SharePoint Web service you can use to access profile information, including a list of your colleagues and their profile information. To use this service, start by adding a reference to your project. Because this is a Web service and not a WCF service, you need to click the advanced tab of the Add Service


RibbonXAddin.cs
Ribbon1.cs*
Ribbon1.cs [Design]*
Solution Explorer
Ribbon1
SharePoint (Built-In)
Get People
Search
Microsoft.Excel.Workbook
Microsoft.InfoPath.Designer
Microsoft.InfoPath.Editor
Microsoft.InfoPath.PrintPreview
Microsoft.Outlook.Appointment
Microsoft.Outlook.Contact
Microsoft.Outlook.DistributionList
Microsoft.Outlook.Explorer
Microsoft.Outlook.Journal
Microsoft.Outlook.Mail.Compose
Microsoft.Outlook.Mail.Read
Microsoft.Outlook.MeetingRequest.Read
Microsoft.Outlook.MeetingRequest.Send
Microsoft.Outlook.Post.Compose
Microsoft.Outlook.Post.Read
Microsoft.Outlook.Report
Microsoft.Outlook.Resend
Microsoft.Outlook.Response.Compose
Data
Design

RibbonType	t.Outlook.Mail.Read
StartFromScratch	False
Tabs	(Collection)
(ApplicationSetting	
Tag	
(Name)	Ribbon1

RibbonType
Defines types of Outlook inspectors the Ribbon is applied to.

Figure 12 **Creating a New Outlook Ribbon**

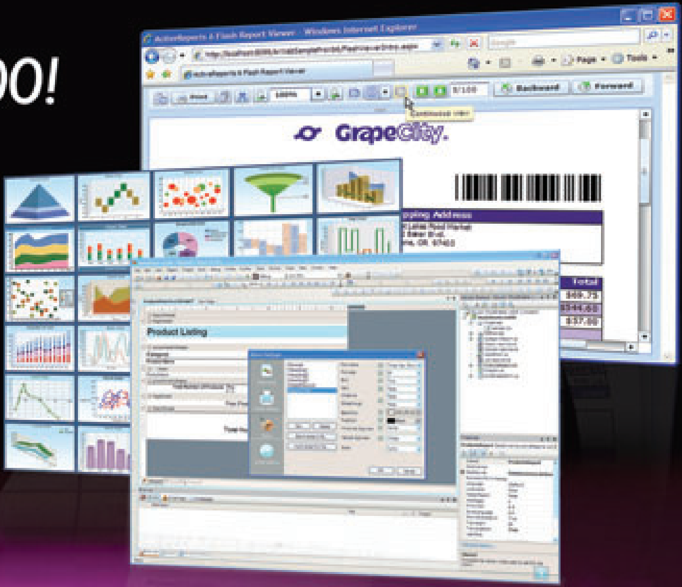

ActiveReports 6 Flash Report Viewer - Windows Internet Explorer
GrapeCity
Backward
Forward
Shipping Address
Total
Product Listing

Figure 13 **Boilerplate Code to Create a Contact**

```
Outlook.ContactItem newContact =
  Globals.ThisAddIn.Application.CreateItem(
  Outlook.OlItemType.olContactItem);

newContact.FirstName = "Paul";
newContact.LastName = "Stubbs";
newContact.Email1Address = "pstubbs@microsoft.com";
newContact.CompanyName = "Microsoft";
newContact.JobTitle = "Technical Evangelist";
newContact.CustomerID = "123456";
newContact.PrimaryTelephoneNumber = "(425)555-0111";
newContact.MailingAddressStreet = "1 Microsoft Way";
newContact.MailingAddressCity = "Redmond";
newContact.MailingAddressState = "WA";
newContact.AddPicture(@"C:\me.png");
newContact.Save();
```

dialog, then click the Add Web Service button. This opens the old Add Web Service dialog that you remember from Visual Studio 2005.

After you add the reference, you can add the code to retrieve your colleagues:

```
// Instantiate the Web service.
UserProfileService userProfileService =
  new UserProfileService();

// Use the current user log-on credentials.
userProfileService.Credentials =
  System.Net.CredentialCache.DefaultCredentials;
```

This code creates an instance of the service and passes your current credentials to the service. Next, call the GetUserColleagues method passing the user that you want to retrieve colleagues for. This will return an array of ContactData objects:

```
ContactData[] contacts =
  userProfileService.GetUserColleagues(
  "contoso\\danj");
```

SharePoint offers Web services as the primary way to access SharePoint data remotely.

We can now loop through all of the ContactData objects that represent profile data for the user's colleagues in SharePoint. We retrieve the extended properties by calling the GetUserProfileByName method, which returns an array of PropertyData that contains key and value pairs for each colleague:

```
// Add each Colleague as an Outlook Contact
foreach (ContactData contact in contacts) {
  // Get the users detailed Properties
  PropertyData[] properties =
  userProfileService.GetUserProfileByName(contact.AccountName);

  // Create a new Outlook Contact
  Outlook.ContactItem newContact =
    Globals.ThisAddIn.Application.CreateItem(
    Outlook.OlItemType.olContactItem);
```

Now we convert those key/value pairs into contact properties:

```
// Set the Contact Properties
newContact.FullName = contact.Name;
newContact.FirstName =
  properties[2].Values[0].Value.ToString();
newContact.LastName =
  properties[4].Values[0].Value.ToString();
newContact.Email1Address =
  properties[41].Values[0].Value.ToString();
...
```

Finally, we grab the contact photo and save the new contact:

```
// Download the users profile image from SharePoint
SetContactImage(properties, newContact);

newContact.Save();
```

The last piece of the puzzle is retrieving the contact's picture from SharePoint. One of the extended properties includes a path to a thumbnail of the user's profile picture. You need to download this picture to a temporary file on disk so that the Outlook API can add it to the ContactItem object:

```
private static void SetContactImage(
  PropertyData[] properties,
  Outlook.ContactItem newContact){

  // Download image to a temp file
  string userid = properties[16].Values[0].Value.ToString();
  string imageUrl = properties[15].Values[0].Value.ToString();
  string tempImage = string.Format(@"C:\{0}.jpg", userid);
  WebClient Client = new WebClient();
  Client.Credentials = CredentialCache.DefaultCredentials;
  Client.DownloadFile(imageUrl, tempImage);
  newContact.AddPicture(tempImage);
}
```

That's it! Now you have an Outlook add-in Ribbon that calls SharePoint to pull social data into Outlook contacts. When you run the application, you'll see a ContactItem populated with SharePoint data, including the user's profile information and image.

Wrap Up

Now you've seen how easy it is to get data from SharePoint into Office clients. We've shown you a variety of options from no-code solutions to highly adaptable solutions using C# or Visual Basic.

Employing WCF Data Services to access SharePoint list data provides a common pattern for .NET developers that's quick and easy to implement. The client object model provides the means to access SharePoint external lists and opens a world of opportunities for bringing LOB data into Office. And, finally, SharePoint Web services enables the most flexible access to data, but also requires a bit more commitment in terms of coding and testing.

Making data in SharePoint available to users as lists is an important step as it enables a great experience in the browser. Taking it a step further, you can leverage a variety of data access options to then bring the data into the Office applications that are familiar to users. Visual Studio 2010 makes all of this much easier to build, debug and deploy. As you can see, these represent some of the new and important development capabilities you can take advantage of with the new product releases.

More training, examples and information can be found online in the Office (msdn.microsoft.com/office) and SharePoint (msdn.microsoft.com/sharepoint) developer centers. ■

DONOVAN FOLLETTE *is a Microsoft technical evangelist working with technologies including Active Directory, Lightweight Directory Services and Active Directory Federation Services. He now focuses on Office development and building integrated solutions with SharePoint 2010. Visit his blog at blogs.msdn.com/donovanf.*

PAUL STUBBS *is a Microsoft technical evangelist who focuses on the information worker development community for SharePoint and Office, Silverlight and Web 2.0 social networking. He's authored three books about solution development with Office, SharePoint and Silverlight. Read his blog at blogs.msdn.com/pstubbs.*

THANKS to the following technical expert for reviewing this article:
John Durant

Alien approved for 20+ Light Years!
THE CONTROLS HAVE LANDED.
180
160
140
120
100
80
s2
s3
s4
s5
240
220
200
180
160
140
120
s6
s1
s7
June
July
2010
ComponentOne
Studio Enterprise 2010
Get yours today at: COMPONENTONE.COM/HERE

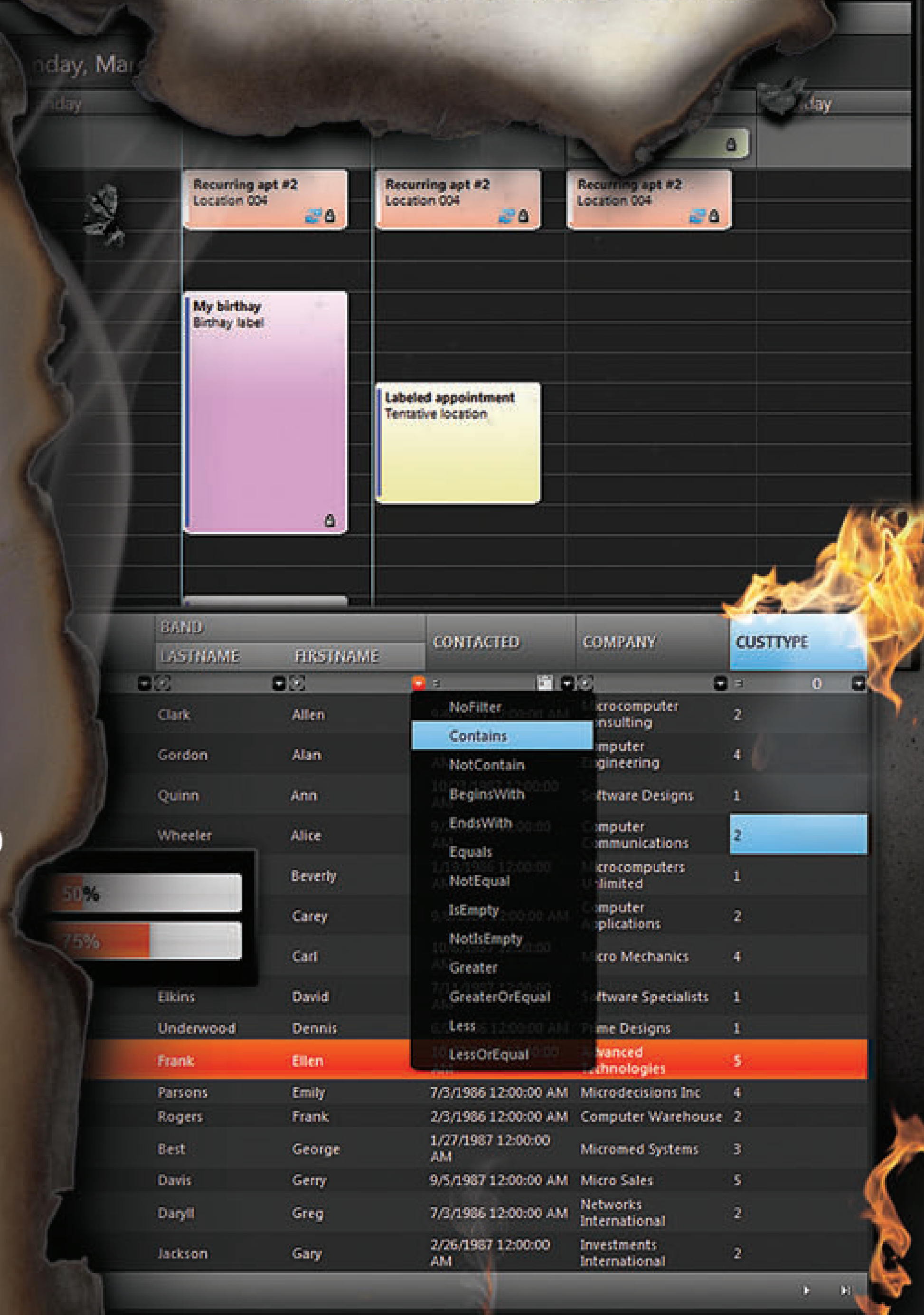
Sheela Castle	20	Peter Campbell 8287	Netherlands	Vanuatu
Irina Vera	37	Kenedy Avenue 9758	Botswana	Vatican City
Noela Meredith	39	5th Avenue 3139	India	Macau --
Michael Days	48	10th Avenue 7141	Namibia	Australia
Alvaro Johansen	21	13th Street 9183	Australia	Cuba
Noela Vera	37	Missisipi 8428	Cape Verde	Gabon
Patric Johnann	19	[illegible]	[illegible]	Tunisia

100'S OF CONTROLS, 7 PLATFORMS

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

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

SOFT VISUAL STUDIO 2010-READY



BAND				
LASTNAME	FIRSTNAME	CONTACTED	COMPANY	CUSTTYPE
		=		= 0
Clark	Allen		crocomputer nsulting	2
Gordon	Alan		mputer gineering	4
Quinn	Ann		ftware Designs	1
Wheeler	Alice		mputer mmunications	2
	Beverly		crocomputers limited	1
	Carey		mputer plications	2
	Carl		cro Mechanics	4
Elkins	David		ftware Specialists	1
Underwood	Dennis		me Designs	1
Frank	Ellen		vanced chnologies	5
Parsons	Emily	7/3/1986 12:00:00 AM	Microdecisions Inc	4
Rogers	Frank	2/3/1986 12:00:00 AM	Computer Warehouse	2
Best	George	1/27/1987 12:00:00 AM	Micromed Systems	3
Davis	Gerry	9/5/1987 12:00:00 AM	Micro Sales	5
Daryll	Greg	7/3/1986 12:00:00 AM	Networks International	2
Jackson	Gary	2/26/1987 12:00:00 AM	Investments International	2

Experience the rich, new features of **Visual Studio 2010** along with the ability to:

- Improve your Web Form apps in minutes with AJAX 4.0 controls.
- Style your Silverlight applications using visual brush creation and ComponentOne ClearStyle technology.
- Add data visualization such as charts and gauges in WinForms and ASP.NET AJAX.
- Add docking and floating capabilities to WPF windows.

ASP.NET · AJAX · JQUERY · CSS · XHTML · JSON · W3C STANDARDS · CLIENT SIDE OBJECT MODEL (CSOM) · ANIMATION · THEME DESIGNER (FOXY)

Studio for ASP.NET AJAX 2010

FUELED BY THE NEW **A2 FRAMEWORK**

Sold separately. Also part of Studio Enterprise.

ComponentOne

COMPONENTONE.COM/AJAX

ComponentOne Sales: 1.800.858.2739 or 1.412.681.4343

Trim SharePoint Search Results for Better Security

Ashley Elenjickal and Pooja Harjani

Microsoft SharePoint search uses an account that usually has full read access across the repository to index its contents. So it's important that when a user queries for some content, he should be restricted to view only the documents he has permission to see. SharePoint uses the access control list (ACL) associated with each document to trim out query results that users have no permission to view, but the default trimming provided by SharePoint (out-of-box trimming) may not always be adequate to meet data security needs. In that case, you may want to further trim the results depending on an organization's authentication structure.

This is where the SharePoint custom security trimming infrastructure is useful. SharePoint lets you implement business logic in a separate module and then integrate it into the workflow of the query processor that serves the queries. In the security trimming path, custom query trimming follows out-of-box security trimming. So the number of query results after custom trimming must be equal to or less than the number of documents recalled before registering the custom security trimmer (CST) assembly.

Before delving into the CST architecture, we'll provide a quick view of SharePoint search and the new claims authentication infrastructure.

This article discusses:

- Claims authentication in SharePoint 2010
- Deploying a custom security trimmer
- Using PowerShell cmdlets
- Troubleshooting

Technologies discussed:

SharePoint, custom security trimmer

Code download available at:

code.msdn.microsoft.com/mag201007Search

SharePoint Search Overview

At a high level, the search system can be divided into two discrete parts: the gatherer pipeline and the query processor pipeline.

Gatherer Pipeline This part is responsible for crawling and indexing content from various repositories, such as SharePoint sites, HTTP sites, file shares, Lotus Notes, Exchange Server and so on. This component lives inside MSSearch.exe. When a request is issued to crawl a repository, the gatherer invokes a filter daemon, MssDmn.exe, to load the required protocol handlers and filters necessary to connect, fetch and parse the content. **Figure 1** represents a simplified view of the gatherer pipeline.

SharePoint can only crawl using a Windows NTLM authentication account. Your content source must authorize the Windows account sent as part of the crawl request in order to access the document content. Though claims authentication is supported in SharePoint 2010, the gatherer is still not a claims-aware application and will not access a content source that has claims authentication only.

Query Processor Pipeline In SharePoint 2010, two of the most important changes in the query processor pipeline are in its topological scalability and authentication model. In Microsoft SharePoint Server (MOSS) 2007, the query processor (search query and site settings service, referred to as search query service from here on) runs in the same process as Web front end (WFE), but in SharePoint 2010 it can run anywhere in the farm—and it also runs as a Web service.

The WFE talks to the search query service through Windows Communication Foundation (WCF) calls. The search query service is now completely built on top of the SharePoint claims authentication infrastructure. This decouples SharePoint search from its tight integration with Windows authentication and forms authentication. As a result, SharePoint now supports various authentication models. The search query service trims the search results according to the rights of the user who issues the query. Custom security trimmers are called by the search query service after out-of-box trimming has completed. See **Figure 2** for the various components involved when a query is performed.

Custom security trimming is part of the query pipeline, so we'll limit this discussion to components of the query pipeline.

Claims Authentication in SharePoint 2010

A basic understanding of claims authentication support in SharePoint 2010 is required to implement custom trimming logic inside a CST assembly. In the claims authenticated environment, the user identity is maintained inside an envelope called a security token. It contains a collection of identity assertions or claims about the user. Examples of claims are username, e-mail address, phone number, role and so on. Each claim will have various attributes such as *type* and *value*. For example, in a claim the UserLogonName may be the *type* and the name of the user who is currently logged in may be the *value*.

Security tokens are issued by an entity called a security token service (STS). This is a Web service that responds to user authentication requests. Once the user is authenticated, STS sends back a security token with all the user rights. STS can be configured either to live inside the same SharePoint farm or act as a relying party to another STS that lives outsides the farm: Identity Provider-STS (IP-STS) and Relying Party-STS (RP-STS), respectively. Whether you want to use IP-STS or RP-STS has to be carefully considered while designing SharePoint deployment.

SharePoint uses the default claims provider shipped with the product in a simple installation. Even if you set up the farm completely using Windows authentication, when a query is issued, a search service application proxy will talk to STS to extract all the claims of the user in a security token. This token is then passed to the search query service through a WCF call.

Workflow of Custom Security Trimming

The workflow logic of a CST can be represented in a simple flowchart as shown in **Figure 3**.

As stated earlier, the search query service first performs out-of-box security trimming and then looks for the presence of any CSTs associated with the search results. The association of a particular content source with a CST is done by defining a *crawl rule* for that specific content source. If the search query service finds any CST associated with any of the URLs in the search results, it calls into that trimmer. Trimmers are loaded into the same IIS worker process, w3wp.exe, in which the search query service is running.

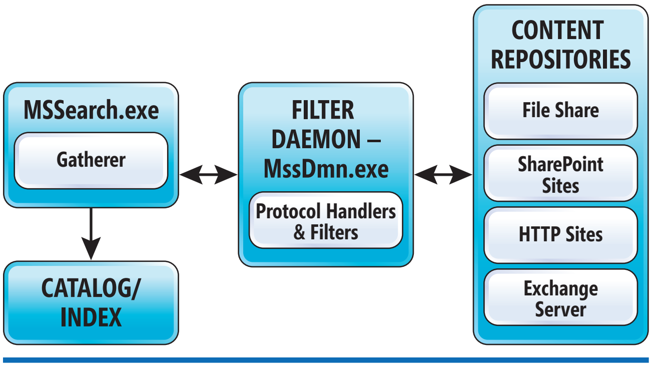


Figure 1 **A Simplifed View of the SharePoint Gatherer Pipeline**

Once the trimmer is loaded, the search query service calls into the CheckAccess method implemented inside the trimmer with an *out-of-box trimming* result set associated with the crawl rule that you defined earlier. The CheckAccess method decides whether a specific URL should be included in the final result set sent back to the user. This is done by returning a bit array. Setting a bit inside this array to either true or false will "include" or "block" the URL from the final result set. In case you want to stop processing the URLs due to performance or some unexpected reason, you must throw a PluggableAccessCheckException. If you throw after processing a partial list of URLs, the processed results are sent back to the user. The search query service will remove all the unprocessed URLs from the final result set.

SharePoint can only crawl using a Windows NTLM authentication account.

Steps Involved in Deploying a Custom Security Trimmer

In a nutshell, there are five steps involved in the successful deployment of a CST:

1. Implement ISecurityTrimmer2 interface.
   a. Implement Initialize and CheckAccess methods using managed code
   b. Create an assembly signing file and include it as part of the project
   c. Build the assembly
2. Deploy the trimmer into the Global Assembly Cache (GAC) of all the machines where a search query service is running.
3. Create a crawl rule for the content sources that you want to custom trim. You can do this from the Search Administration site.
4. Register the trimmer with the crawl rule using the Windows PowerShell cmdlet *New-SPEnterpriseSearchSecurityTrimmer*.

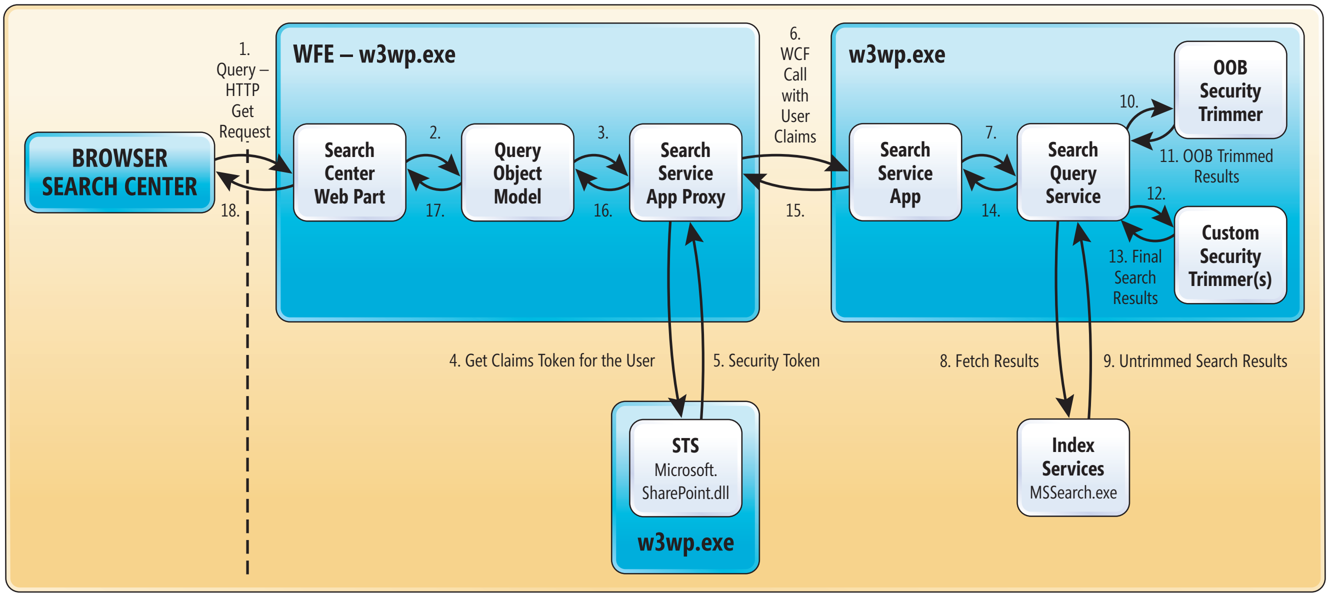


Figure 2 **Workflow of a Query Originating from the Search Center in a SharePoint Site**

5. Perform a *full crawl* of the content sources associated with the crawl rules that you created in step 3. A full crawl is required to properly update all of the related database tables. An incremental crawl will not update the appropriate tables.

Implementing the Custom Security Trimmer Interface

MOSS 2007 and Microsoft Search Server (MSS) 2008 supported custom security trimming of search results through the interface ISecurityTrimmer. This interface has two methods, Initialize and CheckAccess. Because of the architectural changes in SharePoint and the search system in the 2010 versions, both of these methods won't work as they did in MOSS 2007. They need to be re-implemented using the ISecurityTrimmer2 interface. As a result, if you try to register a MOSS 2007 trimmer in SharePoint 2010, it will fail, saying ISecurityTrimmer2 is not implemented. Other changes from MOSS 2007 include:

Changes in the Initialize Method In MOSS 2007, one of the parameters passed was the SearchContext object. SearchContext was the entry point into the search system and it provided the search context for the site or search service provider (SSP). This class has been deprecated in 2010. Instead, use the SearchServiceApplication class:

```
void Initialize(NameValueCollection staticProperties,
SearchServiceApplication searchApplication);
```

Changes in the CheckAccess Method In both MOSS 2007 and SharePoint 2010, the search query service calls into the CST assemblies. In MOSS 2007, the CheckAccess method took only two parameters, but in SharePoint 2010, the search query service passes the user identity into CheckAccess using a third parameter of type IIdentity:

```
public BitArray CheckAccess(IList<String>documentCrawlUrls,
IDictionary<String, Object>sessionProperties, IIdentity passedUserIdentity)
```

ISecurityTrimmer2::Initialize Method This method is called the first time a trimmer is loaded into the search query service IIS worker process. The assembly will live for the duration of the worker process. Here's the signature of this method and a description of how it works:

```
void Initialize(NameValueCollection staticProperties,
SearchServiceApplication searchApplication);
```

staticProperties–The trimmer registration Windows PowerShell cmdlet, New-SPEnterpriseSearchSecurityTrimmer, takes a parameter called "properties" (in MOSS 2007 this was called "configprops") through which you can pass named value pairs separated by ~. This may be useful to initialize your trimmer class properties.

For example: When passing "superadmin~foouser~poweruser~baruser" to the New-SPEnterpriseSearchSecurityTrimmer cmdlet, the NameValueCollection parameter will have two items in the collection with keys as "superadmin" and "poweruser" and values as "foouser" and "baruser," respectively.

searchApplication–If your trimmer requires a deeper knowledge about the search service instance and the SharePoint farm, use a searchApplication object to determine that information. To learn more about the SearchServiceApplication class, refer to msdn.microsoft.com/library/ee573121(v=office.14).

ISecurityTrimmer2::CheckAccess Method This implements all the trimming logic. Pay special attention to two aspects in this method: the identity of the user who issued the query, and the performance latency caused by a large returned query set.

Following are the signature of this method and a description of how it works:

```
public BitArray CheckAccess(IList<String>documentCrawlUrls,
IDictionary<String, Object>sessionProperties, IIdentitypassedUserIdentity)
```

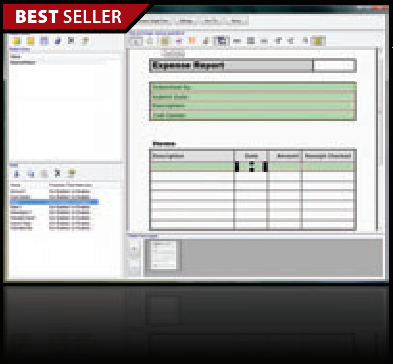
documentCrawlUrls–The collection of URLs to be security trimmed by this trimmer.

sessionProperties–A single query instance is treated as one session. If your query fetches many results, the CheckAccess method is called multiple times. You can use this parameter to share values or to keep track of the URLs processed between these calls.

passedUserIdentity–This is the identity of the user who issued the query. It's the identity by which the code will allow or deny access to content.

ComponentSource®
The Definitive Source of Software Components
www.componentsource.com

BEST SELLER



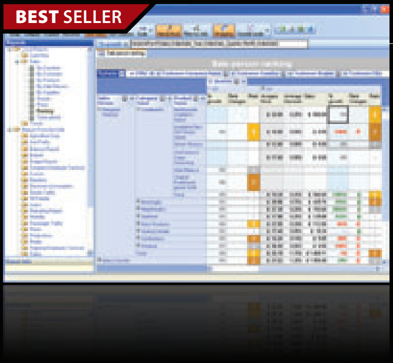
LEADTOOLS Recognition SDK | from $3,595.50

LEAD TECHNOLOGIES INCORPORATED

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

- Features accurate, high-speed multi-threaded OCR and forms recognition
- Supports text, OMR, image, and 1D/2D barcode fields
- Auto-registration and clean-up to improve recognition results
- Includes .NET, C/C++, WPF, WF, WCF and Silverlight interfaces
- Includes comprehensive confidence reports to assess performance

BEST SELLER



ContourCube | from $900.00

Contour Components

OLAP component for interactive reporting and data analysis.

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

BEST SELLER

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

TX TEXT CONTROL word processing components

Word processing components for Visual Studio .NET.

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

BEST SELLER

FusionCharts | from $195.02

InfoSoft Global empowering human thoughts

Interactive and animated charts for ASP and ASP.NET apps.

- Liven up your Web applications using animated Flash charts
- Create AJAX-enabled charts that can change at client-side without invoking server requests
- Export charts as images/PDF and data as CSV for use in reporting
- Also create gauges, financial charts, Gantt charts, funnel charts and over 550 maps
- Used by over 15,000 customers and 250,000 users in 110 countries

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

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

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

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

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

VISA DISCOVER

GSA Schedule Contract GS-35F-0168R

BitArray–You need to return a bit array equal to the number of items in documentCrawlUrls. Setting a bit inside this array to true or false will determine whether the URL at that position should be included or blocked from the final result set sent back to the user.

UserIdentity The SharePoint 2010 search query engine is built upon the claims authentication model. The search query service will pass the query issuer's claims though the IIdentity parameter. In order to get the user name of the user who issued the query, you must traverse through a collection of claims to compare the claim.ClaimType with the SPClaimTypes.UserLogonName.

> The search query service trims the search results according to the rights of the user who issues the query.

The following snippet of code extracts the user logon name from the claims token:

```
IClaimsIdentity claimsIdentity = (IClaimsIdentity)passedUserIdentity;

if (null != claimsIdentity)
{
  foreach (Claim claim in claimsIdentity.Claims)
  {
    if (claim == null)
      continue;
    if (SPClaimTypes.Equals(claim.ClaimType, SPClaimTypes.UserLogonName))
      strUser = claim.Value;
  }
}
```

You may need information about the type of authentication used at the site collection level to correctly call internal APIs. To identify if the user logged in using Windows authentication, look for the presence of *ClaimsType.PrimarySid*. The following code looks for the *PrimarySid* claim and then extracts the user name from it:

```
if (SPClaimTypes.Equals(claim.ClaimType, ClaimTypes.PrimarySid))
{
  // Extract SID in the format “S-1-5-21-xxxxx-xxxxx-xxx”
  strUser = claim.Value;
  // Convert SID into NT Format “FooDomain\BarUser”
  SecurityIdentifier sid = new SecurityIdentifier(strUser);
  strUser = sid.Translate(typeof(NTAccount)).Value;
}
```

For forms or other similar non-Windows authentication providers, look at the *Claim.OriginalIssuer* value inside the claim. For example, if the server is configured for forms authentication using the ASP.NET SQL Membership Provider, the Claim.OriginalIssuer will have the value "Forms:AspNetSqlMembershipProvider":

```
if (SPClaimTypes.Equals(claim.ClaimType, SPClaimTypes.UserLogonName))
{
  strUser = claim.Value;
  strProvider = claim.OriginalIssuer; // For AspNet SQL Provider value will be
                                      // "Forms:AspNetSqlMembershipProvider"
}
```

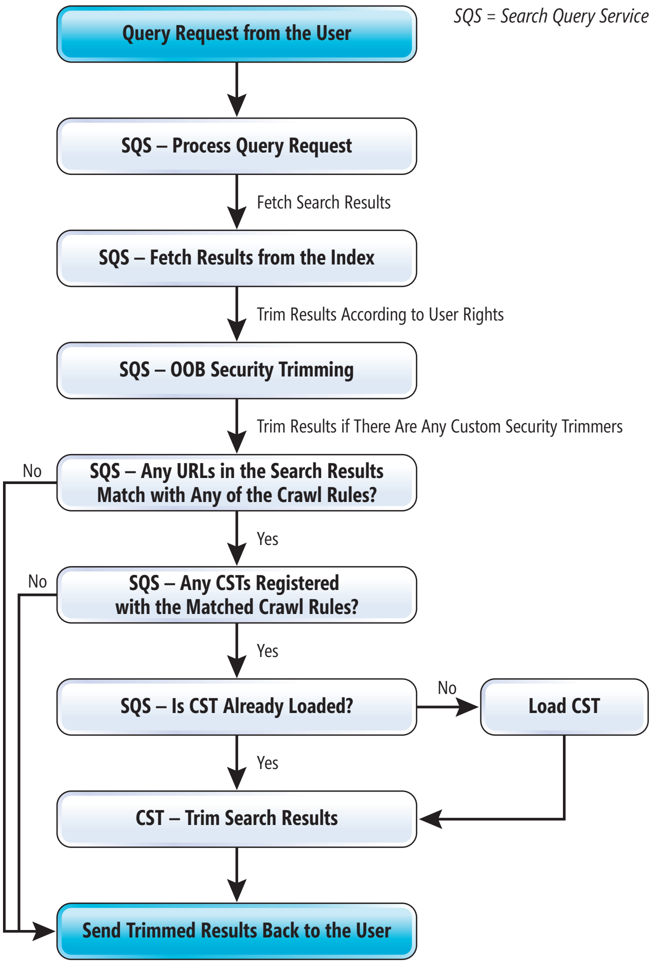
If the query is issued by an anonymous user, the value of the IIdentity.IsAuthenticated method will be false. In this case, claimsIdentity.Name will have the value "NT AUTHORITY\\ ANONYMOUS LOGON."

As a final note on the user context, limit the use of the API WindowsIdentity.GetCurrent().Name to retrieve the user identity. This will always give the application pool identity under which search query service is running. System.Threading.Thread.CurrentPrincipal.Identity will give you the same identity as the one passed to the CheckAccess method.

Performance Considerations Optimize the CheckAccess method to its fullest extent. If the query returns many results, the trimmer may get called multiple times. One of the common methods to take care of this situation is to keep track of the URLs processed inside the trimmer through the sessionProperties parameter. Once the method processes a certain number of result sets, it can throw a PluggableAccessCheckException. When this exception is thrown, the URLs processed up to that point are returned to the user.

Custom Security Trimmer and System Logs

Code inside a trimmer can't write to the system logs maintained at <drive>\ Program Files\Common Files\Microsoft Shared\Web Server Extensions\14\LOGS. The trimmer must maintain its own logging mechanism for both debugging and auditing. The only exception to this is when the method throws the PluggableAccessCheckException. The message string specified while throwing will be logged into the system log. Useful information that the search query service logs



SQS = Search Query Service
Query Request from the User
SQS – Process Query Request
Fetch Search Results
SQS – Fetch Results from the Index
Trim Results According to User Rights
SQS – OOB Security Trimming
Trim Results if There Are Any Custom Security Trimmers
No
SQS – Any URLs in the Search Results Match with Any of the Crawl Rules?
Yes
No
SQS – Any CSTs Registered with the Matched Crawl Rules?
Yes
SQS – Is CST Already Loaded?
No
Load CST
Yes
CST – Trim Search Results
Send Trimmed Results Back to the User

Figure 3 **The Workflow Logic of a CST**

to the file includes the number of documents that were security trimmed. For example, the following log entry suggests that a query passed two documents to the CST, but sent zero documents back to the user, which means the CST trimmed those two documents:

```
04/23/2010 18:13:48.67    w3wp.exe (0x116C)    0x02B4    SharePoint
Server Search    Query Processor    dm2e    Medium    Trim results:
First result position = '0', actual result count = '0', total docs found
= '0', total docs scanned = '2'.    742d0c36-ea37-4eee-bf8c-f2c662bc6a45
```

Custom Security Trimmers and Alerts The SharePoint search service has a feature called *alerts* (available only in Windows authentication mode) that can push the changes in the query results to the user through e-mails. However, when an alert query is issued by the timer service, the search query service will strip out all the URLs associated with CSTs.

Use Windows PowerShell cmdlets to register, view and delete CSTs.

Assembly Signing Requirement On finding the presence of a matching CST, the search query service calls into CST management code to load the specific assembly from the GAC. To do this, the assembly needs to be digitally signed. Refer to "Managing Assembly and Manifest Signing" (msdn.microsoft.com/library/ms247066) for ways to sign an assembly. Once the assembly is built, use the sn.exe tool to get the 64-bit hash known as a public key token. This token is needed at the time of trimmer registration.

Deployment of Custom Security Trimmer The CST assembly must reside in the GAC of each machine on which the search query and site settings service is running. Use Central Administration | System Settings | Services on Server to check the status of the search query and site settings service in each of the machines in the farm. If the service is started, you must import the CST to that machine. Don't confuse the search query and site settings service with the machines that contain query components. The query component lives within MSSearch.exe to pull the results from the index. The search query and site settings service lives in its own IIS worker process of w3wp.exe.

SharePoint Cmdlets to Register, View and Delete CSTs

MOSS 2007 used the stsadm.exe command-line tool to register custom trimmers, but this tool is obsolete and not supported in SharePoint 2010. Instead, use Windows PowerShell cmdlets to register, view and delete CSTs. An assembly should already be available in the GAC to register them. Here's how to use them:

Registration–Use the New-SPEnterpriseSearchSecurityTrimmer to register your trimmer, using the assembly's manifest data such as Version, Culture and PublicKeyToken. This example registers the trimmer to the search application named "search service application":

```
New-SPEnterpriseSearchSecurityTrimmer -SearchApplication "Search
Service Application" -TypeName "SearchCustomSecurityTrimmer.
CustomSecurityTrimmerTest, SearchCustomSecurityTrimmer, Version=14.0.0.0,
Culture=neutral, PublicKeyToken=4ba2b4aceeb50e6d" -RulePath file://
elenjickal2/* -id 102 -Properties superadmin~foouser~poweruser~baruser
```

The cmdlet takes the crawl rule (RulePath), an integer value as the identity (id) of the trimmer, configuration properties (properties) and TypeName, which consists of the manifest data as well as the name of the class that implements the interface. Cmdlet parameters are:

- SearchApplication–Name of the search service application associated with the content source
- TypeName–This consists of the manifest data such as Version, Culture and PublicKeyToken (it also points to the class that implements the interface; this will uniquely identify the assembly from the GAC)
- RulePath–The crawl rule associated with the trimmer
- Id–An int data type that uniquely identifies the trimmer instance
- Properties–Set of name/value pairs separated by ~

View–Use the Get-SPEnterpriseSearchSecurityTrimmer cmdlet and pass the search application name. You can further filter it by passing trimmer identity or other properties that you used while registering (for example: Get-SPEnterpriseSearchSecurityTrimmer -SearchApplication "Search Service Application").

Delete–Use the Remove-SPEnterpriseSearchSecurityTrimmer cmdlet and pass the search application name as well as identity of the trimmer (for example: Remove-SPEnterpriseSearchSecurity-Trimmer -SearchApplication "Search Service Application" –id 102).

Note: After registering the CST, a full crawl of the content source is required.

Troubleshooting Steps

Here are some tips to investigate any unexpected search results:

- Make sure the crawl rule matches the content source location.
- Check the crawl logs to make sure the account used to crawl the content source has access to it. The crawl would have failed if it doesn't.
- Make sure the query user has permission to view the content.
- After trimmer registration, make sure you performed a full crawl.
- Make sure the trimmer assembly is in the GAC of all machines in which search query service is running.
- Check the system logs for the number of documents trimmed by the security trimmer.
- Use the utility ProcessExplorer from technet.microsoft.com/sysinternals/bb896653 to make sure the trimmer assembly is loaded into IIS worker process w3wp.exe.
- Attach the debugger to the worker process in which the assembly is loaded and step through the trimmer logic.

Query Processing Logic Flexibility

Wrapping up, CSTs provide the flexibility to extend the query processing logic to meet customized enterprise security needs. One should always keep in mind that implementation bugs inside the trimmer may cause unexpected search results, so it's important that before the trimmer is deployed in a production environment, it's thoroughly tested against different types of content sources and authentication providers. ■

ASHLEY ELENJICKAL AND POOJA HARJANI *were part of a SharePoint Search feature team responsible for Custom Security Trimmer at Microsoft. They can be reached at AshleyEl@microsoft.com and PVaswani@microsoft.com, respectively.*

THANKS to the following technical expert for reviewing this article: *Michal Piaseczny*

Creating OneNote 2010 Extensions with the OneNote Object Model

Andy Gray

Microsoft Office OneNote is a powerful digital notebook for collecting, organizing, searching and sharing information. With the recent release of Microsoft Office 2010, not only is the OneNote user experience improved, but OneNote notebooks are now more universally available. Users can synchronize content among computers via Windows Live; search, edit and share notes from any Web browser; and access full notebooks from Windows Mobile (and, soon, Windows Phone 7). Further, OneNote was previously included only in some Office editions, but it's now in every edition of Office 2010. All of these factors create a more compelling opportunity than ever before to integrate OneNote into information management solutions.

The OneNote Object Model library on CodePlex, to which this article refers, had not been updated for compatibility with OneNote 2010 at the time of this writing.

This article discusses:

- The evolution of OneNote development
- Accessing OneNote data using the COM API
- Retrieving and updating page content using the COM API
- The OneNote Object Model Library
- Data binding with the OneNote Object Model Library

Technologies discussed:

OneNote 2010, OneNote 2007, Visual Studio 2010, LINQ, OneNote Object Model, XAML Data Binding, Windows Presentation Foundation, C#

Code download available at:

code.msdn.microsoft.com/mag201007OneNote

In this article, I'll provide an overview of developing applications that interoperate with data from Microsoft OneNote 2010 and 2007. In the process, I'll introduce the OneNote Object Model project that is freely available on CodePlex and demonstrate how this library makes it easy to integrate information from OneNote notebooks, sections and pages into client applications.

The Evolution of OneNote Development

The initial release of OneNote 2003 didn't provide an API to external applications. Shortly thereafter, however, OneNote 2003 SP 1 added a COM library, called the OneNote 1.1 Type Library, which enabled programmatic import of images, ink and HTML into OneNote via a simple class called CSimpleImporter. Notably, however, this class only provided data import capabilities; you could use it to push data into OneNote notebooks, but there was no way to get content back out programmatically.

The release of OneNote 2007 brought much more powerful development capabilities with a new COM API that provides the ability to import, export and modify OneNote 2007 content programmatically. The OneNote Application class in that library provides a rich collection of methods for working with:

- **Notebook structure:** discovering, opening, modifying, closing and deleting notebooks, section groups and sections
- **Page content:** discovering, opening, modifying, saving and deleting page content
- **Navigation:** finding, linking to and navigating to pages and objects

Most of these methods return or accept XML documents that represent both notebook structure and page content. Saul Candib wrote a two-part series, "What's New for Developers in OneNote 2007," that documents this API at msdn.microsoft.com/library/ms788684(v=office.12), and the XML schema is detailed at msdn.microsoft.com/library/aa286798(office.12).

The XML schema for OneNote 2010 is substantially similar to that in OneNote 2007. OneNote 2010 introduces a file format change to support some of its new features (such as linked note-taking, versioning, Web sharing, multilevel subpages and equation support). However, OneNote 2010 can continue to work on OneNote 2007 notebooks without changing the file format. In OneNote 2010, retrieving data from sections stored in the OneNote 2007 file format will yield XML documents similar to those in OneNote 2007. The primary differences in the XML schema for OneNote 2010 sections are additive changes to support the new features listed earlier. A new XMLSchema enumeration is available to represent the OneNote schema version; many of the OneNote methods have new overloads that take an XMLSchema parameter to indicate the schema version desired.

Note that the CSimpleImporter class, introduced in OneNote 2003 and still available in OneNote 2007, has been removed from OneNote 2010, so applications that use this class need to be rewritten to use the new interfaces in order to work with OneNote 2010.

Accessing OneNote Data Using the COM API

It's fairly straightforward to start using the OneNote COM API to access live data from OneNote notebooks. Start by creating a new console application in Visual Studio and then add a reference to the Microsoft OneNote 14.0 Type Library COM component (for OneNote 2010) or the Microsoft OneNote 12.0 Type Library COM component (for OneNote 2007).

If you're using Visual Studio 2010 to develop OneNote 2010 applications, take note of a couple minor compatibility issues. First, due to a mismatch of the OneNote interop assembly that shipped with Visual Studio 2010, you should not directly reference the Microsoft.Office.Interop.OneNote component on the .NET tab of the Add Reference dialog, but instead reference the Microsoft OneNote 14.0 Type Library component on the COM tab. This still results in the addition of a OneNote interop assembly to your project's references.

Second, the OneNote 14.0 Type Library is not compatible with the Visual Studio 2010 "NOPIA" feature (in which primary interop assemblies are not embedded in the application by default). Therefore, make sure to set the Embed Interop Types property to False for the OneNote interop assembly reference. (Both of these issues are described in more detail on OneNote Program Manager Daniel Escapa's blog at blogs.msdn.com/descapa/archive/2010/04/27/onenote-2010-and-visual-studio-2010-compatibility-issues.aspx.) With the OneNote library reference in place, you're ready to make calls to the OneNote API. The code in **Figure 1** uses the GetHierarchy method to retrieve an XML document containing a list of OneNote notebooks, then uses LINQ to XML to extract and print the notebook names to the console.

Figure 1 **Enumerating Notebooks**

```
using System;
using System.Linq;
using System.Xml.Linq;
using Microsoft.Office.Interop.OneNote;

class Program
{
  static void Main(string[] args)
  {
    var onenoteApp = new Application();

    string notebookXml;
    onenoteApp.GetHierarchy(null, HierarchyScope.hsNotebooks, out notebookXml);

    var doc = XDocument.Parse(notebookXml);
    var ns = doc.Root.Name.Namespace;
    foreach (var notebookNode in
      from node in doc.Descendants(ns + "Notebook") select node)
    {
      Console.WriteLine(notebookNode.Attribute("name").Value);
    }
  }
}
```

The HierarchyScope enumeration, passed as the second parameter to the GetHierarchy method, specifies the depth of the notebook structure to retrieve. To retrieve sections in addition to the notebooks, simply update this enumeration value to HierarchyScope.hsSections and process the additional XML child nodes, as demonstrated in **Figure 2**.

Retrieving and Updating Page Content

The GetPageContent method will return an XML document containing all of the content on a specified page. The page to retrieve is specified using a OneNote object ID, a string-based

Figure 2 **Enumerating Sections**

```
using System;
using System.Linq;
using System.Xml.Linq;
using Microsoft.Office.Interop.OneNote;

class Program
{
  static void Main(string[] args)
  {
    var onenoteApp = new Application();

    string notebookXml;
    onenoteApp.GetHierarchy(null, HierarchyScope.hsSections, out notebookXml);

    var doc = XDocument.Parse(notebookXml);
    var ns = doc.Root.Name.Namespace;
    foreach (var notebookNode in from node in doc.Descendants(ns +
      "Notebook") select node)
    {
      Console.WriteLine(notebookNode.Attribute("name").Value);
      foreach (var sectionNode in from node in
        notebookNode.Descendants(ns + "Section") select node)
      {
        Console.WriteLine("  " + sectionNode.Attribute("name").Value);
      }
    }
  }
}
```

Figure 3 **Getting Page Content**

```
using System;
using System.Linq;
using System.Xml.Linq;
using Microsoft.Office.Interop.OneNote;

class Program
{
  static void Main(string[] args)
  {
    var onenoteApp = new Application();

    string notebookXml;
    onenoteApp.GetHierarchy(null, HierarchyScope.hsPages, out notebookXml);

    var doc = XDocument.Parse(notebookXml);
    var ns = doc.Root.Name.Namespace;
    var pageNode = doc.Descendants(ns + "Page").Where(n =>
      n.Attribute("name").Value == "Test page").FirstOrDefault();
    if (pageNode != null)
    {
      string pageXml;
      onenoteApp.GetPageContent(pageNode.Attribute("ID").Value, out pageXml);
      Console.WriteLine(XDocument.Parse(pageXml));
    }
  }
}
```

unique identifier for each object in the OneNote notebook hierarchy. This object ID is included as an attribute on the XML nodes returned by the GetHierarchy method.

Figure 3 builds on the previous examples by using the GetHierarchy method to retrieve the OneNote notebook hierarchy down to page scope. It then uses LINQ to XML to select the node for the page named “Test page” and pass that page's object ID to the GetPageContent method. The XML document representing the page content is then printed to the console.

The UpdatePageContent method can be used to make changes to a page. The page content is specified by the same XML document schema that the code in **Figure 3** retrieved; it can contain various content elements that define text outlines, inserted files, images, ink, and audio or video files.

The UpdatePageContent method treats the elements in the provided XML document as a collection of content that may have changed, matching specified content to existing content via its OneNote object ID. You can therefore make changes to existing content by calling the GetPageContent method, making the desired changes to the XML returned, then passing that XML back to the UpdatePageContent method. You can also specify new content elements to be added to the page.

To illustrate this, **Figure 4** adds a date stamp to the bottom of our test page. It uses the approach shown in **Figure 3** to determine the OneNote object ID of the page, and then uses the XDocument and XElement classes in System.Xml.Linq to construct an XML document containing the new content. Because the Page object ID specified in the document matches the object ID of an existing page, the UpdatePageContent method will append the new content to the existing page.

The OneNote Object Model Library

It isn't particularly difficult to interact with OneNote data in this way, but it's a bit awkward to parse and construct XML documents just to perform basic data operations. That's where the OneNote Object Model comes in. It's a managed code library that provides object-oriented abstractions over the COM-based OneNote API. The library is open source and licensed under the Microsoft Public License (Ms-PL).

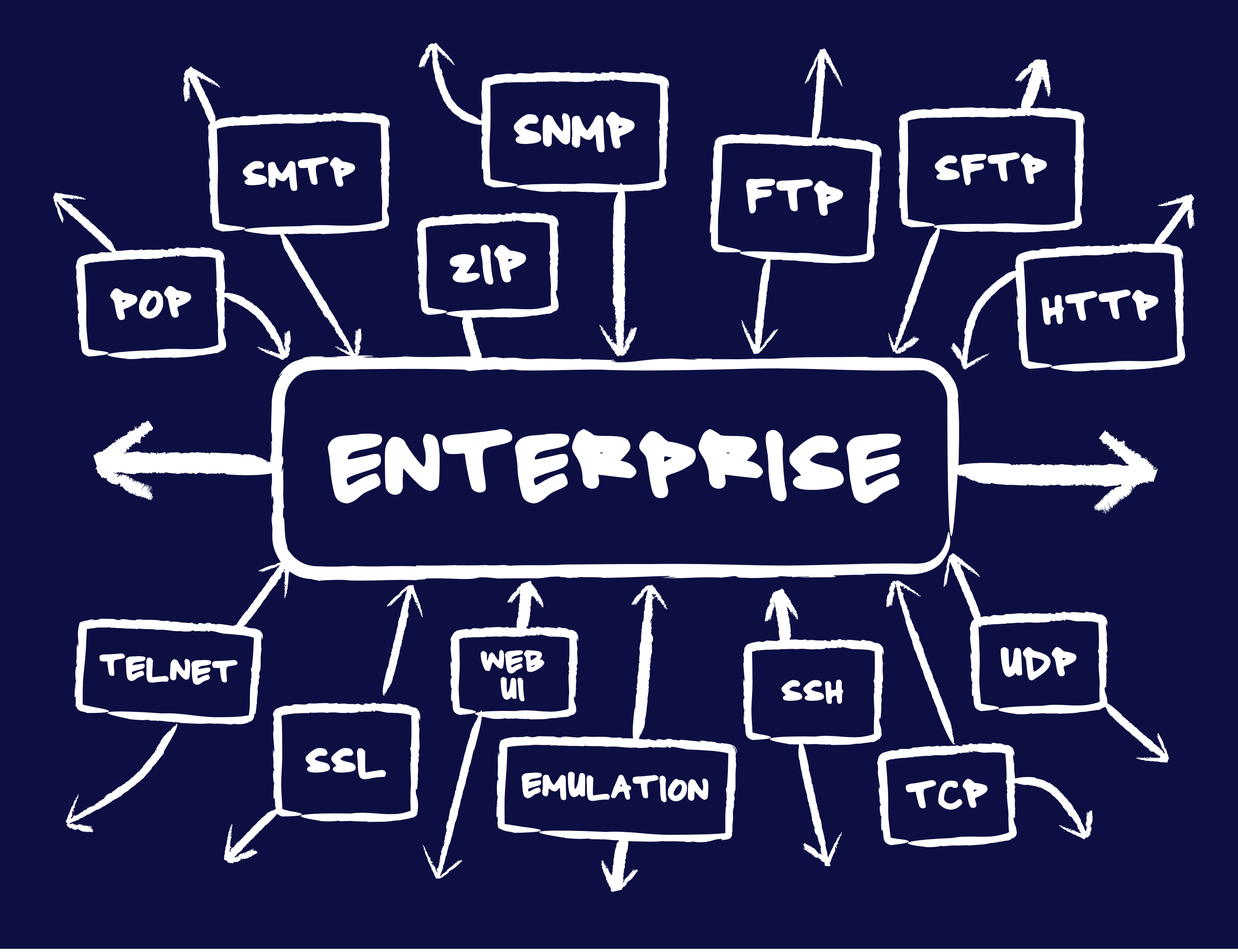
The OneNote Object Model is available for download on CodePlex at onom.codeplex.com. The library was designed for OneNote 2007, and by the time you read this, the release downloads should be updated to provide compatibility with OneNote 2010. If not, you can still use it with OneNote 2007 sections in OneNote 2010 by downloading the source code, removing the existing Microsoft.Office.Interop.OneNote assembly reference in the OneNoteCore project and adding a reference to the Microsoft OneNote 14.0 Type Library as shown previously.

In addition to some unit test projects and sample code, the solution contains two class library projects: OneNoteCore and OneNoteFramework. The OneNoteCore library is the low-level bridge between the OneNote COM API and familiar Microsoft .NET Framework metaphors; it exposes real return values instead of COM *out* parameters, converts COM error codes into .NET exceptions, exposes a OneNoteObjectId struct and XDocument instances instead of raw strings, and more. Studying this code can help you understand how the OneNote API works, but in most cases you won't need to interact with the OneNoteCore library directly.

The OneNoteFramework library provides higher-level abstractions of OneNote concepts. Here you'll find classes with intuitive names like OneNoteNotebook, OneNoteSection and OneNotePage. The primary entry point for interacting with the OneNote hierarchy structure is a class called OneNoteHierarchy, which contains a static member called Current. By adding an

Figure 4 **Updating Page Content**

```
using System;
using System.Linq;
using System.Xml.Linq;
using Microsoft.Office.Interop.OneNote;

class Program
{
  static void Main(string[] args)
  {
    var onenoteApp = new Application();

    string notebookXml;
    onenoteApp.GetHierarchy(null, HierarchyScope.hsPages, out notebookXml);

    var doc = XDocument.Parse(notebookXml);
    var ns = doc.Root.Name.Namespace;
    var pageNode = doc.Descendants(ns + "Page").Where(n =>
      n.Attribute("name").Value == "Test page").FirstOrDefault();
    var existingPageId = pageNode.Attribute("ID").Value;

    if (pageNode != null)
    {
      var page = new XDocument(new XElement(ns + "Page",
                                 new XElement(ns + "Outline",
                                   new XElement(ns + "OEChildren",
                                     new XElement(ns + "OE",
                                       new XElement(ns + "T",
                                         new XCData("Current date: " +
                                           DateTime.Now.
                                             ToLongDateString())))))));
       page.Root.SetAttributeValue("ID", existingPageId);
       onenoteApp.UpdatePageContent(page.ToString(), DateTime.MinValue);
    }
  }
}
```



Internet Connectivity for the Enterprise

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

PowerSNMP for ActiveX and .NET

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

PowerWEB for ASP.NET

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

PowerTCP for ActiveX and .NET

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

SSH
UDP
TCP
SSL
FTP
SFTP
HTTP
POP
SMTP
IMAP
S/MIME
Ping
DNS
Rlogin
Rsh
Rexec
Telnet
VT Emulation
ZIP Compression
more...

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

Download a fully functional product trial today!

DART.com

Figure 5 **Data Binding with Windows Presentation Foundation**

```
<Window x:Class="NotebookTree.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:onf="clr-namespace:Microsoft.Office.OneNote;assembly=
          OneNoteFramework"
        Title="OneNote Notebook Hierarchy" >
  <Grid>
    <Grid.Resources>
      <DataTemplate x:Key="PageTemplate">
        <StackPanel Orientation="Horizontal">
          <Image Source="Images\Page16.png" Margin="0,0,2,0"/>
          <TextBlock Text="{Binding Name}" />
        </StackPanel>
      </DataTemplate>

      <HierarchicalDataTemplate x:Key="SectionTemplate"
        ItemsSource="{Binding Pages}"
        ItemTemplate="{StaticResource PageTemplate}">
        <StackPanel Orientation="Horizontal">
          <Image Source="Images\Section16.png" Margin="0,0,2,0"/>
          <TextBlock Text="{Binding Name}" />
        </StackPanel>
      </HierarchicalDataTemplate>

      <HierarchicalDataTemplate x:Key="NotebookTemplate"
        ItemsSource="{Binding Sections}"
        ItemTemplate="{StaticResource SectionTemplate}">
        <StackPanel Orientation="Horizontal">
          <Image Source="Images\Book16.png" Margin="0,0,2,0"/>
          <TextBlock Text="{Binding Name}" />
        </StackPanel>
      </HierarchicalDataTemplate>
    </Grid.Resources>

    <TreeView Name="NotebookTree" BorderThickness="0"
              HorizontalAlignment="Left" VerticalAlignment="Top"
              ItemsSource="{Binding Notebooks}"
              ItemTemplate="{StaticResource NotebookTemplate}"
              DataContext="{Binding Source={x:Static
                onf:OneNoteHierarchy.Current}}" />
  </Grid>
</Window>
```

assembly reference to the OneNoteFramework library, we can rewrite to our program to enumerate the notebook names (**Figure 1**) much more concisely as follows:

```
using Microsoft.Office.OneNote;

class Program
{
  static void Main(string[] args)
  {
    foreach (var notebook in OneNoteHierarchy.Current.Notebooks)
      System.Console.WriteLine(notebook.Name);
  }
}
```

As you might expect, the OneNoteNotebook class has a property called Sections. Therefore, you can enumerate the section names (**Figure 2**) simply as follows:

```
using Microsoft.Office.OneNote;

class Program
{
  static void Main(string[] args)
  {
    foreach (var notebook in OneNoteHierarchy.Current.Notebooks)
    {
      System.Console.WriteLine(notebook.Name);
      foreach (var section in notebook.Sections)
      {
        System.Console.WriteLine("  " + section.Name);
      }
    }
  }
}
```

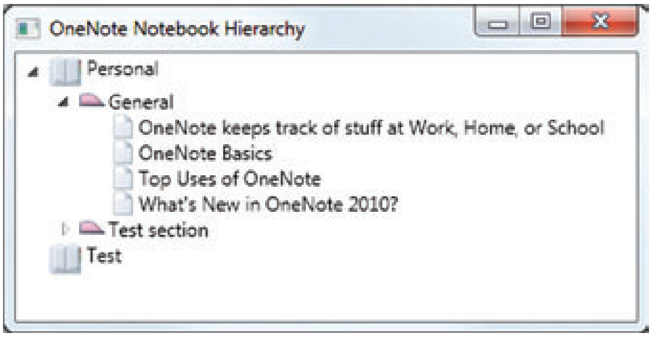

OneNote Notebook Hierarchy
Personal
General
OneNote keeps track of stuff at Work, Home, or School
OneNote Basics
Top Uses of OneNote
What's New in OneNote 2010?
Test section
Test

Figure 6 **Data Binding the Hierarchy to a Tree View**

Collections exposed by OneNote Object Model properties are managed with a specialized generic collection class called OneNoteObjectCollection<T>. Because OneNoteObjectCollection<T> implements IList<T>, as well as IEnumerable<T>, these collections can be queried using LINQ.

For example, given a reference to a OneNoteSection instance in the section variable, we could determine all of the pages that had been modified today with a simple LINQ expression like this:

```
var pagesModifiedToday = from page in section.Pages
                         where page.LastModifiedTime >= DateTime.Today
                         select page;
```

Data Binding with OneNote Object Model Library

The fact that the OneNote Object Model exposes IEnumerable collections also enables XAML-based data binding with Windows Presentation Foundation (WPF). **Figure 5** d demonstrates the use of data binding to display a WPF TreeView of the OneNote notebook hierarchy purely in XAML markup—without requiring the use of code behind.

This XAML first references the OneNoteFramework assembly, giving it the XML namespace prefix *onf*. With this reference in place, the DataContext for the TreeView can then be set to the static Current property of the OneNoteHierarchy class, providing the control with the root of the OneNote hierarchy structure. HierarchicalDataTemplates are then used to data bind each level of the tree with the corresponding collection exposed by the OneNote Object Model (see **Figure 6**).

Simplified Data Access

Wrapping up, the OneNote Object Model library substantially simplifies access to data in Microsoft OneNote notebooks, exposing rich object collections that can be queried and manipulated with LINQ expressions and WPF data binding. A follow-up article will extend these concepts to explore working with OneNote notebooks in Silverlight and Windows Phone applications, and accessing OneNote data in the cloud. ■

ANDY GRAY *is a partner and technology director of Five Talent Software, helping nonprofit organizations operate more effectively through strategic technology solutions. He writes about OneNote development at onenotedev.com.*

THANKS to the following technical experts for reviewing this article: *Michael Gerfen and John Guin*

Does your Team do more than just track bugs?

Free Trial and Single User FreePack™ available at www.alexcorp.com

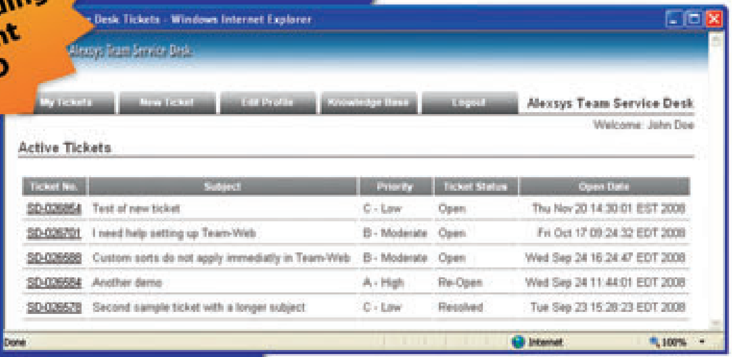
Alexsys Team® does! Alexsys Team 2 is a multi-user Team management system that provides a powerful yet easy way to manage all the members of your team and their tasks - including defect tracking. Use Team right out of the box or tailor it to your needs.

Alexsys Team

Track all your project tasks in one database so you can work together to get projects done.

- Quality Control / Compliance Tracking
- Project Management
- End User Accessible Service Desk Portal
- Bugs and Features
- Action Items
- Sales and Marketing
- Help Desk

Native Smart Card Login Support including Government and DOD



Alexsys Team Service Desk

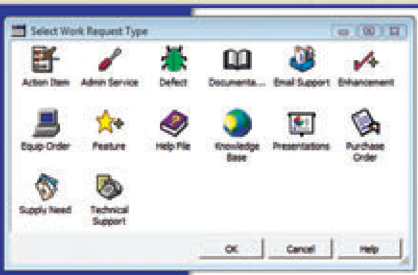
Welcome: John Doe

Active Tickets

Ticket No.	Subject	Priority	Ticket Status	Open Date
SD-026854	Test of new ticket	C - Low	Open	Thu Nov 20 14:30:01 EST 2008
SD-026701	I need help setting up Team-Web	B - Moderate	Open	Fri Oct 17 09:24:32 EDT 2008
SD-026588	Custom sorts do not apply immediatly in Team-Web	B - Moderate	Open	Wed Sep 24 16:24:47 EDT 2008
SD-026584	Another demo	A - High	Re-Open	Wed Sep 24 11:44:01 EDT 2008
SD-026578	Second sample ticket with a longer subject	C - Low	Resolved	Tue Sep 23 15:28:23 EDT 2008

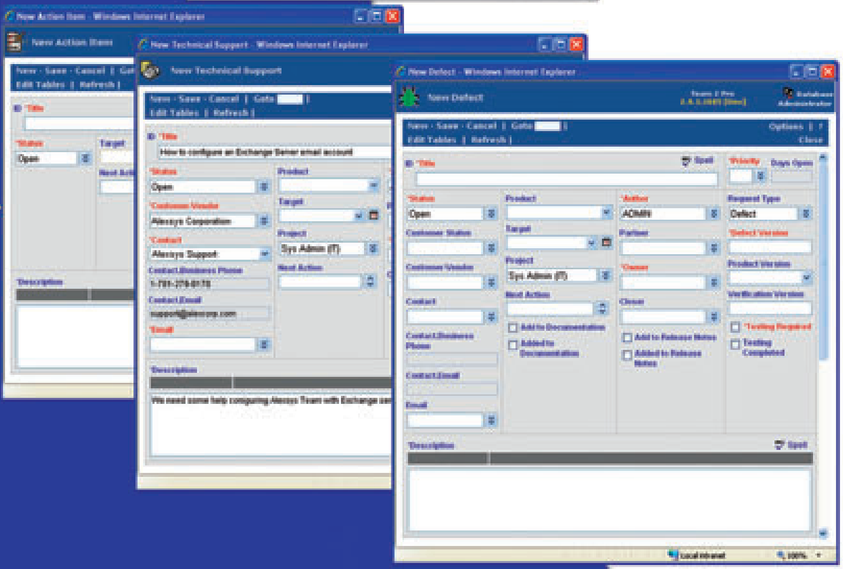
New in Team 2.11

- Full Windows 7 Support
- Windows Single Sign-on
- System Audit Log
- Trend Analysis
- Alternate Display Fields for Data Normalization
- Lookup Table Filters
- XML Export
- Network Optimized for Enterprise Deployment



Service Desk Features

- Fully Secure
- Unlimited Users Self Registered or Active Directory
- Integrated into Your Web Site
- Fast/AJAX Dynamic Content
- Unlimited Service Desks
- Visual Service Desk Builder



How to configure an Exchange Server email account

Open

Alexsys Corporation

Alexsys Support

1-781-279-0170

support@alexcorp.com

Sys Admin (IT)

We need some help conguring Alexsys Team with Exchange ser

Open

ADMIN

Defect

Sys Admin (IT)

Team 2 Features

ZDNet Editors' Pick

- 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.

Merging Word Documents on the Server Side with SharePoint 2010

Ankush Bhatia and Manvir Singh

Business application developers must often create solutions that automate day-to-day activities for their organizations. These activities typically involve processing and manipulating data in various documents—for example, extracting and consolidating data from multiple source documents, merging data into e-mail messages, searching and replacing content in documents, recalculating data in workbooks, extracting images from presentations ... and the list goes on and on.

Microsoft Office makes these kinds of repetitive tasks simpler by providing a rich API that developers can use to automate them. Because such solutions work seamlessly for normal desktop users, developers have taken them to the next level: deploying the solutions to servers that provide a central point where all of this repetitive work can be addressed for multiple users without any human intervention.

Although moving solutions that complete repetitive Office tasks from the desktop to a server seems straightforward, it's not quite as simple as it sounds.

Microsoft designed the Office application suite for desktop computer scenarios where a user is logged on to a machine and is sitting in front of it. For reasons of security, performance and reliability, Office applications are not the right tools for server-side scenarios. Office applications in a server environment may require manual intervention, and that's not optimal for a server-side solution. Microsoft recommends avoiding this kind of solution, as explained in the Microsoft Support article, "Considerations for server-side Automation of Office" (support.microsoft.com/kb/257757).

Since the release of Office 2007, however, the Office automation story has changed a great deal. With Office 2007 Microsoft introduced Office OpenXML and Excel Services for developers who would like to develop Office-based solutions on the server.

With Office 2010 and SharePoint 2010, Microsoft has come up with a new set of components called Application Services. These put a rich set of tools in a developer's bag for Office automation solutions. Application Services include Excel Services, Word Automation Services, InfoPath Forms Services, PerformancePoint Services and Visio Services. You can learn more about the details of these services at msdn.microsoft.com/library/ee559367(v=office.14).

This article discusses:

- The status report template
- Creating a SharePoint document library
- Building the Web Part
- Merging the reports

Technologies discussed:

Office 2010, SharePoint 2010

Code download available at:

code.msdn.microsoft.com/mag201007DocMerge

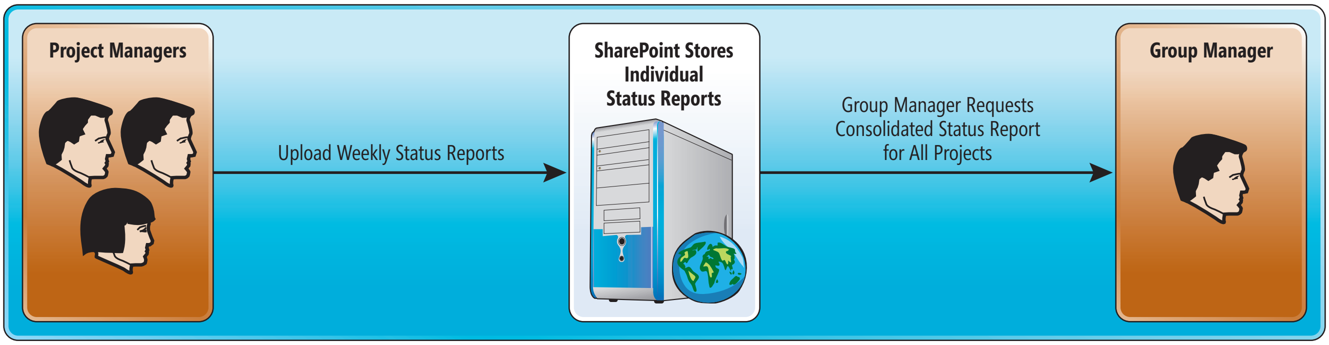

Project Managers
Upload Weekly Status Reports
SharePoint Stores Individual Status Reports
Group Manager Requests Consolidated Status Report for All Projects
Group Manager

Figure 1 **Workflow for Generating a Status Report**

In this article, we will show you how to use Office OpenXML, Word Automation Services and SharePoint to build a simple application that merges separate status reports into a single document.

Status Report Workflow

Let's say you're a developer working at a services-oriented company in which many projects are managed by different teams. Every week, each project manager uses a common template to create a weekly status report and upload it to an internal SharePoint repository. Now your Group Manger wants to get a consolidated report that will contain all of these of weekly status reports and, guess what, you are the chosen one who has to implement this requirement.

You're lucky, though. As we mentioned earlier, your life is easier today because you can implement this requirement with much less effort using OpenXML and Word Automation Services. You'll be able to produce a more robust and stable solution than you could have without these technologies.

Let's start by visualizing the solution. **Figure 1** shows a proposed workflow. The process kicks off with individual project managers filling out status reports and uploading them to SharePoint on the server. The Group Manager can then initiate the process of merging any reports stored on the server and generating a combined report.

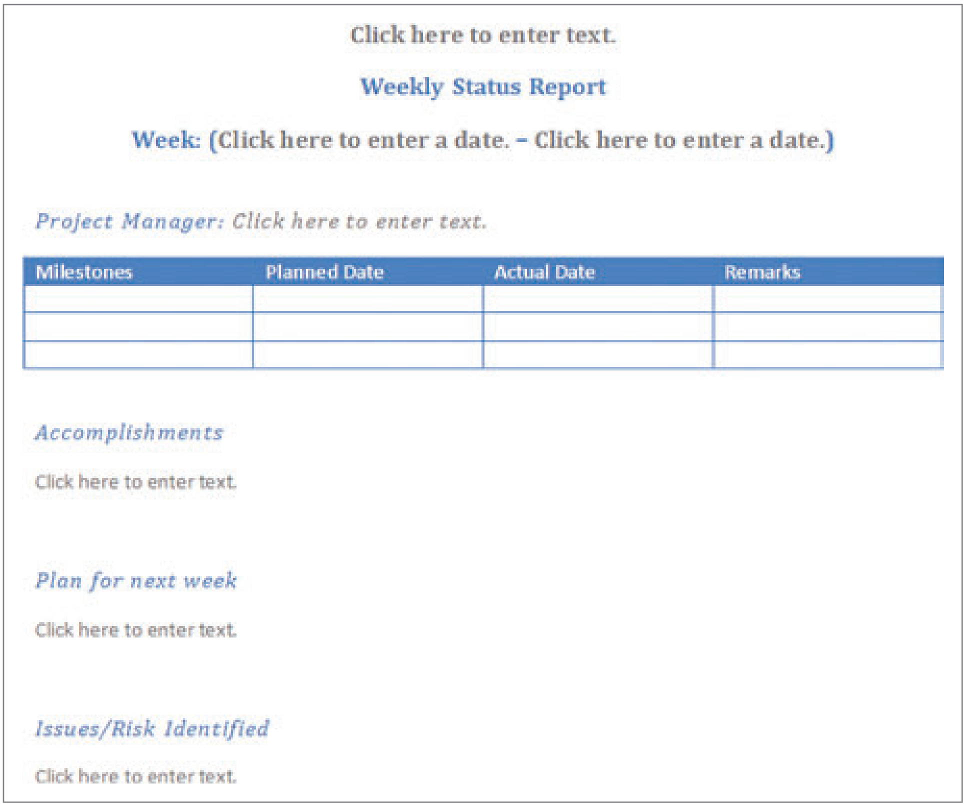
Building a Template

To implement this solution, the first step is to provide a common template to all the project managers for filling out the weekly status reports. When they finish filling in the data, they'll upload the reports to a SharePoint repository. On Monday morning, the Group Manager can then log into the SharePoint site and fire up the logic that performs the following tasks:

1. Reads all of the individual status report documents.
2. Merges them into a single report.
3. Saves the report in the repository for users to access.

Microsoft originally designed the Office application suite for desktop computer scenarios.

Figure 2 shows what the status report template will look like (let's call it WeeklyStatusReport.dotx). As you can see, the template includes fields to capture a title, dates, the project manager's name,



Click here to enter text.

Weekly Status Report

Week: (Click here to enter a date. – Click here to enter a date.)

Project Manager: Click here to enter text.

Milestones	Planned Date	Actual Date	Remarks

Accomplishments

Click here to enter text.

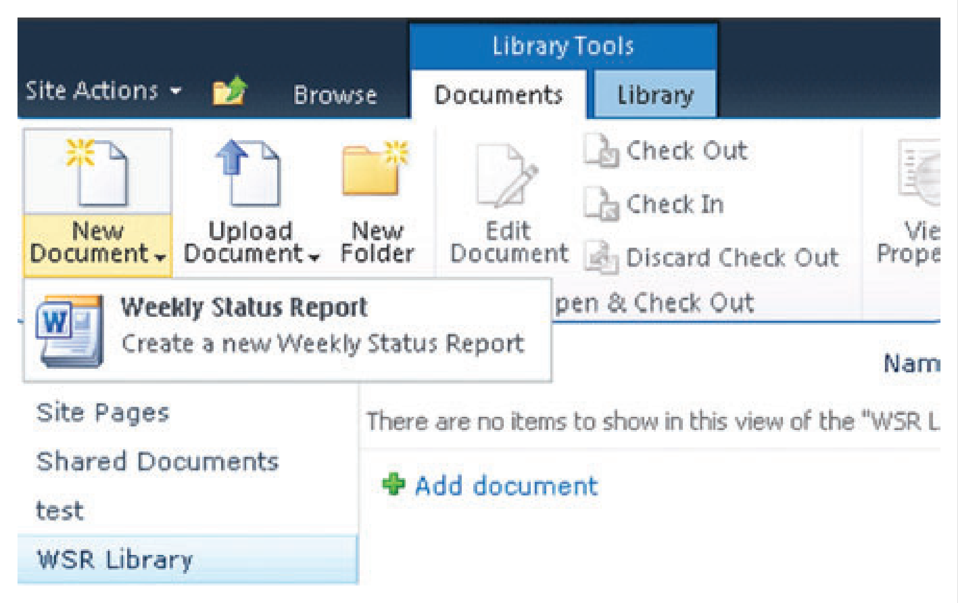
Plan for next week

Click here to enter text.

Issues/Risk Identified

Click here to enter text.

Figure 2 **Weekly Status Report Template**



There are no items to show in this view of the "WSR L

Figure 3 **Selecting the Custom Content Type**

milestones and associated data, and text fields for entering details about accomplishments, future plans and problems. In this case we've used text fields and the date picker control for simplicity, but you could easily use drop-down lists, check boxes or a variety of other controls to streamline data entry.

The Document Library

The next step is to create a custom document library that hosts the weekly status reports based on this template.

In the SharePoint navigation pane, click Libraries and then Create to create a new library. In the Create dialog, filter by Library, select Document Library and type a name for the library (we used WSR Library). Now click Create.

At this point you have the logic in place to generate fully functional consolidated documents on the server.

Now you need to create a content type for the new library. Click Site Actions, then Site Settings, and under the Galleries section, click Site content types. Click Create and then type a name for the content type (we used Weekly Status Report).

In the Select Parent Content Type From list, select Document Content Types. In the Parent Content type list, select Document and click OK.

Under Settings, select Advanced Settings, then choose the "Upload a new document template" radio button and click Browse. Find the report template (WeeklyStatusReport.dotx) and upload it to the library.

Next, go to WSR Library and select Library Settings. Under General Settings, select Advanced Settings. Select Yes for "Allow management of content types," then click OK.

You'll see a list of Content types shown on the library setting page. Select the "Add from Existing Site Content Types" link. Select the content type you created earlier in the available site content types list. In my example, this is Weekly Status Report. Click Add, and click OK.

Again from the content types list, click on Document and select "Delete this content type." Select OK in the warning message box.

Now you should see your content type when you select New Document in your WSR Library, as shown in **Figure 3**.

At this point you can go ahead and add a couple of status reports to the document library.

Creating the Web Part

Next, you need to enable a Group Manager to kick off the consolidation logic. You can do this via a button at the bottom of the default view of the document library.

There are two steps involved here. First, you'll create a Visual Web Part using Visual Studio 2010. Second, you'll add the Web Part to the document library using SharePoint Designer 2010.

To create a custom Web Part, start a new project in Visual Studio 2010 using the Visual Web Part project template. Give the project a name such as DocumentMerge, then click OK.

In the SharePoint Customization Wizard page, select your Web application (the URL to the SharePoint site hosting your document library), then click Finish.

Once the project is created, open the VisualWebPart1.cs file and modify the CreateChildControls method with the following code:

```
protected override void CreateChildControls() {
  Control control = Page.LoadControl(_ascxPath);
  Controls.Add(control);
  base.CreateChildControls();
  Button btnSubmit = new Button();
  btnSubmit.Text = "Merge Reports";
  btnSubmit.Click += new EventHandler(OnSubmitClick);
  Controls.Add(btnSubmit);
}
```

Also add an event handler for the button click:

```
void OnSubmitClick(object sender, EventArgs e) {
  // TODO : Put code to merge documents here
}
```

At this point you can build and deploy your project. We will add the implementation to our OnSubmitClick handler a bit later in this article.

The next step is to add the Web Part to the document library. In SharePoint Designer 2010, open the SharePoint site. Click All Files | WSR Library | Forms, then click on AllItems.aspx to edit it.

INSPIRE! EMPOWER! IMPRESS!

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

Infragistics

KILLER APPS. NO EXCUSES.

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

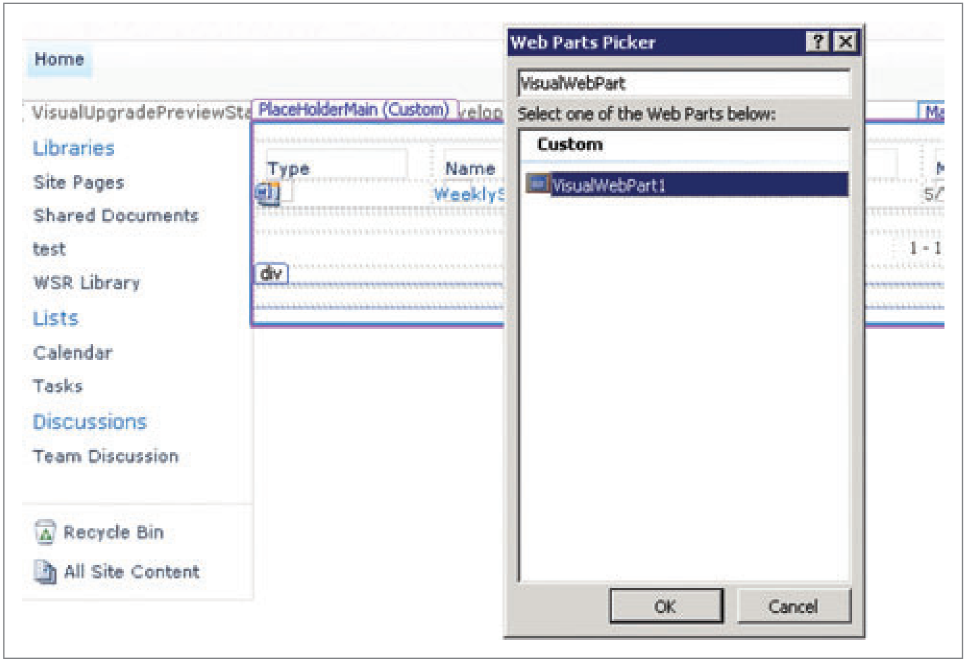


Figure 4 **Inserting the Web Part**

Click the bottom of the page. Click Insert | Web Part, and then select More Web Parts. In the search box, type VisualWebPart (the name of the Web Part you just created and deployed), and click OK (see **Figure 4**). **Figure 5** shows the page with the Web Part in place. Save the page and close SharePoint Designer.

The altChunks get replaced with original content when a document is opened in Word.

Merging the Reports

Now, let's add the logic to merge the uploaded documents in the document library. For simplicity, this code will merge all the documents uploaded to this folder into a single file. A more realistic approach would be to merge only selected items or only items uploaded in a specified time period. You could also save the merged document to a different location or different library. This is when we'll add the implementation to our OnSubmitClick handler of our VisualWebPart project in Visual Studio 2010.

In the OnSubmitClick handler of the Web Part, you need to provide logic for reading the reports that were uploaded to the document library, generating an empty OpenXML document, and merging the reports into the new document.

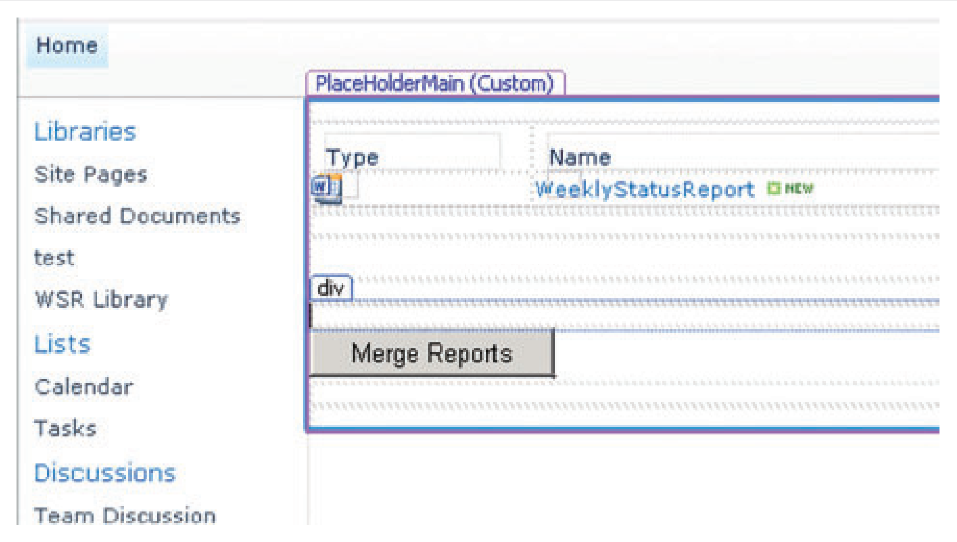


Figure 5 **The Web Part in Place on the Page**

First, you need to read any documents in the current library. You can loop through the SPListItemCollection of the current SPContext, reading each file into a byte array using the SPFile.OpenBinary API:

```
SPListItemCollection files =
  SPContext.Current.List.Items;
foreach (SPListItem item in files) {
  SPFile inputFile = item.File;
  byte[] byteArray =
    inputFile.OpenBinary();

  // process each byte array
}
```

Next, generate the empty OpenXML document. This requires generating the document in memory using a MemoryStream because the OpenXML SDK does not let you save documents to a URI. Instead, the MemoryStream object can dump the document into the library as a new file. The code for creating the file is shown in **Figure 6**.

Note that you need to add DocumentFormat.OpenXml.dll and WindowsBase.dll in the references and the corresponding using statements to the code:

```
using DocumentFormat.OpenXml.Packaging;
using DocumentFormat.OpenXml.Wordprocessing;
```

The next step is to implement the logic for saving the merged document to the library as a new document. This requires a bit of effort, but you can make it easier by using the SharePoint Managed Client Object Model. You'll need to add two references to the

Figure 6 **Creating a New File for the Merged Report**

```
// String containing the blank document part for our new DOCX
string strEmptyMainPart =
  "<?xml version='1.0' encoding='UTF-8' standalone='yes'?>" +
  "<w:document xmlns:w='http://schemas.openxmlformats.org/
wordprocessingml/2006/main'>" +
  "<w:body><w:p><w:r><w:t></w:t></w:r></w:p></w:body></w:document>";

// In-memory stream for our consolidated DOCX.
MemoryStream memOut = new MemoryStream();

// Output document's OpenXML object
WordprocessingDocument outputDoc =
  WordprocessingDocument.Create(memOut,
  DocumentFormat.OpenXml.WordprocessingDocumentType.Document);

MainDocumentPart mainPart = outputDoc.AddMainDocumentPart();

Stream partStream = mainPart.GetStream();
UTF8Encoding encoder = new UTF8Encoding();

// Add blank main part string to the newly created document
Byte[] buffer = encoder.GetBytes(strEmptyMainPart);
partStream.Write(buffer, 0, buffer.Length);

// Save the document in memory
mainPart.Document.Save();
```

project, Microsoft.SharePoint.Client.dll and Microsoft.SharePoint.Client.Runtime.dll, which are found in the following folder:

```
%ProgramFiles%\Common Files\Microsoft Shared\web server extensions\14\ISAPI
```

Create a new document in the SharePoint library with this code:

```
ClientContext clientContext =
  new ClientContext(SPContext.Current.Site.Url);
ClientOM.File.SaveBinaryDirect(clientContext,
  outputPath, memOut, true);
```

For these instructions to work, you'll need the following using statements in the source file:

```
using Microsoft.SharePoint.Client;
using ClientOM = Microsoft.SharePoint.Client;
```

Making the Document Searchable

At this point you have the logic in place to generate fully functional consolidated documents on the server when a user clicks the Merge Reports button.

However, there's one small catch: the generated document is not compatible with the SharePoint crawling mechanism because it contains OpenXML altChunk markup. This is a by-product of merging the reports into the blank document using the code we showed you earlier. The altChunks get replaced with original content when a document is opened in Word.

Figure 7 **Converting altChunks in the Merged Document**

```
string docPath = string.Format(@"{0}{1}",
  SPContext.Current.Site.Url.Replace(@"\\", ""),
  outputPath);

ConversionJobSettings JobSettings =
  new ConversionJobSettings();
JobSettings.OutputFormat = SaveFormat.Document;
JobSettings.OutputSaveBehavior =
  SaveBehavior.AlwaysOverwrite;

ConversionJob ConvJob = new ConversionJob(
  "Word Automation Services", JobSettings);
ConvJob.UserToken = SPContext.Current.Site.UserToken;
ConvJob.AddFile(docPath, docPath);
ConvJob.Start();
```

With the new Word Automation Services in SharePoint 2010, this task can be performed programmatically using ConversionJob class. This class is part of the Microsoft.Office.Word.Server.dll assembly, so add the reference to this assembly to the project manually. Once you've added this reference, you can use the code in **Figure 7** to perform conversion of the altChunks.

See the code download for this article for additional details of the solution, which you can use as the basis of your own reporting system.

Final Steps

In order to test this code, we modified our SharePoint server's configuration to run the Automation Service after one minute of getting a run request. By default, this interval is set to five minutes, and we didn't want to wait that long for our conversion to happen.

If you'd like to change this setting, you can set it in SharePoint Central Administration under Application Management | Manage Service Applications | Word Automation Services, and set the Frequency to start conversions under Conversion Throughput to one minute.

The final generated report contains all the weekly status reports you created, merged into a single new document with each of the individual reports stacked one after the other.

And that's it. In a future article we'll take the concept of server-side merging of document contents to the next level. We'll show you how to implement a mail-merge type of scenario on the server side, again using Office 2010, SharePoint 2010 and Visual Studio 2010. Until then, happy coding.

> The final generated report contains all the weekly status reports you created, merged into a single new document with each of the individual reports stacked one after the other.

For more information on Office 2010 and SharePoint 2010, see the Office (msdn.microsoft.com/office) and SharePoint (msdn.microsoft.com/sharepoint) developer centers. Information about Office OpenXML can be found at msdn.microsoft.com/library/bb448854, and you can read about Word Automation Services at msdn.microsoft.com/library/ee558278(v=office.14). ■

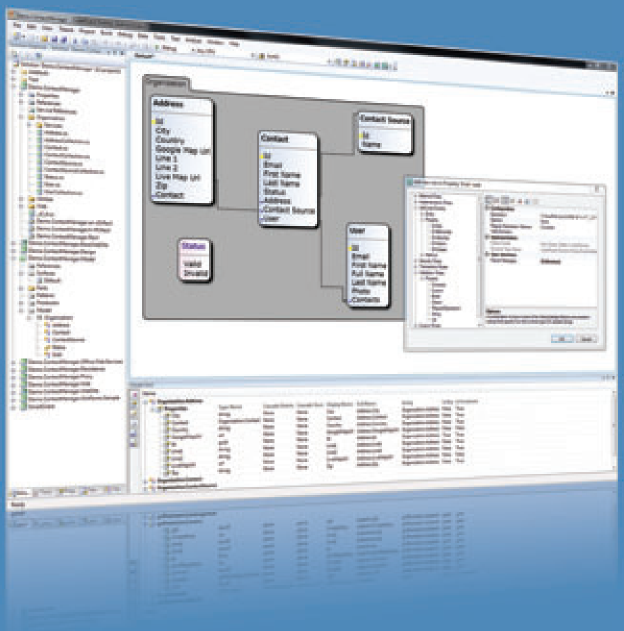
MANVIR SINGH *and* **ANKUSH BHATIA** *are part of the Visual Studio Developer Support Team in Microsoft Product Support Services (PSS), helping customers on programming issues involving Office client applications. You can reach Singh at manvir.singh@microsoft.com or manvirsingh.net. You can reach Bhatia at ankush.bhatia@microsoft.com or abhatia.wordpress.com.*

THANKS to the following technical expert for reviewing this article: *Eric White*

...HOSTED ENVIRONMENT ... ≡BLAH BLAH≡ ... ZERO DEPLOYMENT ... FEATURE...
WOW... HOSTED APPS... IS HE AWARE OF OUR DESKTOP-ONLY ARCHITECTURE?
?
C.E.O.
DEV. LEAD
PROG. MGR.
≡BLAH BLAH≡ LESS ENERGY ... THE ...NEED TO MANAGE HARDWARE ≡BLAH≡ ... RESOURCES...
ERRHH... NOT SURE, IT'S PROBABLY A 3-YEAR PLAN OR SO...
C.E.O.
DEV. LEAD
PROG. MGR.
...WE'RE PLEASED TO ANNOUNCE A CLOUD VERSION IN THE NEXT ...
...FEW WEEKS!
"WE"?
BRAVO!
CLAP!
CLAP!
CLAP!
APPLAUSE!!

Need a ***HAND?***

Use **CODEFLUENT ENTITIES!**



APPLICATION BLOCKS

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

SUPPORTED ARCHITECTURES

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

FEATURED TECHNOLOGIES

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

CUSTOMER APPROVED MODEL-DRIVEN SOFTWARE FACTORY

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

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

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

Download your ***FREE*** *trial today at:*

www.CodeFluentEntities.com/Msdn

CodeFluent Entities
the Model Driven Software factOry

Contact: info@softfluent.com
www.CodeFluentEntities.com

Building Distributed Apps with NHibernate and Rhino Service Bus

Oren Eini

For a long time, I dealt almost exclusively in Web applications. When I moved over to build a smart client application, at first I was at quite a loss as to how to approach building such an application. How do I handle data access? How do I communicate between the smart client application and the server?

Furthermore, I already had a deep investment in an existing toolset that drastically reduced the time and cost for development, and I really wanted to be able to continue using those tools. It took me a while to figure out the details to my satisfaction, and during that time, I kept thinking how much simpler a Web app would be—if only because I knew how to handle such apps already.

There are advantages and disadvantages to smart client applications. On the plus side, smart clients are responsive and promote interactivity with the user. You also reduce server load by moving processing to a client machine, and enable users to work even while disconnected from back-end systems.

On the other hand, there are the challenges inherent in such smart clients, including contending with the speed, security, and bandwidth limitations of data access over the intranet or Internet. You're also responsible for synchronizing data between front-end and back-end systems, distributed change-tracking, and handling the issues of working in an occasionally connected environment.

A smart client application, as discussed in this article, can be built with either Windows Presentation Foundation (WPF) or Silverlight. Because Silverlight exposes a subset of WPF features, the techniques and approaches I outline here are applicable to both.

In this article, I start the processes of planning and building a smart client application using NHibernate for data access and Rhino Service Bus for reliable communication with the server. The application will function as the front end for an online lending library, which I called Alexandra. The application itself is split into two major pieces. First, there's an application server running a set of services (where most of the business logic will reside), accessing the database using NHibernate. Second, the smart client UI will make exposing those services to the user easy.

NHibernate (nhforge.org) is an object-relational mapping (O/RM) framework designed to make it as easy to work with relational databases as it is to work with in-memory data. Rhino Service Bus (github.com/rhino-esb/rhino-esb) is an open source service bus implementation built on the Microsoft .NET Framework, focusing primarily on ease of development, deployment and use.

This article discusses:

- Distribution of responsibilities
- Fallacies of distributed computing
- Queues and disconnected operation
- Session and transaction management

Technologies discussed:

NHibernate, Rhino Service Bus

Distribution of Responsibilities

The first task in building the lending library is to decide on the proper distribution of responsibility between the front-end and back-end systems. One path is to focus the application primarily on the UI so that most of the processing is done on the client machine. In this case the back end serves mostly as a data repository.

In essence, this is just a repetition of the traditional client/server application, with the back end serving as a mere proxy for the data store. This is a valid design choice if the back-end system is just a data repository. A personal book catalog, for example, might benefit from such architecture, because the behavior of the application is limited to managing data for the users, with no manipulation of the data on the server side.

For such applications, I recommend making use of WCF RIA Services or WCF Data Services. If you want the back-end server to expose a CRUD interface for the outside world, then leveraging WCF RIA Services or WCF Data Services allows you to drastically cut down the time required to build the application. But while both technologies let you add your own business logic to the CRUD interface, any attempt to implement significant application behavior using this approach would likely result in an unmaintainable, brittle mess.

I won't cover building such an application in this article, but Brad Adams has shown a step-by-step approach for building just such an application using NHibernate and WCF RIA Services on his blog at blogs.msdn.com/brada/archive/2009/08/06/business-apps-example-for-silverlight-3-rtm-and-net-ria-services-july-update-part-nhibernate.aspx.

Going all the way to the other extreme, you can choose to implement most of the application behavior on the back end, leaving the front end with purely presentation concerns. While this seems reasonable at first, because this is how you typically write Web-based applications, it means that you can't take advantage of running a real application on the client side. State management would be harder. Essentially you're back writing a Web application, with all the complexities this entails. You won't be able to shift processing to the client machine and you won't be able to handle interruptions in connectivity.

Worse, from the user perspective, this approach means that you present a more sluggish UI since all actions require a roundtrip to the server.

I'm sure it won't surprise you that the approach I'm taking in this example is somewhere in the middle. I'm going to take advantage of the possibilities offered by running on the client machine, but at the same time significant parts of the application run as services on the back end, as shown in **Figure 1**.

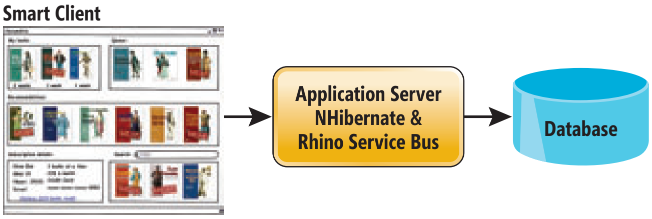


Figure 1 **The Application's Architecture**

The sample solution is composed of three projects, which you can download from github.com/ayende/alexandria. Alexandria.Backend is a console application that hosts the back-end code. Alexandria.Client contains the front-end code, and Alexandria.Messages contains the message definitions shared between them. To run the sample, both Alexandria.Backend and Alexandria.Client need to be running.

One advantage of hosting the back end in a console application is that it allows you to easily simulate disconnected scenarios by simply shutting down the back-end console application and starting it up at a later time.

Fallacies of Distributed Computing

With the architectural basics in hand, let's take a look at the implications of writing a smart client application. Communication with the back end is going to be through an intranet or the Internet. Considering the fact that the main source for remote calls in most Web applications is a database or another application server located in the same datacenter (and often in the same rack), this is a drastic change with several implications.

Intranet and Internet connections suffer from issues of speed, bandwidth limitations and security. The vast difference in the costs of communication dictate a different communication structure than the one you'd adopt if all the major pieces in the application were residing in the same datacenter.

Among the biggest hurdles you have to deal with in distributed applications are the fallacies of distributed computing. These are a set of assumptions that developers tend to make when building distributed applications, which ultimately prove false. Relying on these false assumptions usually results in reduced capabilities or a very high cost to redesign and rebuild the system. There are eight fallacies:

- The network is reliable.
- Latency is zero.
- Bandwidth is infinite.
- The network is secure.
- Topology doesn't change.
- There is one administrator.
- Transport cost is zero.
- The network is homogeneous.

Any distributed application that doesn't take these fallacies into account is going to run into sever problems. A smart client application needs to deal with those issues head on. The use of caching is a topic of great importance in such circumstances. Even if you aren't interested in working in a disconnected fashion, a cache is almost always useful for increasing application responsiveness.

Intranet and Internet connections suffer from issues of speed, bandwidth and security.

Another aspect you need to consider is the communication model for the application. It may seem that the simplest model is a standard service proxy that allows you to perform remote procedure calls (RPCs), but this tends to cause problems down the road. It leads to more-complex code to handle a disconnected state and requires you to explicitly handle asynchronous calls if you want to avoid blocking in the UI thread.

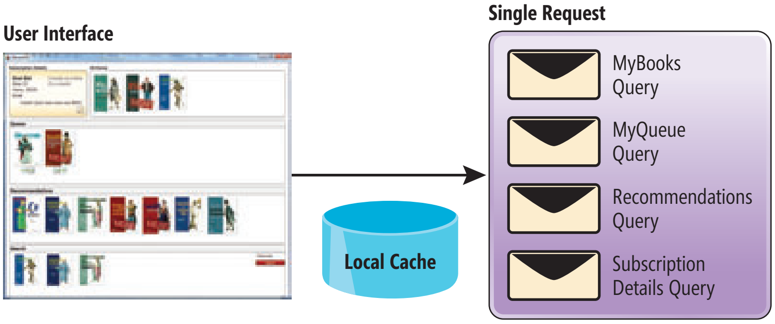


Figure 2 **A Single Request to the Server Contains Several Messages**

Back-End Basics

Next, there's the problem of how to structure the back end of the application in a way that provides both good performance and a degree of separation from the way the UI is structured.

The ideal scenario from a performance and responsiveness perspective is to make a single call to the back end to get all the data you need for the presented screen. The problem with going this route is that you end up with a service interface that mimics the smart client UI exactly. This is bad for a whole host of reasons. Mainly, the UI is the most changeable part in an application. Tying the service interface to the UI in this fashion results in frequent changes to the service, driven by purely UI changes.

> Let's make use of a local cache and a message-oriented communication model.

That, in turn, means deployment of the application just got a lot harder. You have to deploy both the front end and the back end at the same time, and trying to support multiple versions at the same time is likely to result in greater complexity. In addition, the service interface can't be used to build additional UIs or as an integration point for third-party or additional services.

If you try going the other route—building a standard, fine-grained interface—you'll run head on into the fallacies (a fine-grained interface leads to a high number of remote calls, resulting in issues with latency, reliability and bandwidth).

The answer to this challenge is to break away from the common RPC model. Instead of exposing methods to be called remotely, let's use a local cache and a message-oriented communication model.

Figure 2 shows how you pack several requests from the front end to the back end. This allows you to make a single remote call, but keep a programming model on the server side that isn't tightly coupled to the needs of the UI.

To increase responsiveness, you can include a local cache that can answer some queries immediately, leading to a more-responsive application.

One of the things you have to consider in these scenarios is what types of data you have and the freshness requirements for any data you display. In the Alexandria application, I lean heavily on the local cache because it is acceptable to show the user cached data while the application requests fresh data from the back-end system. Other applications—stock trading, for example—should probably show nothing at all rather than stale data.

Disconnected Operations

The next problem you have to face is handling disconnected scenarios. In many applications, you can specify that a connection is mandatory, which means you can simply show the user an error if the back-end servers are unavailable. But one benefit of a smart client application is that it *can* work in a disconnected manner, and the Alexandria application takes full advantage of that.

However, this means the cache becomes even more important because it's used both to speed communication and to serve data from the cache if the back-end system is unreachable.

By now, I believe you have a good understanding of the challenges involved in building such an application, so let's move on to see how to solve those challenges.

Queues Are One of My Favorite Things

In Alexandria, there's no RPC communication between the front end and the back end. Instead, as shown in **Figure 3**, all communication is handled via one-way messages going through queues.

Queues provide a rather elegant way of solving the communication issues identified earlier. Instead of communicating directly between the front end and the back end (which means supporting disconnected scenarios is *hard*), you can let the queuing subsystem handle all of that.

Using queues is quite simple. You ask your local queuing subsystem to send a message to some queue. The queuing subsystem takes ownership of the message and ensures that it reaches its destination at some point. Your application, however, doesn't wait for the message to reach its destination and can carry on doing its work.

If the destination queue is not currently available, the queuing subsystem will wait until the destination queue becomes available again, then deliver the message. The queuing subsystem usually

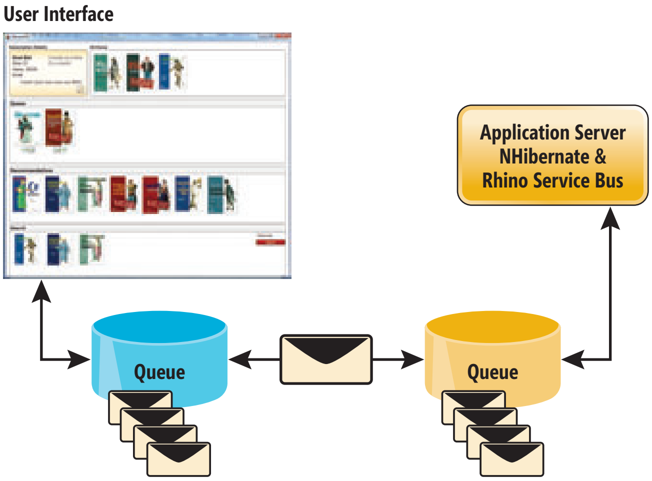


Figure 3 **The Alexandria Communication Model**

persists the message to disk until it's delivered, so pending messages will still arrive at their destination even if the source machine has been restarted.

When using queues, it's easy to think in terms of messages and destinations. A message arriving at a back-end system will trigger some action, which may then result in a reply sent to the original sender. Note that there's no blocking on either end, because each system is completely independent.

Queuing subsystems include MSMQ, ActiveMQ, RabbitMQ, and others. The Alexandria application uses Rhino Queues (github.com/rhino-queues/rhino-queues), an open source, xcopy-deployed queuing subsystem. I chose Rhino Queues for the simple reason that it requires no installation or administration, making it ideal for use in samples and in applications that you need to deploy to many machines. It's also worth noting that I wrote Rhino Queues. I hope you like it.

Putting Queues to Work

Let's see how you can handle getting the data for the main screen using queues. Here's the ApplicationModel initialization routine:

```
protected override void OnInitialize() {
  bus.Send(
    new MyBooksQuery { UserId = userId },
    new MyQueueQuery { UserId = userId },
    new MyRecommendationsQuery { UserId = userId },
    new SubscriptionDetailsQuery { UserId = userId });
}
```

I'm sending a batch of messages to the server, requesting several pieces of information. There are a number of things to notice here. The granularity of the messages sent is high. Rather than sending a single, general message such as MainWindowQuery, I send many messages, (MyBooksQuery, MyQueueQuery, and so on), each for a very specific piece of information. As discussed previously, this allows you to benefit both from sending several messages in a single batch (reducing network roundtrips) and reducing the coupling between the front end and the back end.

RPC Is Thy Enemy

One of the most common mistakes in building a distributed application is to ignore the distribution aspect of the application. WCF, for example, makes it easy to ignore the fact that you're making a method call over the network. While that's a very simple programming model, it means you need to be extremely careful not to violate one of the fallacies of distributed computing.

Indeed, it's the very fact that the programming model offered by frameworks such as WCF is so similar to the one you use for calling methods on the local machine that leads you to make those false assumptions.

A standard RPC API means blocking when making a call over the network, higher cost for each remote method call and the potential for failure if the back-end server is not available. It's certainly possible to build a good distributed application on top of this foundation, but it takes greater care.

Taking a different approach leads you to a programming model based on explicit message exchanges (as opposed to the implicit message exchanges common in most SOAP-based RPC stacks). That model may look strange at first, and it does require you to shift your thinking a bit, but it turns out that by making this shift, you significantly reduce the amount of complexity to worry about overall.

My example Alexandria application is built on top of a one-way messaging platform, and it makes full use of this platform so the application is aware of the fact it's distributed and actually takes advantage of that.

Figure 4 **Consuming a Query on the Back-End System**

```
public class MyBooksQueryConsumer :
  ConsumerOf<MyBooksQuery> {

  private readonly ISession session;
  private readonly IServiceBus bus;

  public MyBooksQueryConsumer(
    ISession session, IServiceBus bus) {

    this.session = session;
    this.bus = bus;
  }

  public void Consume(MyBooksQuery message) {
    var user = session.Get<User>(message.UserId);

    Console.WriteLine("{0}'s has {1} books at home",
      user.Name, user.CurrentlyReading.Count);

    bus.Reply(new MyBooksResponse {
      UserId = message.UserId,
      Timestamp = DateTime.Now,
      Books = user.CurrentlyReading.ToBookDtoArray()
    });
  }
}
```

Note that all of the messages end with the term Query. This is a simple convention I use to denote pure query messages that change no state and expect some sort of response.

Finally, notice that I don't seem to be getting any sort of reply from the server. Because I'm using queues, the mode of communication is fire and forget. I fire off a message (or a batch of messages) now, and I deal with the replies at a later stage.

Before looking at how the front end deals with the responses, let's see how the back end processes the messages I just sent. **Figure 4** shows how the back-end server consumes a query for books. And here, for the first time, you can see how I use both NHibernate and Rhino Service Bus.

But before diving into the actual code that handles this message, let's discuss the structure in which this code is running.

It's All About Messages

Rhino Service Bus (hibernatingrhinos.com/open-source/rhino-service-bus) is, unsurprisingly, a service bus implementation. It's a communication framework based on a one-way queued message exchange, heavily inspired by NServiceBus (nservicebus.com).

A message sent on the bus will arrive at its destination queue, where a message consumer will be invoked. That message consumer in **Figure 4** is MyBooksQueryConsumer. A message consumer is a class that implements ConsumerOf<TMsg>, and the Consume method is invoked with the appropriate message instance to handle the message.

You can probably surmise from the MyBooksQueryConsumer constructor that I'm using an Inversion of Control (IoC) container to supply dependencies for the message consumer. In the case of

Figure 5 **Initializing Messaging Sessions**

```
public class AlexandriaBootStrapper :
  AbstractBootStrapper {

  public AlexandriaBootStrapper() {
    NHibernateProfiler.Initialize();
  }

  protected override void ConfigureContainer() {
    var cfg = new Configuration()
      .Configure("nhibernate.config");
    var sessionFactory = cfg.BuildSessionFactory();

    container.Kernel.AddFacility(
      "factory", new FactorySupportFacility());

    container.Register(
      Component.For<ISessionFactory>()
        .Instance(sessionFactory),
      Component.For<IMessageModule>()
        .ImplementedBy<NHibernateMessageModule>(),
      Component.For<ISession>()
        .UsingFactoryMethod(() =>
          NHibernateMessageModule.CurrentSession)
        .LifeStyle.Is(LifestyleType.Transient));

    base.ConfigureContainer();
  }
}
```

MyBooksQueryConsumer, those dependencies are the bus itself and the NHibernate session.

The actual code for consuming the message is straightforward. You get the appropriate user from the NHibernate session and send a reply back to the message originator with the requested data.

The front end also has a message consumer. This consumer is for MyBooksResponse:

```
public class MyBooksResponseConsumer :
  ConsumerOf<MyBooksResponse> {

  private readonly ApplicationModel applicationModel;

  public MyBooksResponseConsumer(
    ApplicationModel applicationModel) {
    this.applicationModel = applicationModel;
  }

  public void Consume(MyBooksResponse message) {
    applicationModel.MyBooks.UpdateFrom(message.Books);
  }
}
```

> When using queues, it's easy to think in terms of messages and destinations.

This simply updates the application model with the data from the message. One thing, however, should be noted: the consume method is *not* called on the UI thread. Instead, it's called on a background thread. The application model is bound to the UI, however, so updating it *must* happen on the UI thread. The UpdateFrom method is aware of that and will switch to the UI thread to update the application model in the correct thread.

The code for handling the other messages on both the back end and the front end is similar. This communication is purely asynchronous. At no point are you waiting for a reply from the back end, and you aren't using the .NET Framework's asynchronous API. Instead, you have an explicit message exchange, which usually happens almost instantaneously, but can also stretch over a longer time period if you're working in a disconnected mode.

Earlier, when I sent the queries to the back end, I just told the bus to send the messages, but I didn't say where to send them. In **Figure 4**, I just called Reply, again not specifying where the message should be sent. How does the bus know where to send those messages?

In the case of sending messages to the back end, the answer is: configuration. In the App.config, you'll find the following configuration:

```
<messages>
  <add name="Alexandria.Messages"
    endpoint="rhino.queues://localhost:51231/alexandria_backend"/>
</messages>
```

This tells the bus that all messages whose namespace starts with Alexandria.Messages should be sent to the alexandria_backend endpoint.

In the handling of the messages in the back-end system, calling Reply simply means sending the message back to its originator.

This configuration specifies the ownership of a message, that is, to whom to send this message when it's placed on the bus and where to send a subscription request so you'll be included in the distribution list when messages of this type are published. I'm not using message publication in the Alexandria application, so I won't cover that.

Figure 6 **Managing Session Lifetime**

```
public class NHibernateMessageModule : IMessageModule {
  private readonly ISessionFactory sessionFactory;
  [ThreadStatic]
  private static ISession currentSession;

  public static ISession CurrentSession {
    get { return currentSession; }
  }

  public NHibernateMessageModule(
    ISessionFactory sessionFactory) {

    this.sessionFactory = sessionFactory;
  }

  public void Init(ITransport transport,
    IServiceBus serviceBus) {

    transport.MessageArrived += TransportOnMessageArrived;
    transport.MessageProcessingCompleted
      += TransportOnMessageProcessingCompleted;
  }

  private static void
    TransportOnMessageProcessingCompleted(
    CurrentMessageInformation currentMessageInformation,
    Exception exception) {

    if (currentSession != null)
        currentSession.Dispose();
    currentSession = null;
  }

  private bool TransportOnMessageArrived(
    CurrentMessageInformation currentMessageInformation) {

    if (currentSession == null)
        currentSession = sessionFactory.OpenSession();
    return false;
  }
}
```

Imagine...

...an intranet employees want to use

Why is user adoption such a large hurdle for intranets?

eIntranet overcomes this hurdle by transforming the user experience. Employees connect with the right people and content instantly. Information finds them, no matter where they go.

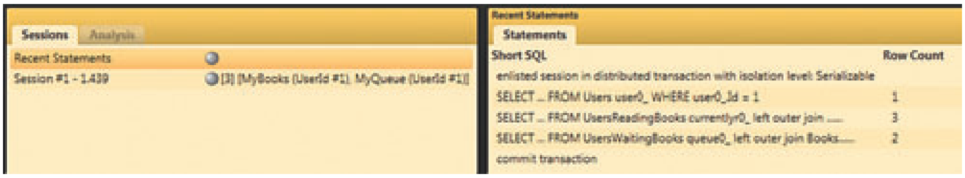
- **Collaboration** – Complete projects faster in collaborative groupspaces with powerful communication and sharing tools
- **Timeline and Social Navigation** – Find content and collateral based on when it was created and who is using it
- **Easy to deploy, customize and extend** – Integrate with business infrastructures and extend the functionality to meet unique needs
- **Mobile engagement** – Engage employees on the go, delivering updates via SMS alerts, e-mail or the eIntranet Mobile App

Learn more:

eIntranet™
Built on Ektron

http://www.ektron.com/intranet

ektron
What do you want your website to do?



Recent Statements
Session #1 - 1.439 [3] [MyBooks (UserId #1), MyQueue (UserId #1)]

Recent Statements
Statements

```
Short SQL                                                                 Row Count
enlisted session in distributed transaction with isolation level: Serializable
SELECT ... FROM Users user0_ WHERE user0_.Id = 1                              1
SELECT ... FROM UsersReadingBooks currentlyr0_ left outer join ....           3
SELECT ... FROM UsersWaitingBooks queue0_ left outer join Books....           2
commit transaction
```

Figure 7 **The NHibnerate Profiler View of Processing Requests**

Session Management

You've seen how the communication mechanism works now, but there are infrastructure concerns to address before moving forward. As in any NHibernate application, you need some way of managing the session lifecycle and handling transactions properly.

The standard approach for Web applications is to create a session per request, so each request has its own session. For a messaging application, the behavior is almost identical. Instead of having a session per request, you have a session per message batch.

It turns out that this is handled almost completely by the infrastructure. **Figure 5** shows the initialization code for the back-end system.

Bootstrapping is an explicit concept in Rhino Service Bus, implemented by classes deriving from AbstractBootStrapper. The bootstrapper has the same job as the Global.asax in a typical Web application. In **Figure 5** I first build the NHibernate session factory, then set up the container (Castle Windsor) to provide the NHibernate session from the NHibenrateMessageModule.

A message module has the same purpose as an HTTP module in a Web application: to handle cross-cutting concerns across all requests. I use the NHibernateMessageModule to manage the session lifetime, as shown in **Figure 6**.

The code is pretty simple: register for the appropriate events, create and dispose of the session in the appropriate places and you're done.

One interesting implication of this approach is that all messages in a batch will share the same session, which means that in many cases you can take advantage of NHibernate's first-level cache.

Transaction Management

That's it for session management, but what about transactions?

A best practice for NHibernate is that all interactions with the database should be handled via transactions. But I'm not using NHibernate's transactions here. Why?

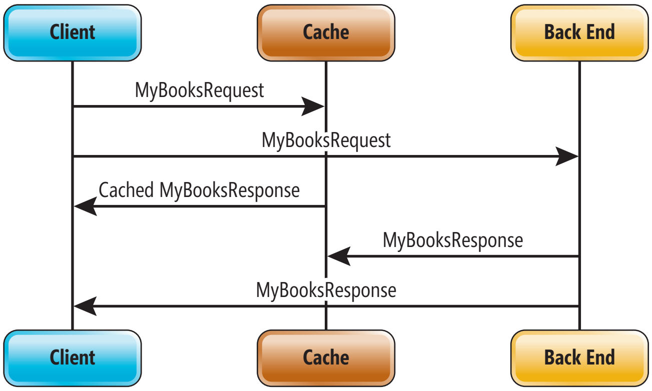


Figure 8 **Using the Cache in Concurrent Messaging Operations**

The answer is that transactions are handled by Rhino Service Bus. Instead of making each consumer manage its own transactions, Rhino Service Bus takes a different approach. It makes use of System.Transactions.TransactionScope to create a single transaction that encompasses all the consumers for messages in the batch.

That means all the actions taken in a response to a *message batch* (as opposed to a single message) are part of the same transaction. NHibernate will automatically enlist a session in the ambient transaction, so when you're using Rhino Service Bus you have no need to explicitly deal with transactions.

The standard approach for Web applications is to create a session per request.

The combination of a single session and a single transaction makes it easy to combine multiple operations into a single transactional unit. It also means you can directly benefit from NHibernate's first-level cache. For example, here's the relevant code to handle MyQueueQuery:

```
public void Consume(MyQueueQuery message) {
  var user = session.Get<User>(message.UserId);

  Console.WriteLine("{0}'s has {1} books queued for reading",
    user.Name, user.Queue.Count);

  bus.Reply(new MyQueueResponse {
    UserId = message.UserId,
    Timestamp = DateTime.Now,
    Queue = user.Queue.ToBookDtoArray()
  });
}
```

The actual code for handling a MyQueueQuery and MyBooksQuery is nearly identical. So, what's the performance implication of a single transaction per session for the following code?

```
bus.Send(
  new MyBooksQuery {
    UserId = userId
  },
  new MyQueueQuery {
    UserId = userId
  });
```

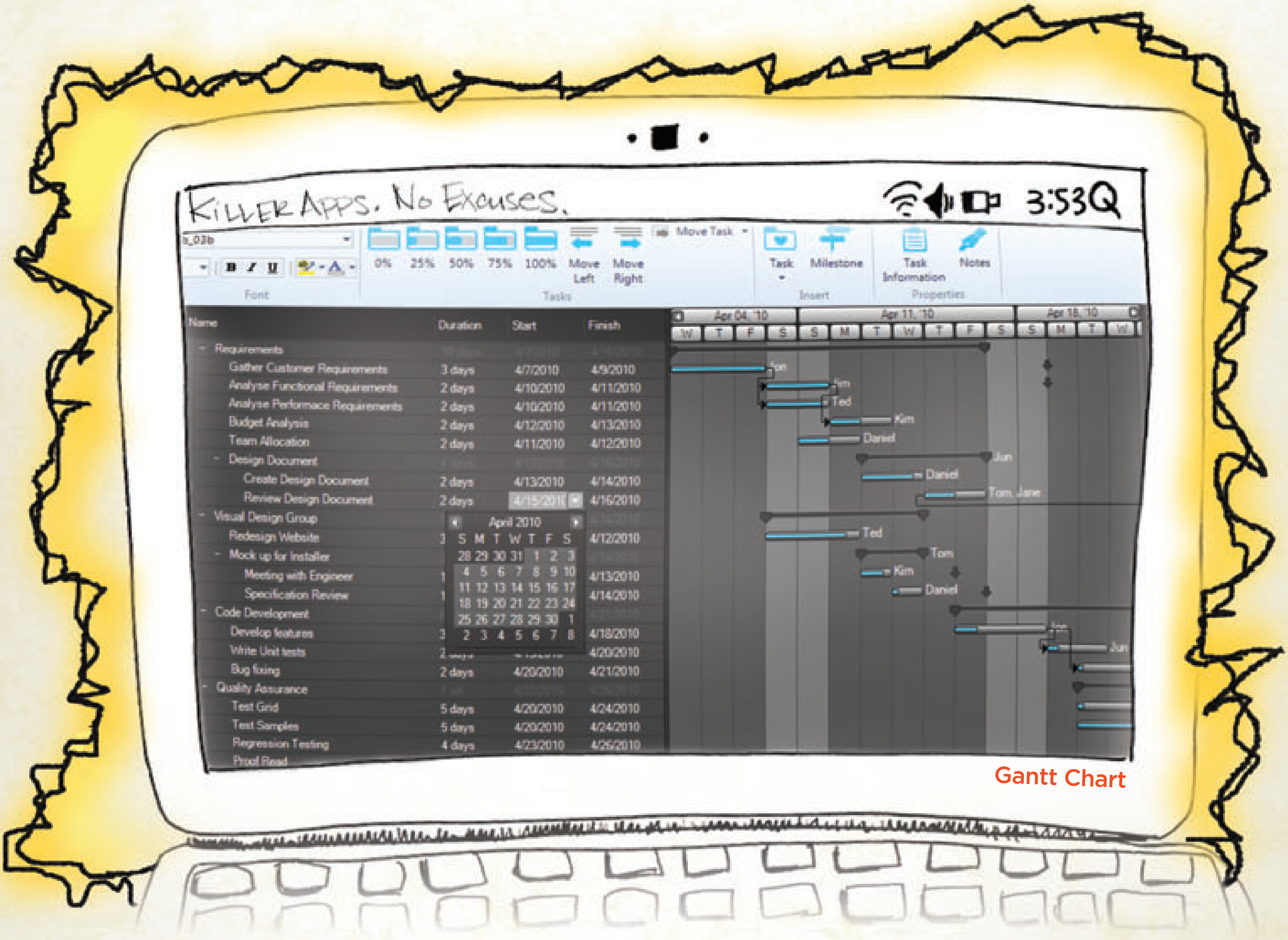
At first glance, it looks like it would take four queries to gather all the required information. In MyBookQuery, one query to get the appropriate user and another to load the user's books. The same appears to be the case in MyQueueQuery: one query to get the user and another to load the user's queue.

The use of a single session for the entire batch, however, shows that you're actually using the first-level cache to avoid unnecessary queries, as you can see in the NHibernate Profiler (nhprof.com) output in **Figure 7**.

Supporting Occasionally Connected Scenarios

As it stands, the application won't throw an error if the back-end server can't be reached, but it wouldn't be very useful, either.

INNOVATION TO SPARK KILLER APPS



Name	Duration	Start	Finish
− Requirements			
Gather Customer Requirements	3 days	4/7/2010	4/9/2010
Analyse Functional Requirements	2 days	4/10/2010	4/11/2010
Analyse Performace Requirements	2 days	4/10/2010	4/11/2010
Budget Analysis	2 days	4/12/2010	4/13/2010
Team Allocation	2 days	4/11/2010	4/12/2010
− Design Document			
Create Design Document	2 days	4/13/2010	4/14/2010
Review Design Document	2 days	4/15/2010	4/16/2010
− Visual Design Group			
Redesign Website	3		4/12/2010
− Mock up for Installer			
Meeting with Engineer	1		4/13/2010
Specification Review	1		4/14/2010
− Code Development			
Develop features	3		4/18/2010
Write Unit tests	2 days		4/20/2010
Bug fixing	2 days	4/20/2010	4/21/2010
− Quality Assurance			
Test Grid	5 days	4/20/2010	4/24/2010
Test Samples	5 days	4/20/2010	4/24/2010
Regression Testing	4 days	4/23/2010	4/26/2010
Proof Read			

Gantt Chart

You have the vision, but time, budget and staff constraints prevent you from seeing it through. With rich user interface controls like Gantt Charts that Infragistics **NetAdvantage® for .NET** adds to your Visual Studio 2010 toolbox, you can go to market faster with extreme functionality, complete usability and the "Wow-factor!" Go to infragistics.com/spark now to get innovative controls for creating Killer Apps.

Infragistics®

Killer Apps. No Excuses.

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

Figure 9 **Caching Incoming Connections**

```
private bool TransportOnMessageArrived(
  CurrentMessageInformation
  currentMessageInformation) {

  var cachableResponse =
    currentMessageInformation.Message as
    ICacheableResponse;
  if (cachableResponse == null)
    return false;

  var alreadyInCache = cache.Get(cachableResponse.Key);
  if (alreadyInCache == null ||
    alreadyInCache.Timestamp <
    cachableResponse.Timestamp) {

    cache.Put(cachableResponse.Key,
      cachableResponse.Timestamp, cachableResponse);
  }
  return false;
}
```

The next step in the evolution of this application is to turn it into a real occasionally connected client by introducing a cache that allows the application to continue operating even if the back-end server is not responding. However, I won't use the traditional caching architecture in which the application code makes explicit calls to the cache. Instead, I'll apply the cache at the infrastructure level.

Figure 8 shows the sequence of operations when the cache is implemented as part of the messaging infrastructure when you send a single message requesting data about a user's books.

The client sends a MyBooksQuery message. The message is sent on the bus while, at the same time, the cache is queried to see if it has the *response* for this request. If the cache contains the response for the previous request, the cache immediately causes the cached message to be consumed as if it just arrived on the bus.

The response from the back-end system arrives. The message is consumed normally and is also placed in the cache. On the surface, this approach seems to be complicated, but it results in effective caching behavior and allows you to almost completely ignore caching concerns in the application code. With a persistent cache (one that survives application restarts), you can operate the application completely independently without requiring any data from the back-end server.

Now let's implement this functionality. I assume a persistent cache (the sample code provides a simple implementation that uses binary serialization to save the values to disk) and define the following conventions:

Figure 10 **Dispatching Messages**

```
private void TransportOnMessageSent(
  CurrentMessageInformation
  currentMessageInformation) {

  var cacheableQuerys =
    currentMessageInformation.AllMessages.OfType<
    ICacheableQuery>();
  var responses =
    from msg in cacheableQuerys
    let response = cache.Get(msg.Key)
    where response != null
    select response.Value;

  var array = responses.ToArray();
  if (array.Length == 0)
    return;
  bus.ConsumeMessages(array);
}
```

- A message can be cached if it's part of a request/response message exchange.
- Both the request and response messages carry the cache key for the message exchange.

The message exchange is defined by an ICacheableQuery interface with a single Key property and an ICacheableResponse interface with Key and Timestamp properties.

To implement this convention, I write a CachingMessageModule that will run on the front end, intercepting incoming and outgoing messages. **Figure 9** shows how incoming messages are handled.

There isn't much going on here—if the message is a cacheable response, I put it in the cache. But there is one thing to note: I handle the case of out-of-order messages—messages that have an earlier timestamp arriving after messages with later timestamps. This ensures that only the latest information is stored in the cache.

Handling outgoing messages and dispatching the messages from the cache is more interesting, as you can see in **Figure 10**.

I gather the cached responses from the cache and call ConsumeMessages on them. That causes the bus to invoke the usual message invocation logic, so it looks like the message has arrived again.

> Even though there's a cached response, you still send the message.

Note, however, that even though there's a cached response, you still send the message. The reasoning is that you can provide a quick (cached) response for the user, and update the information shown to the user when the back end replies to new messages.

Next Steps

I have covered the basic building blocks of a smart client application: how to structure the back end and the communication mode between the smart client application and the back end. The latter is important because choosing the wrong communication mode can lead to the fallacies of distributed computing. I also touched on batching and caching, two very important approaches to improving the performance of a smart client application.

On the back end, you've seen how to manage transactions and the NHibernate session, how to consume and reply to messages from the client and how everything comes together in the bootstrapper.

In this article, I focused primarily on infrastructure concerns; in the next installment I'll cover best practices for sending data between the back end and the smart client application, and patterns for distributed change management. ■

OREN EINI *(who works under the pseudonym Ayende Rahien) is an active member of several open source projects (NHibernate and Castle among them) and is the founder of many others (Rhino Mocks, NHibernate Query Analyzer and Rhino Commons among them). Eini is also responsible for the NHibernate Profiler (nhprof.com), a visual debugger for NHibernate. You can follow Eini's work at ayende.com/blog.*

VSLive!
Presents:

SharePoint Live

August 3–6, 2010
Redmond, WA
Microsoft Campus

4 FULL DAYS OF EXCLUSIVE SHAREPOINT SESSIONS!

Maximize the Development Capability of SharePoint 2010!

CONFERENCE DAY 1—TUESDAY, AUGUST 3	
9:45– 11:00 AM	Whats New in the SharePoint 2010 Development Platform
11:15 AM– 12:30 PM	Introduction to SharePoint Development with Visual Studio 2010
1:45– 3:00 PM	Silverlight Development with SharePoint 2010
3:45– 5:00 PM	Feature Upgrade Enhancements in SharePoint 2010

CONFERENCE DAY 2—WEDNESDAY, AUGUST 4	
9:45– 11:00 AM	Creating SharePoint 2010 Workflows with SharePoint Designer 2010
11:15– 12:30 PM	Creating Workflow Templates with Visual Studio 2010
1:30– 2:45 PM	Integrating SharePoint 2010 and Azure: Evolving Towards the Cloud
3:00– 4:15 PM	Client-Side Development for SharePoint 2010 Using JavaScript and jQuery
5:00– 6:15 PM	PowerPivot and Excel Services

CONFERENCE DAY 3—THURSDAY, AUGUST 5		
8:30– 9:45 AM	SharePoint 2010 Support for Social Computing and Communities	
10:00– 11:15 AM	Developing Social Computing Solutions	SharePoint 2010 WCM for Developers with Visual Studio 2010
11:30 AM– 12:45 PM	SharePoint Developers Deep Dive into Claim-Based Security	Understanding How the Sandbox Works
1:45– 3:00 PM	Creating No-Code SharePoint Applications with SharePoint Designer	Best Practices for Upgrading Web Parts
3:15– 4:30 PM	Creating Extensions for the Visual Studio 2010 SharePoint Tools	Application Lifecycle Management for Developers in SharePoint 2010

POST-CONFERENCE WORKSHOP—FRIDAY, AUGUST 6 (SEPERATE ENTRY FEE REQUIRED)	
9:00 AM– 5:00 PM	SharePoint 2010 for ASP.NET Developers

RESERVE YOUR PLACE ON CAMPUS BEFORE IT'S TOO LATE!

VSLIVE.COM/REDMOND

USE PRIORITY CODE SPL3

Platinum Sponsors

CodeFluent Entities VERSANT

Gold Sponsors

AvePoint® ESRI

Supported by:

Microsoft® msdn Visual Studio MAGAZINE Microsoft® Visual Studio® 1105 MEDIA

New C# Features in the .NET Framework 4

Chris Burrows

Since its initial release in 2002, the C# programming language has been improved to enable programmers to write clearer, more maintainable code. The enhancements have come from the addition of features such as generic types, nullable value types, lambda expressions, iterator methods, partial classes and a long list of other useful language constructs. And, often, the changes were accompanied by giving the Microsoft .NET Framework libraries corresponding support.

This trend toward increased usability continues in C# 4.0. The additions make common tasks involving generic types, legacy interop and working with dynamic object models much simpler. This article aims to give a high-level survey of these new features. I'll begin with generic variance and then look at the legacy and dynamic interop features.

Covariance and Contravariance

Covariance and contravariance are best introduced with an example, and the best is in the framework. In System.Collections.Generic, IEnumerable<T> and IEnumerator <T> represent, respectively, an object that's a sequence of T's and the enumerator (or iterator) that does the work of iterating the sequence. These interfaces have done a lot of heavy lifting for a long time, because they support the implementation of the foreach loop construct. In C# 3.0, they became even more prominent because of their central role in LINQ and LINQ to Objects—they're the .NET interfaces to represent sequences.

So if you have a class hierarchy with, say, an Employee type and a Manager type that derives from it (managers are employees, after all), then what would you expect the following code to do?

```
IEnumerable<Manager> ms = GetManagers();
IEnumerable<Employee> es = ms;
```

It seems as though one ought to be able to treat a sequence of Managers as though it were a sequence of Employees. But in C# 3.0, the assignment will fail; the compiler will tell you there's no conversion. After all, it has no idea what the semantics of IEnumerable<T> are. This could be any interface, so for any arbitrary interface IFoo<T>, why would an IFoo<Manager> be more or less substitutable for an IFoo<Employee>?

In C# 4.0, though, the assignment works because IEnumerable<T>, along with a few other interfaces, has changed, an alteration enabled by new support in C# for covariance of type parameters.

IEnumerable<T> is eligible to be more special than the arbitrary IFoo<T> because, though it's not obvious at first glance, members that use the type parameter T (GetEnumerator in IEnumerable<T> and the Current property in IEnumerator<T>) actually use T only in the position of a return value. So you only get a Manager out of the sequence, and you never put one in.

This article discusses:

- Covariance and contravariance
- Dynamic dispatch
- Named arguments and optional properties
- COM interop

Technologies discussed:

C#, Microsoft .NET Framework 4, COM

In contrast, think of List<T>. Making a List<Manager> substitutable for a List<Employee> would be a disaster, because of the following:

```
List<Manager> ms = GetManagers();
List<Employee> es = ms; // Suppose this were possible
es.Add(new EmployeeWhoIsNotAManager()); // Uh oh
```

As this shows, once you think you're looking at a List<Employee>, you can insert any employee. But the list in question is actually a List<Manager>, so inserting a non-Manager must fail. You've lost type safety if you allow this. List<T> cannot be covariant in T.

The new language feature in C# 4.0, then, is the ability to define types, such as the new IEnumerable<T>, that admit conversions among themselves when the type parameters in question bear some relationship to one another. This is what the .NET Framework developers who wrote IEnumerable<T> used, and this is what their code looks like (simplified, of course):

```
public interface IEnumerable<out T> { /* ... */ }
```

Notice the out keyword modifying the definition of the type parameter, T. When the compiler sees this, it will mark T as covariant and check that, in the definition of the interface, all uses of T are up to snuff (in other words, that they're used in out positions only—that's why this keyword was picked).

Why is this called covariance? Well, it's easiest to see when you start to draw arrows. To be concrete, let's use the Manager and Employee types. Because there's an inheritance relationship between these classes, there's an implicit reference conversion from Manager to Employee:

Manager → Employee

And now, because of the annotation of T in IEnumerable<out T>, there's also an implicit reference conversion from IEnumerable<Manager> to IEnumerable<Employee>. That's what the annotation provides for:

IEnumerable<Manager> → IEnumerable<Employee>

This is called covariance, because the arrows in each of the two examples point in the same direction. We started with two types, Manager and Employee. We made new types out of them, IEnumerable<Manager> and IEnumerable<Employee>. The new types convert the same way as the old ones.

Contravariance is when this happens backward. You might anticipate that this could happen when the type parameter, T, is used only as input, and you'd be right. For example, the System namespace contains an interface called IComparable<T>, which has a single method called CompareTo:

```
public interface IComparable<in T> {
  bool CompareTo(T other);
}
```

If you have an IComparable<Employee>, you should be able to treat it as though it were an IComparable<Manager>, because the only thing you can do is put Employees in to the interface. Because a manager is an employee, putting a manager in should work, and it does. The in keyword modifies T in this case, and this scenario functions correctly:

```
IComparable<Employee> ec = GetEmployeeComparer();
IComparable<Manager> mc = ec;
```

This is called contravariance because the arrow got reversed this time:

Manager → Employee
IComparable<Manager> ← IComparable<Employee>

So the language feature here is pretty simple to summarize: You can add the keyword in or out whenever you define a type parameter, and doing so gives you free extra conversions. There are some limitations, though.

First, this works with generic interfaces and delegates only. You can't declare a generic type parameter on a class or struct in this manner. An easy way to rationalize this is that delegates are very much like interfaces that have just one method, and in any case, classes would often be ineligible for this treatment because of fields. You can think of any field on the generic class as being both an input and an output, depending on whether you write to it or read from it. If those fields involve type parameters, the parameters can be neither covariant nor contravariant.

Second, whenever you have an interface or delegate with a covariant or contravariant type parameter, you're granted new conversions on that type only when the type arguments, in the usage of the interface (not its definition), are reference types. For instance, because int is a value type, the IEnumerator<int> doesn't convert to IEnumerator <object>, even though it looks like it should:

IEnumerator <int> ↛ IEnumerator <object>

The reason for this behavior is that the conversion must preserve the type representation. If the int-to-object conversion were allowed, calling the Current property on the result would be impossible, because the value type int has a different representation on the stack than an object reference does. All reference types have the same representation on the stack, however, so only type arguments that are reference types yield these extra conversions.

Very likely, most C# developers will happily use this new language feature—they'll get more conversions of framework types and fewer compiler errors when using some types from the .NET Framework (IEnumerable<T>, IComparable<T>, Func<T>, Action<T>, among others). And, in fact, anyone designing a library with generic interfaces and delegates is free to use the new in and out type parameters when appropriate to make life easier for their users.

By the way, this feature does require support from the runtime—but the support has always been there. It lay dormant for several releases, however, because no language made use of it. Also, previous versions of C# allowed some limited conversions that were contravariant. Specifically, they let you make delegates out of methods that had compatible return types. In addition, array types have always been covariant. These existing features are distinct from the new ones in C# 4.0, which actually let you define your own types that are covariant and contravariant in some of their type parameters.

Dynamic Dispatch

On to the interop features in C# 4.0, starting with what is perhaps the biggest change.

C# now supports dynamic late-binding. The language has always been strongly typed, and it continues to be so in version 4.0. Microsoft believes this makes C# easy to use, fast and suitable for all the work .NET programmers are putting it to. But there are times when you need to communicate with systems not based on .NET.

Traditionally, there were at least two approaches to this. The first was simply to import the foreign model directly into .NET as a proxy. COM Interop provides one example. Since the original release of the

.NET Framework, it has used this strategy with a tool called TLBIMP, which creates new .NET proxy types you can use directly from C#.

LINQ-to-SQL, shipped with C# 3.0, contains a tool called SQLMETAL, which imports an existing database into C# proxy classes for use with queries. You'll also find a tool that imports Windows Management Instrumentation (WMI) classes to C#. Many technologies allow you to write C# (often with attributes) and then perform interop using your handwritten code as basis for external actions, such as LINQ-to-SQL, Windows Communication Foundation (WCF) and serialization.

The second approach abandons the C# type system entirely—you embed strings and data in your code. This is what you do whenever you write code that, say, invokes a method on a JScript object or when you embed a SQL query in your ADO.NET application. You're even doing this when you defer binding to run time using reflection, even though the interop in that case is with .NET itself.

The dynamic keyword in C# is a response to dealing with the hassles of these other approaches. Let's start with a simple example—reflection. Normally, using it requires a lot of boilerplate infrastructure code, such as:

```
object o = GetObject();
Type t = o.GetType();
object result = t.InvokeMember("MyMethod",
  BindingFlags.InvokeMethod, null,
  o, new object[] { });
int i = Convert.ToInt32(result);
```

With the dynamic keyword, instead of calling a method MyMethod on some object using reflection in this manner, you can now tell the compiler to please treat o as dynamic and delay all analysis until run time. Code that does that looks like this:

```
dynamic o = GetObject();
int i = o.MyMethod();
```

It works, and it accomplishes the same thing with code that's much less convoluted.

> C# 4.0 offers a simplified, consistent view of dynamic operations.

The value of this shortened, simplified C# syntax is perhaps more clear if you look at the ScriptObject class that supports operations on a JScript object. The class has an InvokeMember method that has more and different parameters, except in Silverlight, which actually has an Invoke method (notice the difference in the name) with fewer parameters. Neither of these are the same as what you'd need to invoke a method on an IronPython or IronRuby object or on any number of non-C# objects you might come into contact with.

In addition to objects that come from dynamic languages, you'll find a variety of data models that are inherently dynamic and have different APIs supporting them, such as HTML DOMs, the System.Xml DOM and the XLinq model for XML. COM objects are often dynamic and can benefit from the delay to run time of some compiler analysis.

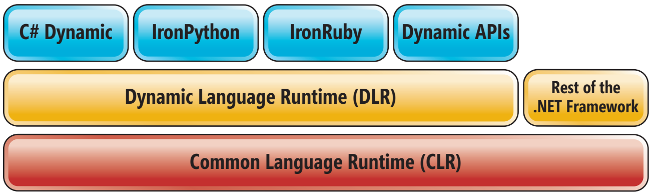

C# Dynamic
IronPython
IronRuby
Dynamic APIs
Dynamic Language Runtime (DLR)
Rest of the .NET Framework
Common Language Runtime (CLR)

Figure 1 **The DLR Runs on Top of the CLR**

Essentially, C# 4.0 offers a simplified, consistent view of dynamic operations. To take advantage of it, all you need to do is specify that a given value is dynamic, ensuring that analysis of all operations on the value will be delayed until run time.

In C# 4.0, dynamic is a built-in type, and a special pseudo-keyword signifies it. Note, however, that dynamic is different from var. Variables declared with var actually do have a strong type, but the programmer has left it up to the compiler to figure it out. When the programmer uses dynamic, the compiler doesn't know what type is being used—the programmer leaves figuring it out up to the runtime.

Dynamic and the DLR

The infrastructure that supports these dynamic operations at run time is called the Dynamic Language Runtime (DLR). This new .NET Framework 4 library runs on the CLR, like any other managed library. It's responsible for brokering each dynamic operation between the language that initiated it and the object it occurs on. If a dynamic operation isn't handled by the object it occurs on, a runtime component of the C# compiler handles the bind. A simplified and incomplete architecture diagram looks something like **Figure 1**.

The interesting thing about a dynamic operation, such as a dynamic method call, is that the receiver object has an opportunity to inject itself into the binding at run time and can, as a result, completely determine the semantics of any given dynamic operation. For instance, take a look at the following code:

```
dynamic d = new MyDynamicObject();
d.Bar("Baz", 3, d);
```

If MyDynamicObject was defined as shown here, then you can imagine what happens:

```
class MyDynamicObject : DynamicObject {
  public override bool TryInvokeMember(
    InvokeMemberBinder binder,
    object[] args, out object result) {

    Console.WriteLine("Method: {0}", binder.Name);
    foreach (var arg in args) {
      Console.WriteLine("Argument: {0}", arg);
    }

    result = args[0];
    return true;
  }
}
```

In fact, the code prints:

```
Method: Bar
Argument: Baz
Argument: 3
Argument: MyDynamicObject
```

By declaring d to be of type dynamic, the code that consumes the MyDynamicObject instance effectively opts out of compile-time checking for the operations d participates in. Use of dynamic means "I don't know what type this is going to be, so I don't know what methods or properties there are right now. Compiler, please let them all through and then figure it out when you really have an object at run time." So the call to Bar compiles even though the compiler doesn't know what it means. Then at run time, the object itself is asked what to do with this call to Bar. That's what TryInvokeMember knows how to handle.

Now, suppose that instead of a MyDynamicObject, you used a Python object:

```
dynamic d = GetPythonObject();
d.bar("Baz", 3, d);
```

If the object is the file listed here, then the code also works, and the output is much the same:

```
def bar(*args):
  print "Method:", bar.__name__
  for x in args:
    print "Argument:", x
```

Under the covers, for each use of a dynamic value, the compiler generates a bunch of code that initializes and uses a DLR CallSite. That CallSite contains all the information needed to bind at run time, including such things as the method name, extra data, such as whether the operation takes place in a checked context, and information about the arguments and their types.

This code, if you had to maintain it, would be every bit as ugly as the reflection code shown earlier or the ScriptObject code or strings that contain XML queries. That's the point of the dynamic feature in C#—you don't have to write code like that!

When using the dynamic keyword, your code can look pretty much the way you want: like a simple method invocation, a call to an indexer, an operator, such as +, a cast or even compounds, like += or ++. You can even use dynamic values in statements—for example, if(d) and foreach(var x in d). Short-circuiting is also supported, with code such as d && ShortCircuited or d ?? ShortCircuited.

The value of having the DLR provide a common infrastructure for these sorts of operations is that you're no longer having to deal with a different API for each dynamic model you'd like to code against—there's just a single API. And you don't even have to use it. The C# compiler can use it for you, and that should give you more time to actually write the code you want—the less infrastructure code you have to maintain means more productivity for you.

The C# language provides no shortcuts for defining dynamic objects. Dynamic in C# is all about *consuming and using* dynamic objects. Consider the following:

```
dynamic list = GetDynamicList();
dynamic index1 = GetIndex1();
dynamic index2 = GetIndex2();
string s = list[++index1, index2 + 10].Foo();
```

This code compiles, and it contains a lot of dynamic operations. First, there's the dynamic pre-increment on index1, then the dynamic add with index2. Then a dynamic indexer get is called on list. The product of those operations calls the member Foo. Finally, the total result of the expression is converted to a string and stored in s. That's five dynamic operations in one line, each dispatched at run time.

The compile-time type of each dynamic operation is itself dynamic, and so the "dynamicness" kind of flows from computation to computation. Even if you hadn't included dynamic expressions multiple times, there still would be a number of dynamic operations. There are still five in this one line:

```
string s = nonDynamicList[++index1, index2 + 10].Foo();
```

Because the results of the two indexing expressions are dynamic, the index itself is as well. And because the result of the index is dynamic, so is the call to Foo. Then you're confronted with converting a dynamic value to a string. That happens dynamically, of course, because the object could be a dynamic one that wants to perform some special computation in the face of a conversion request.

Notice in the previous examples that C# allows implicit conversions from any dynamic expression to any type. The conversion to string at the end is implicit and did not require an explicit cast operation. Similarly, any type can be converted to dynamic implicitly.

Dynamic in C# is all about consuming and using dynamic objects.

In this respect, dynamic is a lot like object, and the similarities don't stop there. When the compiler emits your assembly and needs to emit a dynamic variable, it does so by using the type object and then marking it specially. In some sense, dynamic is kind of an alias for object, but it adds the extra behavior of dynamically resolving operations when you use it.

You can see this if you try to convert between generic types that differ only in dynamic and object; such conversions will always work, because at runtime, an instance of List<dynamic> actually is an instance of List<object>:

```
List<dynamic> ld = new List<object>();
```

You can also see the similarity between dynamic and object if you try to override a method that's declared with an object parameter:

```
class C {
  public override bool Equals(dynamic obj) {
    /* ... */
  }
}
```

Although it resolves to a decorated object in your assembly, I do like to think of dynamic as a real type, because it serves as a reminder that you can do most things with it that you can do with any other type. You can use it as a type argument or, say, as a return value. For instance, this function definition will let you use the result of the function call dynamically without having to put its return value in a dynamic variable:

```
public dynamic GetDynamicThing() {
  /* ... */ }
```

There are a lot more details about the way dynamic is treated and dispatched, but you don't need to know them to use the feature. The essential idea is that you can write code that looks like C#, and if any part of the code you write is dynamic, the compiler will leave it alone until run time.

I want to cover one final topic concerning dynamic: failure. Because the compiler can't check whether the dynamic thing you're using really has the method called Foo, it can't give you an error. Of course, that doesn't mean that your call to Foo will work at run time. It may work, but there are a lot of objects that don't have a method called Foo. When your expression fails to bind at run time, the binder makes its best attempt to give you an exception that's more or less exactly what the compiler would've told you if you hadn't used dynamic to begin with.

Consider the following code:

```
try
{
  dynamic d = "this is a string";
  d.Foo();
}
catch (Microsoft.CSharp.RuntimeBinder.RuntimeBinderException e)
{
  Console.WriteLine(e.Message);
}
```

Here I have a string, and strings clearly do not have a method called Foo. When the line that calls Foo executes, the binding will fail and you'll get a RuntimeBinderException. This is what the previous program prints:

```
'string' does not contain a definition for 'Foo'
```

Which is exactly the error message you, as a C# programmer, expect.

Named Arguments and Optional Parameters

In another addition to C#, methods now support optional parameters with default values so that when you call such a method you can omit those parameters. You can see this in action in this Car class:

```
class Car {
  public void Accelerate(
    double speed, int? gear = null,
    bool inReverse = false) {

    /* ... */
  }
}
```

You can call the method this way:

```
Car myCar = new Car();
myCar.Accelerate(55);
```

This has exactly the same effect as:

```
myCar.Accelerate(55, null, false);
```

It's the same because the compiler will insert all the default values that you omit.

C# 4.0 will also let you call methods by specifying some arguments by name. In this way, you can pass an argument to an optional parameter without having to also pass arguments for all the parameters that come before it.

Say you want to call Accelerate to go in reverse, but you don't want to specify the gear parameter. Well, you can do this:

```
myCar.Accelerate(55, inReverse: true);
```

This is a new C# 4.0 syntax, and it's the same as if you had written:

```
myCar.Accelerate(55, null, true);
```

In fact, whether or not parameters in the method you're calling are optional, you can use names when passing arguments. For instance, these two calls are permissible and identical to one another:

```
Console.WriteLine(format: "{0:f}", arg0: 6.02214179e23);
Console.WriteLine(arg0: 6.02214179e23, format: "{0:f}");
```

If you're calling a method that takes a long list of parameters, you can even use names as a sort of in-code documentation to help you remember which parameter is which.

On the surface, optional arguments and named parameters don't look like interop features. You can use them without ever even thinking about interop. However, the motivation for these features comes from the Office APIs. Consider, for example, Word programming and something as simple as the SaveAs method on the Document interface. This method has 16 parameters, all of which are optional. With previous versions of C#, if you want to call this method you have to write code that looks like this:

```
Document d = new Document();
object filename = "Foo.docx";
object missing = Type.Missing;
d.SaveAs(ref filename, ref missing, ref missing, ref missing, ref
missing, ref missing, ref missing, ref missing, ref missing, ref
missing, ref missing, ref missing, ref missing, ref missing, ref
missing, ref missing);
```

Now, you can write this:

```
Document d = new Document();
d.SaveAs(FileName: "Foo.docx");
```

I would say that's an improvement for anyone who works with APIs like this. And improving the lives of programmers who need to write Office programs was definitely a motivating factor for adding named arguments and optional parameters to the language.

Now, when writing a .NET library and considering adding methods that have optional parameters, you're faced with a choice. You can either add optional parameters or you can do what C# programmers have done for years: introduce overloads. In the Car.Accelerate example, the latter decision might lead you to produce a type that looks like this:

```
class Car {
  public void Accelerate(uint speed) {
    Accelerate(speed, null, false);
  }
  public void Accelerate(uint speed, int? gear) {
    Accelerate(speed, gear, false);
  }
  public void Accelerate(uint speed, int? gear,
    bool inReverse) {
    /* ... */
  }
}
```

Selecting the model that suits the library you're writing is up to you. Because C# hasn't had optional parameters until now, the .NET Framework (including the .NET Framework 4) tends to use overloads. If you decide to mix and match overloads with optional parameters, the C# overload resolution has clear tie-breaking rules to determine which overload to call under any given circumstances.

Selecting the model that suits the library you're writing is up to you.

Indexed Properties

Some smaller language features in C# 4.0 are supported only when writing code against a COM interop API. The Word interop in the previous illustration is one example.

C# code has always had the notion of an indexer that you can add to a class to effectively overload the [] operator on instances of that class. This sense of indexer is also called a default indexer, since it isn't given a name and calling it requires no name. Some COM APIs also have indexers that aren't default, which is to say that you can't effectively call them simply by using []—you must specify a name. You can, alternatively, think of an indexed property as a property that takes some extra arguments.

C# 4.0 supports indexed properties on COM interop types. You can't define types in C# that have indexed properties, but you can use them provided you're doing so on a COM type. For an example of what C# code that does this looks like, consider the Range property on an Excel worksheet:

```
using Microsoft.Office.Interop.Excel;

class Program {
  static void Main(string[] args) {
    Application excel = new Application();
    excel.Visible = true;

    Worksheet ws =
      excel.Workbooks.Add().Worksheets["Sheet1"];
    // Range is an indexed property
    ws.Range["A1", "C3"].Value = 123;
    System.Console.ReadLine();
    excel.Quit();
  }
}
```

In this example, Range["A1", "C3"] isn't a property called Range that returns a thing that can be indexed. It's one call to a Range accessor that passes A1 and C3 with it. And although Value might not look like an indexed property, it, too, is one! All of its arguments are optional, and because it's an indexed property, you omit them by not specifying them at all. Before the language supported indexed properties, you would have written the call like this:

```
ws.get_Range("A1", "C3").Value2 = 123;
```

Here, Value2 is a property that was added simply because the indexed property Value wouldn't work prior to C# 4.0.

Omitting the Ref Keyword at COM Call Sites

Some COM APIs were written with many parameters passed by reference, even when the implementation doesn't write back to them. In the Office suite, Word stands out as an example—its COM APIs all do this.

When you're confronted with such a library and you need to pass arguments by reference, you can no longer pass any expression that's not a local variable or field, and that's a big headache. In the Word SaveAs example, you can see this in action—you had to declare a local called filename and a local called missing just to call the SaveAs method, since those parameters needed to be passed by reference.

```
Document d = new Document();
object filename = "Foo.docx";
object missing = Type.Missing;
d.SaveAs(ref filename, ref missing, // ...
```

You may have noticed in the new C# code that followed, I no longer declared a local for filename:

```
d.SaveAs(FileName: "Foo.docx");
```

This is possible because of the new omit ref feature for COM interop. Now, when calling a COM interop method, you can pass any argument by value instead of by reference. If you do, the compiler will create a temporary local on your behalf and pass that local by reference for you if required. Of course, you won't be able to see the effect of the method call if the method mutates the argument—if you want that, pass the argument by ref.

This should make code that uses APIs like this much cleaner.

Embedding COM Interop Types

This is more of a C# compiler feature than a C# language feature, but now you can use a COM interop assembly without that assembly having to be present at run time. The goal is to reduce the burden of deploying COM interop assemblies with your application.

When COM interop was introduced in the original version of the .NET Framework, the notion of a Primary Interop Assembly (PIA) was created. This was an attempt to solve the problem of sharing COM objects among components. If you had different interop assemblies that defined an Excel Worksheet, we wouldn't be able to share these Worksheets between components, because they would be different .NET types. The PIA fixed this by existing only once—all clients used it, and the .NET types always matched.

> Though a fine idea on paper, in practice deploying a PIA turns out to be a headache.

Though a fine idea on paper, in practice deploying a PIA turns out to be a headache, because there's only one, and multiple applications could try to install or uninstall it. Matters are complicated because PIAs are often large, Office doesn't deploy them with default Office installations, and users can circumvent this single assembly system easily just by using TLBIMP to create their own interop assembly.

So now, in an effort to fix this situation, two things have happened:

- The runtime has been given the smarts to treat two structurally identical COM interop types that share the same identifying characteristics (name, GUID and so on) as though they were actually the same .NET type.
- The C# compiler takes advantage of this by simply reproducing the interop types in your own assembly when you compile, removing the need for the interop assembly to exist at run time.

I have to omit some details in the interest of space, but even without knowledge of the details, this is another feature—like dynamic—you should be able to use without a problem. You tell the compiler to embed interop types for you in Visual Studio by setting the Embed Interop Types property on your reference to true.

Because the C# team expects this to be the preferred method of referencing COM assemblies, Visual Studio will set this property to True by default for any new interop reference added to a C# project. If you're using the command-line compiler (csc.exe) to build your code, then to embed interop types you must reference the interop assembly in question using the /L switch rather than /R.

Each of the features I've covered in this article could itself generate much more discussion, and the topics all deserve articles of their own. I've omitted or glossed over many details, but I hope this serves as a good starting point for exploring C# 4.0 and you find time to investigate and make use of these features. And if you do, I hope you enjoy the benefits in productivity and program readability they were designed to give you.

Chris Burrows *is a developer at Microsoft on the C# compiler team. He implemented dynamic in the C# compiler and has been involved with the development of Visual Studio for nine years.*

Thanks to the following technical expert for reviewing this article:
Eric Lippert

Problems and Solutions with Model-View-ViewModel

Robert McCarter

Windows Presentation Foundation (WPF) and Silverlight provide rich APIs for building modern applications, but understanding and applying all the WPF features in harmony with each other to build well-designed and easily maintained apps can be difficult. Where do you start? And what is the right way to compose your application?

The Model-View-ViewModel (MVVM) design pattern describes a popular approach for building WPF and Silverlight applications. It's both a powerful tool for building applications and a common language for discussing application design with developers. While MVVM is a really useful pattern, it's still relatively young and misunderstood.

When is the MVVM design pattern applicable, and when is it unnecessary? How should the application be structured? How much work is the ViewModel layer to write and maintain, and what alternatives exist for reducing the amount of code in the ViewModel layer? How are related properties within the Model handled elegantly? How should you expose collections within the Model to the View? Where should ViewModel objects be instantiated and hooked up to Model objects?

This article discusses:

- Model, ViewModel, and View
- Why use a ViewModel?
- Using dynamic properties
- A document manager adapter

Technologies discussed:

Windows Presentation Foundation, Silverlight

In this article I'll explain how the ViewModel works, and discuss some benefits and issues involved in implementing a ViewModel in your code. I'll also walk you through some concrete examples of using ViewModel as a document manager for exposing Model objects in the View layer.

Model, ViewModel and View

Every WPF and Silverlight application I've worked on so far had the same high-level component design. The Model was the core of the application, and a lot of effort went into designing it according to object-oriented analysis and design (OOAD) best practices.

For me the Model is the heart of the application, representing the biggest and most important business asset because it captures all the complex business entities, their relationships and their functionality.

Sitting atop the Model is the ViewModel. The two primary goals of the ViewModel are to make the Model easily consumable by the WPF/XAML View and to separate and encapsulate the Model from the View. These are excellent goals, although for pragmatic reasons they're sometimes broken.

You build the ViewModel knowing how the user will interact with the application at a high level. However, it's an important part of the MVVM design pattern that the ViewModel knows nothing about the View. This allows the interaction designers and graphics artists to create beautiful, functional UIs on top of the ViewModel while working closely with the developers to design a suitable ViewModel to support their efforts. In addition, decoupling between View and ViewModel also allows the ViewModel to be more unit testable and reusable.

To help enforce a strict separation between the Model, View and ViewModel layers, I like to build each layer as a separate Visual Studio project. Combined with the reusable utilities, the main executable assembly and any unit testing projects (you have plenty of these, right?), this can result in a lot of projects and assemblies, as illustrated in **Figure 1**.

Given the large number of projects, this strict-separation approach is obviously most useful on large projects. For small applications with only one or two developers, the benefits of this strict separation may not outweigh the inconvenience of creating, configuring and maintaining multiple projects, so simply separating your code into different namespaces within the same project may provide more than sufficient isolation.

Writing and maintaining a ViewModel is not trivial and it should not be undertaken lightly. However, the answer to the most basic questions—when should you consider the MVVM design pattern and when is it unnecessary—is often found in your domain model.

In large projects, the domain model may be very complex, with hundreds of classes carefully designed to work elegantly together for any type of application, including Web services, WPF or ASP.NET applications. The Model may comprise several assemblies working together, and in very large organizations the domain model is sometimes built and maintained by a specialized development team.

When you have a large and complex domain model, it's almost always beneficial to introduce a ViewModel layer.

On the other hand, sometimes the domain model is simple, perhaps nothing more than a thin layer over the database. The classes may be automatically generated and they frequently implement INotifyPropertyChanged. The UI is commonly a collection of lists or grids with edit forms allowing the user to manipulate the underlying data. The Microsoft toolset has always been very good at building these kinds of applications quickly and easily.

If your model or application falls into this category, a ViewModel would probably impose unacceptably high overhead without sufficiently benefitting your application design.

That said, even in these cases the ViewModel can still provide value. For example, the ViewModel is an excellent place to implement undo functionality. Alternatively, you can choose to use MVVM for a portion of the application (such as document management, as I'll discuss later) and pragmatically expose your Model directly to the View.

Why Use a ViewModel?

If a ViewModel seems appropriate for your application, there are still questions to be answered before you start coding. One of the first is how to reduce the number of proxy properties.

The separation of the View from the Model promoted by the MVVM design pattern is an important and valuable aspect of the pattern. As a result, if a Model class has 10 properties that need to be exposed in the View, the ViewModel typically ends up having 10 identical properties that simply proxy the call to the underlying model instance. These proxy properties usually raise a property-changed event when set to indicate to the View that the property has been changed.

Not every Model property needs to have a ViewModel proxy property, but every Model property that needs to be exposed in the View will typically have a proxy property. The proxy properties usually look like this:

```
public string Description {
  get {
    return this.UnderlyingModelInstance.Description;
  }
  set {
    this.UnderlyingModelInstance.Description = value;
    this.RaisePropertyChangedEvent("Description");
  }
}
```

Any non-trivial application will have tens or hundreds of Model classes that need to be exposed to the user through the ViewModel in this fashion. This is simply intrinsic to the separation provided by MVVM.

The Model is the heart of the application.

Writing these proxy properties is boring and therefore error-prone, especially because raising the property-changed event requires a string that must match the name of the property (and will not be included in any automatic code refactoring). To eliminate these proxy events, the common solution is to expose the model instance from the ViewModel wrapper directly, then have the domain model implement the *INotifyPropertyChanged* interface:

```
public class SomeViewModel {
  public SomeViewModel( DomainObject domainObject ) {
    Contract.Requires(domainObject!=null,
      "The domain object to wrap must not be null");
    this.WrappedDomainObject = domainObject;
  }
  public DomainObject WrappedDomainObject {
    get; private set;
  }
...
```

Thus, the ViewModel can still expose the commands and additional properties required by the view without duplicating Model properties or creating lots of proxy properties. This approach certainly has its appeal, especially if the Model classes already implement the INotifyPropertyChanged interface. Having the model implement this interface isn't necessarily a bad thing and it was even common with Microsoft .NET Framework 2.0 and Windows Forms applications. It does clutter up the domain model, though, and wouldn't be useful for ASP.NET applications or domain services.

With this approach the View has a dependency on the Model, but it's only an indirect dependency through data binding, which does not require a project reference from the View project to the Model project. So for purely pragmatic reasons this approach is sometimes useful.

However, this approach does violate the spirit of the MVVM design pattern, and it reduces your ability to introduce new ViewModel-specific functionality later (such as undo capabilities). I've encountered scenarios with this approach that caused a fair bit of rework. Imagine the not-uncommon situation where there's a data binding on a deeply nested property. If the Person ViewModel is the current data context, and the Person has an Address, the data binding might look something like this:

```
{Binding WrappedDomainObject.Address.Country}
```

If you ever need to introduce additional ViewModel functionality on the Address object, you'll need to remove data binding references to WrappedDomainObject.Address and instead use new ViewModel properties. This is problematic because updates to the XAML data binding (and possibly the data contexts as well) are hard to test. The View is the one component that doesn't have automated and comprehensive regression tests.

The ViewModel is built knowing how the user will interact with the application at a high level.

Dynamic Properties

My solution to the proliferation of proxy properties is to use the new .NET Framework 4 and WPF support for dynamic objects and dynamic method dispatch. The latter allows you to determine at run time how to handle reading or writing to a property that does not actually exist on the class. This means you can eliminate all the handwritten proxy properties in the ViewModel while still encapsulating the underlying model. Note, however, that Silverlight 4 does not support binding to dynamic properties.

The simplest way to implement this capability is to have the ViewModel base class extend the new System.Dynamic.DynamicObject class and override the TryGetMember and TrySetMember methods. The Dynamic Language Runtime (DLR) calls these two methods when the property being referenced does not exist on the class, allowing the class to determine at run time how to implement the missing properties. Combined with a small amount of reflection, the ViewModel class can dynamically proxy the property access to the underlying model instance in only a few lines of code:

```
public override bool TryGetMember(
  GetMemberBinder binder, out object result) {

  string propertyName = binder.Name;
  PropertyInfo property =
    this.WrappedDomainObject.GetType().GetProperty(propertyName);

  if( property==null || property.CanRead==false ) {
    result = null;
    return false;
  }

  result = property.GetValue(this.WrappedDomainObject, null);
  return true;
}
```

The method starts by using reflection to find the property on the underlying Model instance. (For more details, see the June 2007 "CLR Inside Out" column "Reflections on Reflection" at msdn.microsoft.com/magazine/cc163408.) If the model doesn't have such a property, then the method fails by returning false and the data binding fails. If the property exists, the method uses the property information to retrieve and return the Model's property value. This is more work than the traditional proxy property's get method, but this is the only implementation you need to write for all models and all properties.

The real power of the dynamic proxy property approach is in the property setters. In TrySetMember, you can include common logic such as raising property-changed events. The code looks something like this:

```
public override bool TrySetMember(
  SetMemberBinder binder, object value) {

  string propertyName = binder.Name;
  PropertyInfo property =
    this.WrappedDomainObject.GetType().GetProperty(propertyName);

  if( property==null || property.CanWrite==false )
    return false;

  property.SetValue(this.WrappedDomainObject, value, null);

  this.RaisePropertyChanged(propertyName);
  return true;
}
```

Again, the method starts by using reflection to grab the property from the underlying Model instance. If the property doesn't exist or the property is read-only, the method fails by returning false. If the property exists on the domain object, the property information is used to set the Model property. Then you can include any logic common to all property setters. In this sample code I simply raise the property-changed event for the property I just set, but you can easily do more.

One of the challenges of encapsulating a Model is that the Model frequently has what Unified Modeling Language calls derived properties. For example, a Person class probably has a BirthDate property and a derived Age property. The Age property is read-only and automatically calculates the age based on the birth date and the current date:

```
public class Person : DomainObject {
  public DateTime BirthDate {
    get; set;
  }

  public int Age {
    get {
      var today = DateTime.Now;
      // Simplified demo code!
      int age = today.Year - this.BirthDate.Year;
      return age;
    }
  }
...
```

When the BirthDate property changes, the Age property also implicitly changes because the age is derived mathematically from the birth date. So when the BirthDate property is set, the ViewModel class needs to raise a property-changed event for both the BirthDate property and the Age property. With the dynamic ViewModel approach, you can do this automatically by making this inter-property relationship explicit within the model.

First, you need a custom attribute to capture the property relationship:

```
[AttributeUsage(AttributeTargets.Property, AllowMultiple=true)]
public sealed class AffectsOtherPropertyAttribute : Attribute {
  public AffectsOtherPropertyAttribute(
    string otherPropertyName) {
    this.AffectsProperty = otherPropertyName;
  }

  public string AffectsProperty {
    get;
    private set;
  }
}
```

I set AllowMultiple to true to support scenarios where a property can affect multiple other properties. Applying this attribute to codify the relationship between BirthDate and Age directly in the model is straightforward:

```
[AffectsOtherProperty("Age")]
public DateTime BirthDate { get; set; }
```

To use this new model metadata within the dynamic ViewModel class, I can now update the TrySetMember method with three additional lines of code, so it looks like this:

```
public override bool TrySetMember(
  SetMemberBinder binder, object value) {
...
  var affectsProps = property.GetCustomAttributes(
    typeof(AffectsOtherPropertyAttribute), true);
  foreach(AffectsOtherPropertyAttribute otherPropertyAttr
    in affectsProps)
    this.RaisePropertyChanged(
      otherPropertyAttr.AffectsProperty);
}
```

With the reflected property information already in hand, the GetCustomAttributes method can return any AffectsOtherProperty attributes on the model property. Then the code simply loops over the attributes, raising property-changed events for each one. So changes to the BirthDate property through the ViewModel now automatically raise both BirthDate and Age property-changed events.

It's important to realize that if you explicitly program a property on the dynamic ViewModel class (or, more likely, on model-specific derived ViewModel classes), the DLR will not call the TryGetMember and TrySetMember methods and will instead call the properties directly. In that case, you lose this automatic behavior. However, the code could easily be refactored so that custom properties could use this functionality as well.

Let's return to the problem of the data binding on a deeply nested property (where the ViewModel is the current WPF data context) that looks like this:

```
{Binding WrappedDomainObject.
Address.Country}
```

Using dynamic proxy properties means the underlying wrapped domain object is no longer exposed, so the data binding would actually look like this:

```
{Binding Address.Country}
```

In this case, the Address property would still access the underlying model Address instance directly. However, now when you want to introduce a ViewModel around the Address, you simply add a new property on the Person ViewModel class. The new Address property is very simple:

```
public DynamicViewModel Address {
  get {
    if( addressViewModel==null )
      addressViewModel =
        new DynamicViewModel(this.Person.Address);
    return addressViewModel;
  }
}

private DynamicViewModel addressViewModel;
```

No XAML data bindings need to be changed because the property is still called Address, but now the DLR calls the new concrete property rather than the dynamic TryGetMember method. (Notice that the lazy instantiation within this Address property is not thread-safe. However, only the View should be accessing the ViewModel and the WPF/Silverlight view is single-threaded, so this is not a concern.)

This approach can be used even when the model implements INotifyPropertyChanged. The ViewModel can notice this and choose not to proxy property-changed events. In this case, it listens for them from the underlying model instance and then re-raises the events as its own. In the constructor of the dynamic ViewModel class, I perform the check and remember the result:

```
public DynamicViewModel(DomainObject model) {
  Contract.Requires(model != null,
    "Cannot encapsulate a null model");
  this.ModelInstance = model;

  // Raises its own property changed events
  if( model is INotifyPropertyChanged ) {
    this.ModelRaisesPropertyChangedEvents = true;
    var raisesPropChangedEvents =
      model as INotifyPropertyChanged;
    raisesPropChangedEvents.PropertyChanged +=
      (sender,args) =>
      this.RaisePropertyChanged(args.PropertyName);
  }
}
```

To prevent duplicate property-changed events, I also need to make a slight modification to the TrySetMember method.

```
if( this.ModelRaisesPropertyChangedEvents==false )
  this.RaisePropertyChanged(property.Name);
```

So you can use a dynamic proxy property to dramatically simplify the ViewModel layer by eliminating standard proxy properties.

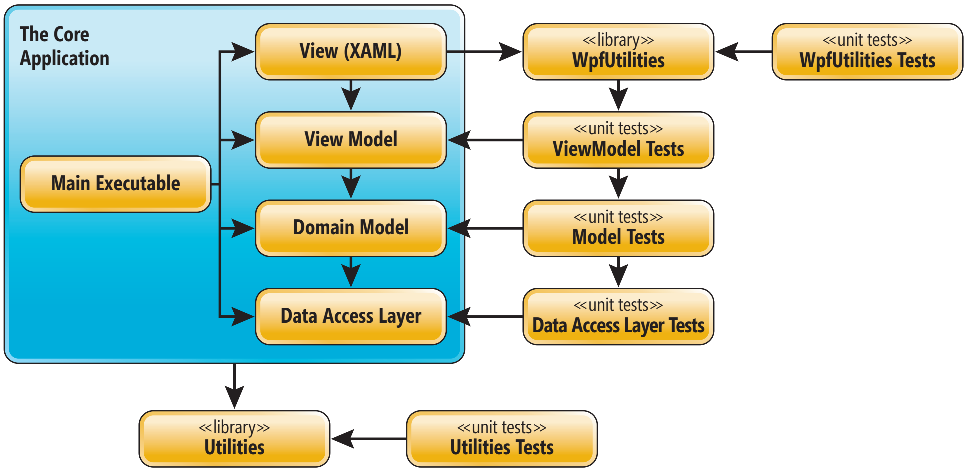


Figure 1 **The Components of an MVVM Application**

This significantly reduces coding, testing, documentation and long-term maintenance. Adding new properties to the model no longer requires updating the ViewModel layer unless there is very special View logic for the new property. Additionally, this approach can solve difficult issues like related properties. The common TrySetMember method could also help you implement an undo capability because user-driven property changes all flow through the TrySetMember method.

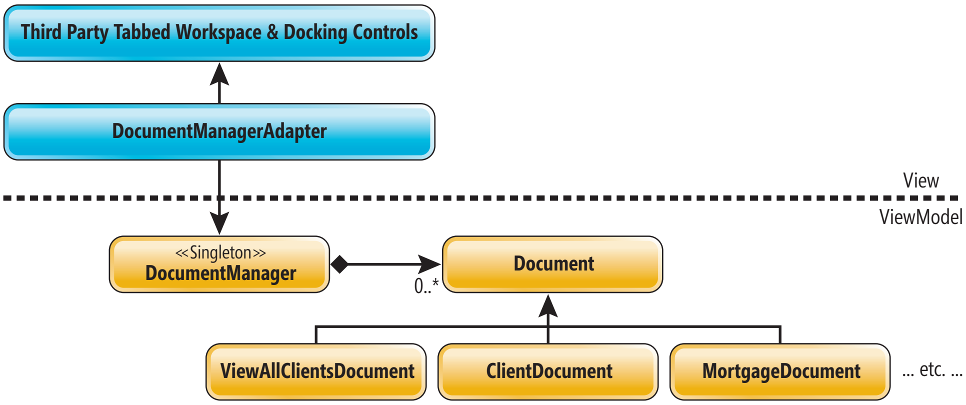


Figure 2 **Document Manager View Adapter**

Pros and Cons

Many developers are leery of reflection (and the DLR) because of performance concerns. In my own work I haven't found this to be a problem. The performance penalty for the user when setting a single property in the UI is not likely to be noticed. That may not be the case in highly interactive UIs, such as multi-touch design surfaces.

The only major performance issue is in the initial population of the view when there are a large number of fields. Usability concerns should naturally limit the number of fields you're exposing on any screen so that the performance of the initial data bindings through this DLR approach is undetectable.

Nevertheless, performance should always be carefully monitored and understood as it relates to the user experience. The simple approach previously described could be rewritten with reflection caching. For additional details, see Joel Pobar's article in the July 2005 issue of *MSDN Magazine* (msdn.microsoft.com/magazine/cc163759).

There is some validity to the argument that code readability and maintainability are negatively affected using this approach because the View layer seems to be referencing properties on the ViewModel that don't actually exist. However, I believe the benefits of eliminating most of the hand-coded proxy properties far outweigh the problems, especially with proper documentation on the ViewModel.

The dynamic proxy property approach does reduce or eliminate the ability to obfuscate the Model layer because the properties on the Model are now referenced by name in the XAML. Using traditional proxy properties does not limit your ability to obfuscate the Model because the properties are referenced directly and would be obfuscated with the rest of the application. However, as most obfuscation tools do not yet work with XAML/BAML, this is largely irrelevant. A code cracker can start from the XAML/BAML and work into the Model layer in either case.

Finally, this approach could be abused by attributing model properties with security-related metadata and expecting the ViewModel to be responsible for enforcing security. Security doesn't seem like a View-specific responsibility, and I believe this is placing too many responsibilities on the ViewModel. In this case, an aspect-oriented approach applied within the Model would be more suitable.

Collections

Collections are one of the most difficult and least satisfactory aspects of the MVVM design pattern. If a collection in the underlying Model is changed by the Model, it's the responsibility of the ViewModel to somehow expose the change so that the View can update itself appropriately.

Unfortunately, in all likelihood the Model does not expose collections that implement the INotifyCollectionChanged interface. In the .NET Framework 3.5, this interface is in the System.Windows.dll, which strongly discourages its use in the Model. Fortunately, in the .NET Framework 4, this interface has migrated to System.dll, making it much more natural to use observable collections from within the Model.

Observable collections in the Model open up new possibilities for Model development and could be used in Windows Forms and Silverlight applications. This is currently my preferred approach because it's much simpler than anything else, and I'm happy the INotifyCollectionChanged interface is moving to a more common assembly.

Without observable collections in the Model, the best that can be done is to expose some other mechanism—most likely custom events—on the Model to indicate when the collection has changed. This should be done in a Model-specific way. For example, if the Person class had a collection of addresses it could expose events such as:

```
public event EventHandler<AddressesChangedEventArgs>
  NewAddressAdded;
public event EventHandler<AddressesChangedEventArgs>
  AddressRemoved;
```

This is preferable to raising a custom collection event designed specifically for the WPF ViewModel. However, it's still difficult to expose collection changes in the ViewModel. Likely, the only recourse is to raise a property-changed event on the entire ViewModel collection property. This is an unsatisfactory solution at best.

Another problem with collections is determining when or if to wrap each Model instance in the collection within a ViewModel instance. For smaller collections, the ViewModel may expose a new observable collection and copy everything in the underlying Model collection into the ViewModel observable collection, wrapping each Model item in the collection in a corresponding ViewModel instance as it goes. The ViewModel might need to listen for collection-changed events to transmit user changes back to the underlying Model.

However, for very large collections that will be exposed in some form of virtualizing panel, the easiest and most pragmatic approach is just to expose the Model objects directly.

Instantiating the ViewModel

Another problem with the MVVM design pattern that's seldom discussed is where and when the ViewModel instances should be instantiated. This problem is also frequently overlooked in discussions of similar design patterns such as MVC.

My preference is to write a ViewModel singleton that provides the main ViewModel objects from which the View can easily retrieve all other ViewModel objects as required. Often this master ViewModel object provides the command implementations so the View can support opening of documents.

However, most of the applications I've worked with provide a document-centric interface, usually using a tabbed workspace similar to Visual Studio. So in the ViewModel layer I want to think in terms of documents, and the documents expose one or more ViewModel objects wrapping particular Model objects. Standard WPF commands in the ViewModel layer can then use the persistence layer to retrieve the necessary objects, wrap them in ViewModel instances and create ViewModel document managers to display them.

In the sample application included with this article, the ViewModel command for creating a new Person is:

```
internal class OpenNewPersonCommand : ICommand {
...
  // Open a new person in a new window.
  public void Execute(object parameter) {
    var person = new MvvmDemo.Model.Person();
    var document = new PersonDocument(person);
    DocumentManager.Instance.ActiveDocument = document;
  }
}
```

The ViewModel document manager referenced in the last line is a singleton that manages all open ViewModel documents. The question is, how does the collection of ViewModel documents get exposed in the View?

The built-in WPF tab control does not provide the kind of powerful multiple-document interface users have come to expect. Fortunately, third-party docking and tabbed-workspace products are available. Most of them strive to emulate the tabbed document look of Visual Studio, including the dockable tool windows, split views, Ctrl+Tab pop-up windows (with mini-document views) and more.

Unfortunately, most of these components don't provide built-in support for the MVVM design pattern. But that's OK, because you can easily apply the Adapter design pattern to link the ViewModel document manager to the third-party view component.

Document Manager Adapter

The adapter design shown in **Figure 2** ensures that the ViewModel doesn't require any reference to the View, so it respects the main goals of the MVVM design pattern. (However, in this case, the concept of a document is defined in the ViewModel layer rather than the Model layer because it's purely a UI concept.)

> The real power of the dynamic proxy property approach is in the property setters.

The ViewModel document manager is responsible for maintaining the collection of open ViewModel documents and knowing which document is currently active. This design allows the ViewModel layer to open and close documents using the document manager, and to change the active document without any knowledge of the View. The ViewModel side of this approach is reasonably straightforward. The ViewModel classes in the sample application are shown in **Figure 3**.

The Document base class exposes several internal lifecycle methods (Activated, LostActivation and DocumentClosed) that are called by the document manager to keep the document up-to-date about what's going on. The document also implements an INotifyPropertyChanged interface so that it can support data binding. For example, the adapter data binds the view document's Title property to the ViewModel's DocumentTitle property.

The most complex piece of this approach is the adapter class, and I've provided a working copy in the project accompanying this article. The adapter subscribes to events on the document manager and uses those events to keep the tabbed-workspace control

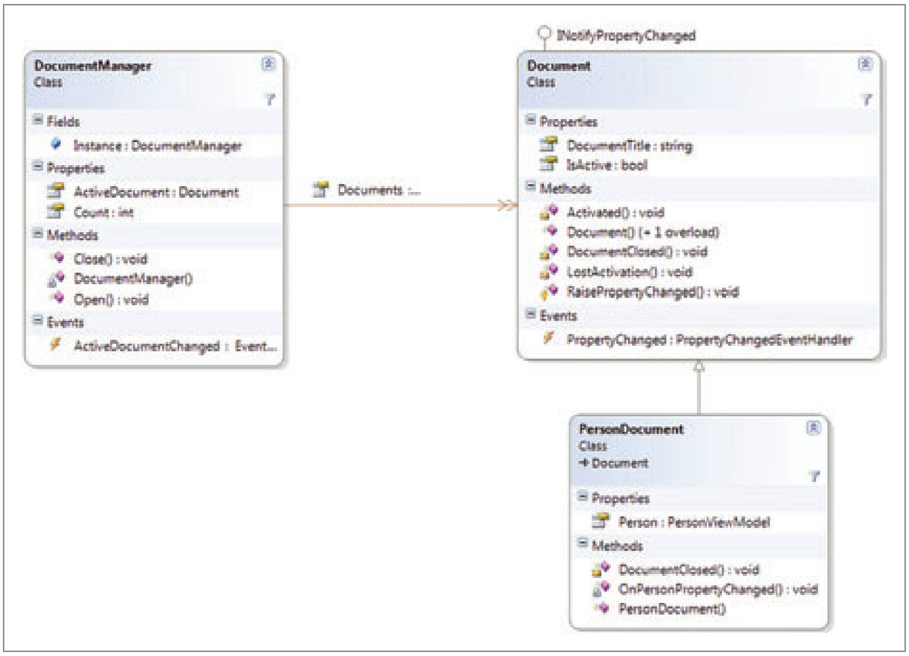


Figure 3 **The ViewModel Layer's Document Manager and Document Classes**

Figure 4 **Linking the View Control and ViewModel Document**

```
private static readonly DependencyProperty
  ViewModelDocumentProperty =
  DependencyProperty.RegisterAttached(
  "ViewModelDocument", typeof(Document),
  typeof(DocumentManagerAdapter), null);

private static Document GetViewModelDocument(
  AvalonDock.ManagedContent viewDoc) {

  return viewDoc.GetValue(ViewModelDocumentProperty)
    as Document;
}

private static void SetViewModelDocument(
  AvalonDock.ManagedContent viewDoc, Document document) {

  viewDoc.SetValue(ViewModelDocumentProperty, document);
}
```

Figure 5 **Setting the Attached Property**

```
private AvalonDock.DocumentContent CreateNewViewDocument(
  Document viewModelDocument) {

  var viewDoc = new AvalonDock.DocumentContent();
  viewDoc.DataContext = viewModelDocument;
  viewDoc.Content = viewModelDocument;

  Binding titleBinding = new Binding("DocumentTitle") {
    Source = viewModelDocument };

  viewDoc.SetBinding(AvalonDock.ManagedContent.TitleProperty,
    titleBinding);
  viewDoc.Closing += OnUserClosingDocument;
  DocumentManagerAdapter.SetViewModelDocument(viewDoc,
    viewModelDocument);

  return viewDoc;
}
```

up-to-date. For example, when the document manager indicates that a new document has been opened, the adapter receives an event, wraps the ViewModel document in whatever WPF control is required and then exposes that control in the tabbed workspace.

The adapter has one other responsibility: keeping the ViewModel document manager synchronized with the user's actions. The adapter must therefore also listen for events from the tabbed workspace control so that when the user changes the active document or closes a document the adapter can notify the document manager.

While none of this logic is very complex, there are some caveats. There are several scenarios where the code becomes re-entrant, and this must be handled gracefully. For example, if the ViewModel uses the document manager to close a document, the adapter will receive the event from the document manager and close the physical document window in the view. This causes the tabbed workspace control to also raise a document-closing event, which the adapter will also receive, and the adapter's event handler will, of course, notify the document manager that the document should be closed. The document has already been closed, so the document manager needs to be sympathetic enough to allow this.

The other difficulty is that the View's adapter must be able to link a View tabbed-document control with a ViewModel Document object. The most robust solution is to use a WPF attached dependency property. The adapter declares a private attached dependency property that's used to link the View window control to its ViewModel document instance.

In the sample project for this article, I use an open source tabbed workspace component called AvalonDock, so my attached dependency property looks like the code shown in **Figure 4**.

When the adapter creates a new View window control, it sets the attached property on the new window control to the underlying ViewModel document (see **Figure 5**). You can also see the title data binding being configured here, and see how the adapter is configuring both the data context and the content of the View document control.

By setting the View document control's content, I let WPF do the heavy lifting of figuring out how to display this particular type of ViewModel document. The actual data templates for the ViewModel documents are in a resource dictionary included by the main XAML window.

I've used this ViewModel document-manager approach with both WPF and Silverlight successfully. The only View layer code is the adapter, and this can be tested easily and then left alone. This approach keeps the ViewModel completely independent of the View, and I have on one occasion switched vendors for my tabbed workspace component with only minimal changes in the adapter class and absolutely no changes to the ViewModel or Model.

The ability to work with documents in the ViewModel layer feels elegant, and implementing ViewModel commands like the one I demonstrated here is easy. The ViewModel document classes also become obvious places to expose ICommand instances related to the document.

Observable collections in the Model open up new possibilities.

The View hooks into these commands and the beauty of the MVVM design pattern shines through. Additionally, the ViewModel document manager approach also works with the singleton approach if you need to expose data before the user has created any documents (perhaps in a collapsible tool window).

Wrap Up

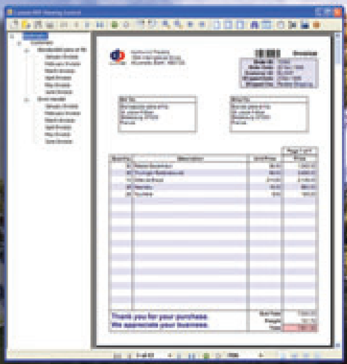
The MVVM design pattern is a powerful and useful pattern, but no design pattern can solve every issue. As I've demonstrated here, combining the MVVM pattern and goals with other patterns, such as adapters and singletons, while also leveraging new .NET Framework 4 features, such as dynamic dispatch, can help address many common concerns around implementing the MVVM design pattern. Employing MVVM the right way makes for much more elegant and maintainable WPF and Silverlight applications. For further reading about MVVM, see Josh Smith's article in the February 2009 issue of *MSDN Magazine* at msdn.microsoft.com/magazine/dd419663. ■

ROBERT MCCARTER *is a Canadian freelance software developer, architect and entrepreneur. Read his blog at robertmccarter.wordpress.com.*

THANKS to the following technical expert for reviewing this article:
Josh Smith

DynamicPDF

Dependable Developer Components



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.

Try our three new products FREE today!

Fully functional and never expiring evaluation editions available at www.cete.com/download

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.

ceTesoftware
INFINITE POSSIBILITIES

www.cete.com
info@cete.com

800.631.5006
+1 410.772.8620

SECURITY BRIEFS

BRYAN SULLIVAN

View State Security

Effectively managing user state in Web applications can be a tricky balancing act of performance, scalability, maintainability and security. The security consideration is especially evident when you're managing user state stored on the client. I have a colleague who used to say that handing state data to a client is like handing an ice cream cone to a 5-year-old: you may get it back, but you definitely can't expect to get it back in the same shape it was when you gave it out!

In this month's column, we'll examine some security implications around client-side state management in ASP.NET applications; specifically, we're going to look at view state security. (Please note: this article assumes that you're familiar with the concept of ASP.NET view state. If not, check out "Understanding ASP.NET View State" by Scott Mitchell at msdn.microsoft.com/library/ms972976).

If you don't think there's any data stored in your applications' view state worth protecting, think again. Sensitive information can find its way into view state without you even realizing it. And even if you're vigilant about preventing sensitive information loss through view state, an attacker can still tamper with that view state and cause even bigger problems for you and your users. Luckily, ASP.NET has some built-in defenses against these attacks. Let's take a look at how these defenses can be used correctly.

Threat No. 1: Information Disclosure

At Microsoft, development teams use the STRIDE model to classify threats. STRIDE is a mnemonic that stands for:

- Spoofing
- Tampering
- Repudiation
- Information Disclosure
- Denial of Service
- Elevation of Privilege

The main two STRIDE categories of concern from the view state security perspective are Information Disclosure and Tampering (although a successful tampering attack can lead to a possible Elevation of Privilege; we'll discuss that in more detail later). Information disclosure is the simpler of these threats to explain, so we'll discuss that first.

One of the most unfortunately persistent misconceptions around view state is that it is encrypted or somehow unreadable by the user. After all, a view state string certainly doesn't look decomposable:

```
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE" value="/
wEPDwULLTE2MTY2ODcyMjkPFgIeCHBhc3N3b3JkBQlzd29yZGZpc2hkZA==" />
```

However, this string is merely base64-encoded, not encrypted with any kind of cryptographically strong algorithm. We can easily decode and deserialize this string using the limited object serialization (LOS) formatter class System.Web.UI.LosFormatter:

```
LosFormatter formatter = new LosFormatter();
object viewstateObj = formatter.Deserialize("/
wEPDwULLTE2MTY2ODcyMjkPFgIeCHBhc3N3b3JkBQlzd29yZGZpc2hkZA==");
```

A quick peek in the debugger (see **Figure 1**) reveals that the deserialized view state object is actually a series of System.Web.UI.Pair objects ending with a System.Web.UI.IndexedString object with a value of "password" and a corresponding string value of "swordfish."

Encryption does not provide defense against tampering. Even with encrypted data, it's still possible for an attacker to flip bits in the encrypted data.

If you don't want to go to the trouble of writing your own code to deserialize view state objects, there are several good view state decoders available for free download on the Internet, including Fritz Onion's ViewState Decoder tool available at alt.pluralsight.com/tools.aspx.

Encrypting View State

In "The Security Development Lifecycle: SDL: A Process for Developing Demonstrably More Secure Software" (Microsoft Press, 2006), Michael Howard and Steve Lipner discuss technologies that can be used to mitigate STRIDE threats. **Figure 2** shows threat types and their associated mitigation techniques.

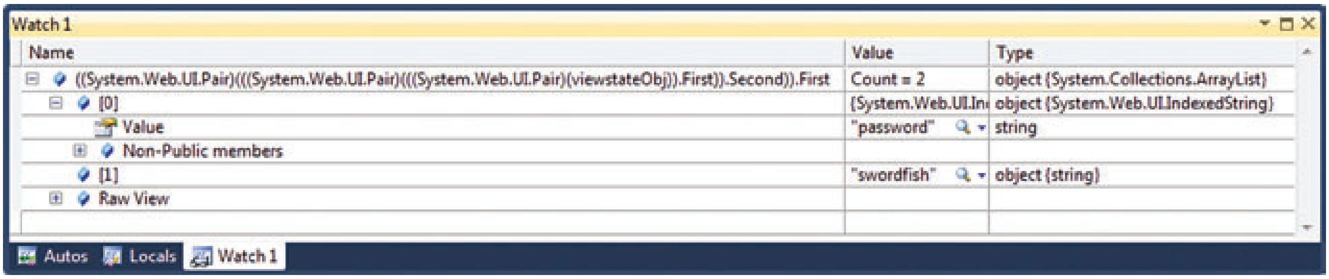
Because we're dealing with an information disclosure threat to our data stored in the view state, we need to apply a confidentiality mitigation technique; the most effective confidentiality mitigation technology in this case is encryption.

ASP.NET version 2.0 has a built-in feature to enable encryption of view state—the ViewStateEncryptionMode property, which can be enabled either through a page directive or in the application's web.config file:

```
<%@ Page ViewStateEncryptionMode="Always" %>
```

Or

```
<configuration>
   <system.web>
      <pages viewStateEncryptionMode="Always">
```



Name	Value	Type
((System.Web.UI.Pair)(((System.Web.UI.Pair)(((System.Web.UI.Pair)(viewstateObj)).First)).Second)).First	Count = 2	object {System.Collections.ArrayList}
[0]	{System.Web.UI.In	object {System.Web.UI.IndexedString}
Value	"password"	string
Non-Public members		
[1]	"swordfish"	object {string}
Raw View		

Figure 1 **Secret View State Data Revealed by the Debugger**

There are three possible values for ViewStateEncryptionMode: Always (the view state is always encrypted); Never (the view state is never encrypted); and Auto (the view state is only encrypted if one of the page's controls explicitly requests it). The Always and Never values are pretty self-explanatory, but Auto requires a little more explanation.

If a server control persists sensitive information into its page's view state, the control can request that the page encrypt the view state by calling the Page.RegisterRequiresViewStateEncryption method (note that in this case the entire view state is encrypted, not just the view state corresponding to the control that requested it):

```
public class MyServerControl : WebControl
{
   protected override void OnInit(EventArgs e)
   {
      Page.RegisterRequiresViewStateEncryption();
      base.OnInit(e);
   }
   ...
}
```

However, there is a caveat. The reason the method is named RegisterRequiresViewStateEncryption, and not something like EnableViewStateEncryption, is because the page can choose to ignore the request. If the page's ViewStateEncryptionMode is set to Auto (or Always), the control's request will be granted and the view state will be encrypted. If ViewStateEncryptionMode is set to Never, the control's request will be ignored and the view state will be unprotected.

This is definitely something to be aware of if you're a control developer. You should consider keeping potentially sensitive information out of the view state (which is always a good idea). In extreme cases where this isn't possible, you might consider overriding the control's SaveViewState and LoadViewState methods to manually encrypt and decrypt the view state there.

Server Farm Considerations

In a single-server environment, it's sufficient just to enable ViewStateEncryptionMode, but in a server farm environment there's some additional work to do. Symmetric encryption algorithms—like the ones that ASP.NET uses to encrypt the view state—require a key. You can either explicitly specify a key in the web.config file, or ASP.NET can automatically generate a key for you. Again, in a single-server environment it's fine to let the framework handle key generation, but this won't work for a server farm. Each server will generate its own unique key, and requests that get load balanced between different servers will fail because the decryption keys won't match.

You can explicitly set both the cryptographic algorithm and the key to use in the machineKey element of your application's web.config file:

```
<configuration>
  <system.web>
    <machineKey decryption="AES" decryptionKey="143a…">
```

For the encryption algorithm, you can choose AES (the default value), DES or 3DES. Of these, DES is explicitly banned by the Microsoft SDL Cryptographic Standards, and 3DES is strongly discouraged. I recommend that you stick with AES for maximum security.

Once you've selected an algorithm, you need to create a key. However, remember that the strength of this system's security depends on the strength of that key. Don't use your pet's name, your significant other's birthday or any other easily guessable value! You need to use a cryptographically strong random number. Here's a code snippet to create one in the format that the machineKey element expects (hexadecimal characters only) using the .NET RNGCryptoServiceProvider class:

```
RNGCryptoServiceProvider csp = new RNGCryptoServiceProvider();
byte[] data = new byte[24];
csp.GetBytes(data);
string value = String.Join("", BitConverter.ToString(data).Split('-'));
```

At a minimum, you should generate 16-byte random values for your keys; this is the minimum value allowed by the SDL Cryptographic Standards. The maximum length supported for AES keys is 24 bytes (48 hex chars) in the Microsoft .NET Framework 3.5 and earlier, and 32 bytes (64 hex chars) in the .NET Framework 4. DES supports a maximum key length of only 8 bytes and 3DES a maximum of 24 bytes, regardless of the framework version. Again, I recommend that you avoid these algorithms and use AES instead.

Threat No. 2: Tampering

Tampering is the other significant threat. You might think the same encryption defense that keeps attackers from prying into the view state would also prevent them from changing it, but this is wrong. Encryption doesn't provide defense against tampering: Even with encrypted data, it's still possible for an attacker to flip bits in the encrypted data.

Figure 2 **Techniques to Mitigate STRIDE Threats**

Threat Type	Mitigation Technique
Spoofing	Authentication
Tampering	Integrity
Repudiation	Non-repudiation services
Information Disclosure	Confidentiality
Denial of Service	Availability
Elevation of Privilege	Authorization

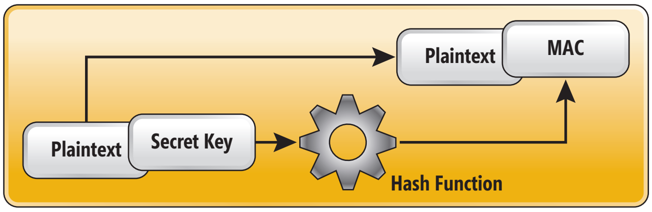


Figure 3 **Applying a Message Authentication Code (MAC)**

Take another look at **Figure 2**. To mitigate a tampering threat, we need to use a data integrity technology. The best choice here is still a form of cryptography, and it's still built into ASP.NET, but instead of using a symmetric algorithm to encrypt the data, we'll use a hash algorithm to create a message authentication code (MAC) for the data.

The ASP.NET feature to apply a MAC is called EnableViewStateMac, and just like ViewStateEncryptionMode, you can apply it either through a page directive or through the application's web.config file:

```
<%@ Page EnableViewStateMac="true" %>
```

Or

```
<configuration>
  <system.web>
    <pages enableViewStateMac="true">
```

To understand what EnableViewStateMac is really doing under the covers, let's first take a high-level look at how view state is written to the page when view state MAC is *not* enabled:

1. View state for the page and all participating controls is gathered into a state graph object.
2. The state graph is serialized into a binary format.
3. The serialized byte array is encoded into a base-64 string.
4. The base-64 string is written to the __VIEWSTATE form value in the page.

When view state MAC is enabled, there are three additional steps that take place between the previous steps 2 and 3:

1. View state for the page and all participating controls is gathered into a state graph object.
2. The state graph is serialized into a binary format.
   a. A secret key value is appended to the serialized byte array.
   b. A cryptographic hash is computed for the new serialized byte array.
   c. The hash is appended to the end of the serialized byte array.
3. The serialized byte array is encoded into a base-64 string.
4. The base-64 string is written to the __VIEWSTATE form value in the page.

Whenever this page is posted back to the server, the page code validates the incoming __VIEWSTATE by taking the incoming state graph data (deserialized from the __VIEWSTATE value), adding the same secret key value, and recomputing the hash value. If the new recomputed hash value matches the hash value supplied at the end of the incoming __VIEWSTATE, the view state is considered valid and processing proceeds (see **Figure 3**). Otherwise, the view state is considered to have been tampered with and an exception is thrown.

The security of this system lies in the secrecy of the secret key value. This value is always stored on the server, either in memory or in a configuration file (more on this later)—it is never written to the page. Without knowing the key, there would be no way for an attacker to compute a valid view state hash.

Theoretically, with enough computing power an attacker could reverse-engineer the key: He has knowledge of a computed hash value and knowledge of the corresponding plaintext, and there aren't too many options available for the hash algorithm. He would only have to cycle through all the possible key values, re-compute the hash for the known plaintext plus the current key and compare it to the known hash. Once the values match, he knows he's found the correct key and can now attack the system at will. The only problem with this is the sheer number of possible values: The default key size is 512 bits, which means there are 2 to the power of 512 different possibilities, which is so large a number that a brute force attack is completely unfeasible.

Exploiting MAC-Less View State

The default value of EnableViewStateMac is true, so protecting your applications is as simple as not setting it to false. Unfortunately, there is some misleading documentation concerning the performance impact of EnableViewStateMac, and some Web sites are encouraging developers to disable view state MAC in order to improve the performance of their applications. Even the MSDN online documentation for PagesSection.EnableViewStateMacProperty is guilty of this, stating: "Do not set EnableViewStateMac to true if performance is a key consideration." Do not follow this advice! (Hopefully, by the time you're reading this, the documentation will have been changed to better reflect security considerations.)

> DES is explicitly banned by the Microsoft SDL Cryptographic Standards and 3DES is strongly discouraged. I recommend that you stick with AES for maximum security.

Any page that has its view state MAC-disabled is potentially vulnerable to a cross-site scripting attack against the __VIEWSTATE parameter. The first proof-of-concept of this attack was developed by David Byrne of Trustwave, and demonstrated by Byrne and his colleague Rohini Sulatycki at the Black Hat DC conference in February 2010. To execute this attack, the attacker crafts a view state graph where the malicious script code he wants to execute is set as the persisted value of the innerHtml property of the page's form element. In XML form, this view state graph would look something like **Figure 4**.

The attacker then base-64 encodes the malicious view state and appends this string as the value of a __VIEWSTATE query string parameter for the vulnerable page. For example, if the page

Figure 4 **XML Code for View State MAC Attack**

```
<viewstate>
  <Pair>
    <Pair>
      <String>…</String>
      <Pair>
        <ArrayList>
          <Int32>0</Int32>
          <Pair>
            <ArrayList>
              <Int32>1</Int32>
              <Pair>
                <ArrayList>
                  <IndexedString>innerhtml</IndexedString>
                  <String>…malicious script goes here…</String>
                </ArrayList>
              </Pair>
            </ArrayList>
          </Pair>
        </ArrayList>
      </Pair>
    </Pair>
  </Pair>
</viewstate>
```

home.aspx on the site www.contoso.com was known to have view state MAC disabled, the attack URI would be http://www.contoso.com/home.aspx?__VIEWSTATE=/w143a…

All that remains is to trick a potential victim into following this link. Then the page code will deserialize the view state from the incoming __VIEWSTATE query string parameter and write the malicious script as the innerHtml of the form. When the victim gets the page, the attacker's script will immediately execute in the victim's browser, with the victim's credentials.

This attack is especially dangerous because it completely bypasses all of the usual cross-site scripting (XSS) defenses. The XSS Filter in Internet Explorer 8 will not block it. The ValidateRequest feature of ASP.NET will block several common XSS attack vectors, but it does not deserialize and analyze incoming view state, so it's also no help in this situation. The Microsoft Anti-Cross Site Scripting (Anti-XSS) Library (now included as part of the Microsoft Web Protection Library) is even more effective against XSS than ValidateRequest; however, neither the Anti-XSS Library input sanitization features nor its output encoding features will protect against this attack either. The only real defense is to ensure that view state MAC is consistently applied to all pages.

More Server Farm Considerations

Similar to ViewStateEncryptionMode, there are special considerations with EnableViewStateMac when deploying applications in a server farm environment. The secret value used for the view state hash must be constant across all machines in the farm, or the view state validation will fail.

You can specify both the validation key and the HMAC algorithm to use in the same location where you specify the view state encryption key and algorithm—the machineKey element of the web.config file:

```
<configuration>
  <system.web>
    <machineKey validation="AES" validationKey="143a...">
```

If your application is built on the .NET Framework 3.5 or earlier, you can choose SHA1 (the default value), AES, MD5 or 3DES as the MAC algorithm. If you're running .NET Framework 4, you can also choose MACs from the SHA-2 family: HMACSHA256, HMACSHA384 or HMACSHA512. Of these choices, MD5 is explicitly banned by the SDL Crypto Standards and 3DES is strongly discouraged. SHA1 is also discouraged, but for .NET Framework 3.5 and earlier applications it's your best option. .NET Framework 4 applications should definitely be configured with either HMACSHA512 or HMACSHA256 as the validation algorithm.

After you choose a MAC algorithm, you'll also need to manually specify the validation key. Remember to use cryptographically strong random numbers: if necessary, you can refer to the key generation code specified earlier. You should use at least 128-byte validation keys for either HMACSHA384 or HMACSHA512, and at least 64-byte keys for any other algorithm.

You Can't Hide Vulnerable View State

Unlike a vulnerable file permission or database command that may be hidden deep in the server-side code, vulnerable view state is easy to find just by looking for it. If an attacker wanted to test a page to see whether its view state was protected, he could simply make a request for that page himself and pull the base-64 encoded view state value from the __VIEWSTATE form value. If the LosFormatter class can successfully deserialize that value, then it has not been encrypted. It's a little trickier—but not much—to determine whether view state MAC has been applied.

The MAC is always applied to the end of the view state value, and since hash sizes are constant for any given hash algorithm, it's fairly easy to determine whether a MAC is present. If HMACSHA512 has been used, the MAC will be 64 bytes; if HMACSHA384 has been used, it will be 48 bytes, and if any other algorithm has been used it will be 32 bytes. If you strip 32, 48 or 64 bytes off of the end of the base-64 decoded view state value, and any of these deserialize with LosFormatter into the same object as before, then view state MAC has been applied. If none of these trimmed view state byte arrays will successfully deserialize, then view state MAC hasn't been applied and the page is vulnerable.

Casaba Security makes a free tool for developers called Watcher that can help automate this testing. Watcher is a plug-in for Eric Lawrence's Fiddler Web debugging proxy tool, and it works by passively analyzing the HTTP traffic that flows through the proxy. It will flag any potentially vulnerable resources that pass through—for example, an .aspx page with a __VIEWSTATE missing a MAC. If you're not already using both Fiddler and Watcher as part of your testing process, I highly recommend giving them a try.

Wrapping Up

View state security is nothing to take lightly, especially considering the new view state tampering attacks that have recently been demonstrated. I encourage you to take advantage of the ViewStateEncryptionMode and EnableViewStateMac security mechanisms built into ASP.NET. ■

BRYAN SULLIVAN *is a security program manager for the Microsoft Security Development Lifecycle team, where he specializes in Web application security issues. He's the author of "Ajax Security" (Addison-Wesley, 2007).*

THANKS to the following technical expert for reviewing this article: *Michael Howard*

AUGUST 2–6, 2010

REDMOND, WA | MICROSOFT CAMPUS

VSLive!

Join us at Microsoft HQ!

KEYNOTES ANNOUNCED!

Get the insider's view of Microsoft with:

JASON ZANDER
Corporate Vice President
Visual Studio
Microsoft Corporation

DAVE MENDLEN
Senior Director
Developer Platform and Tools
Microsoft Corporation

Our 70 session conference program is packed with the app-kicking tips and tools you need to take your code to the next level.

PLUS: you'll have EXCLUSIVE access to check out Microsoft's corporate headquarters—INCLUDING the Microsoft employee store!

Supported by:

Microsoft msdn Visual Studio MAGAZINE Microsoft Visual Studio

Platinum Sponsors

CodeFluent Entities VERSANT

Gold Sponsors

AvePoint ESRI

1105 MEDIA

ATTEND VSLIVE! REDMOND
AND CODE ONE-ON-ONE WITH THE PROS!

SPEAKERS YOU DON'T WANT TO MISS:

ROCKFORD LHOTKA
Principal Technology Evangelist
Magenic Technologies

BRIAN RANDELL
Senior Consultant
MCW Technologies

BILLY HOLLIS
Consultant
DotNetMasters

MIGUEL CASTRO
Architect
IDesign

TIM HUCKABY
CEO/Founder
InterKnowlogy

TED PATTISON
Co-Founder, Author
Critical Path Training

Reserve Your Place on Campus Before It's Too Late!

REGISTER TODAY AT

VSLIVE.COM/REDMOND

USE PRIORITY CODE MSDN8

THE WORKING PROGRAMMER

TED NEWARD

Going NoSQL with MongoDB, Part 3

Last time, I continued my exploration of MongoDB via the use of exploration tests. I described how to start and stop the server during a test, then showed how to capture cross-document references and discussed some of the reasoning behind the awkwardness of doing so. Now it's time to explore some more intermediate MongoDB capabilities: predicate queries, aggregate functions and the LINQ support provided by the MongoDB.Linq assembly. I'll also provide some notes about hosting MongoDB in a production environment.

When We Last Left Our Hero . . .

For reasons of space, I won't review much of the previous articles; instead, you can read them online in the May and June issues at msdn.microsoft.com/magazine. In the associated code bundle, however, the exploration tests have been fleshed out to include a pre-existing sample set of data to work with, using characters from one of my favorite TV shows. **Figure 1** shows a previous exploration test, by way of refresher. So far, so good.

Calling All Old People . . .

In previous articles, the client code has fetched either all documents that match a particular criteria (such as having a "lastname" field matching a given String or an "_id" field matching a particular Oid), but I haven't discussed how to do predicate-style queries (such as "find all documents where the 'age' field has a value higher than 18"). As it turns out, MongoDB doesn't use a SQL-style interface to describe the query to execute; instead, it uses ECMAScript/JavaScript, and it can in fact accept blocks of code to execute on the server to filter or aggregate data, almost like a stored procedure.

This provides some LINQ-like capabilities, even before looking at the LINQ capabilities supported by the Mongo.Linq assembly. By specifying a document containing a field named "$where" and a code block describing the ECMAScript code to execute, arbitrarily complex queries can be created:

```
[TestMethod]
public void Where()
{
  ICursor oldFolks =
    db["exploretests"]["familyguy"].Find(
    new Document().Append("$where",
    new Code("this.gender === 'F'")));
  bool found = false;
  foreach (var d in oldFolks.Documents)
    found = true;
  Assert.IsTrue(found, "Found people");
}
```

As you can see, the Find call returns an ICursor instance, which, although itself isn't IEnumerable (meaning it can't be used in the foreach loop), contains a Documents property that's an IEnumerable<Document>. If the query would return too large a set of data, the ICursor can be limited to return the first *n* results by setting its Limit property to *n*.

The predicate query syntax comes in four different flavors, shown in **Figure 2**.

In the second and third forms, "this" always refers to the object being examined.

You can send any arbitrary command (that is, ECMAScript code) through the driver to the database, in fact, using documents to convey the query or command. So, for example, the Count method provided by the IMongoCollection interface is really just a convenience around this more verbose snippet:

```
[TestMethod]
public void CountGriffins()
{
  var resultDoc = db["exploretests"].SendCommand(
    new Document()
      .Append("count", "familyguy")
      .Append("query",
        new Document().Append("lastname", "Griffin"))
    );
  Assert.AreEqual(6, (double)resultDoc["n"]);
}
```

This means that any of the aggregate operations described by the MongoDB documentation, such as "distinct" or "group," for example, are accessible via the same mechanism, even though they may not be surfaced as methods on the MongoDB.Driver APIs.

You can send arbitrary commands outside of a query to the database via the "special-name" syntax "$eval," which allows any legitimate ECMAScript block of code to be executed against the server, again essentially as a stored procedure:

```
[TestMethod]
public void UseDatabaseAsCalculator()
{
  var resultDoc = db["exploretests"].SendCommand(
    new Document()
      .Append("$eval",
        new CodeWScope {
          Value = "function() { return 3 + 3; }",
          Scope = new Document() }));
  TestContext.WriteLine("eval returned {0}", resultDoc.ToString());
  Assert.AreEqual(6, (double)resultDoc["retval"]);
}
```

Or, use the provided Eval function on the database directly. If that isn't flexible enough, MongoDB permits the storage of user-defined ECMAScript functions on the database instance

Code download available at code.msdn.microsoft.com/mag201007WorkProg.

Figure 1 **An Example Exploration Test**

```
[TestMethod]
   public void StoreAndCountFamilyWithOid()
   {
     var oidGen = new OidGenerator();
     var peter = new Document();
     peter["firstname"] = "Peter";
     peter["lastname"] = "Griffin";
     peter["_id"] = oidGen.Generate();

     var lois = new Document();
     lois["firstname"] = "Lois";
     lois["lastname"] = "Griffin";
     lois["_id"] = oidGen.Generate();

     peter["spouse"] = lois["_id"];
     lois["spouse"] = peter["_id"];

     var cast = new[] { peter, lois };
     var fg = db["exploretests"]["familyguy"];
     fg.Insert(cast);

     Assert.AreEqual(peter["spouse"], lois["_id"]);
     Assert.AreEqual(
       fg.FindOne(new Document().Append("_id",
         peter["spouse"])).ToString(), lois.ToString());

     Assert.AreEqual(2,
       fg.Count(new Document().Append("lastname", "Griffin")));
   }
```

for execution during queries and server-side execution blocks by adding ECMAScript functions to the special database collection "system.js," as described on the MongoDB Web site (MongoDB.org).

The Missing LINQ

The C# MongoDB driver also has LINQ support, allowing developers to write MongoDB client code such as what's shown in **Figure 3**.

And, in keeping with the dynamic nature of the MongoDB database, this sample requires no code-generation, just the call to Linq to return an object that "enables" the MongoDB LINQ provider. At the time of this writing, LINQ support is fairly rudimentary, but it's being improved and by the time this article reaches print, it will be significantly better. Documentation of the new features and examples will be in the wiki of the project site (wiki.github.com/samus/mongodb-csharp/).

Figure 2 **Four Different Predicate Query Syntaxes**

```
[TestMethod]
public void PredicateQuery()
{
  ICursor oldFolks =
    db["exploretests"]["familyguy"].Find(
    new Document().Append("age",
    new Document().Append("$gt", 18)));
  Assert.AreEqual(6, CountDocuments(oldFolks));

  oldFolks =
    db["exploretests"]["familyguy"].Find(
    new Document().Append("$where",
    new Code("this.age > 18")));
  Assert.AreEqual(6, CountDocuments(oldFolks));

  oldFolks =
    db["exploretests"]["familyguy"].Find("this.age > 18");
  Assert.AreEqual(6, CountDocuments(oldFolks));

  oldFolks =
    db["exploretests"]["familyguy"].Find(
    new Document().Append("$where",
    new Code("function(x) { return this.age > 18; }")));
  Assert.AreEqual(6, CountDocuments(oldFolks));
}
```

Shipping Is a Feature

Above all else, if MongoDB is going to be used in a production environment, a few things need to be addressed to make it less painful for the poor chaps who have to keep the production servers and services running.

To begin, the server process (mongod.exe) needs to be installed as a service—running it in an interactive desktop session is typically not allowed on a production server. To that end, mongod.exe supports a service install option, "--install," which installs it as a service that can then be started either by the Services panel or the command line: "net start MongoDB." However, as of this writing, there's one small quirk in the --install command—it infers the path to the executable by looking at the command line used to execute it, so the full path must be given on the command line. This means that if MongoDB is installed in C:\Prg\mongodb, you must install it as a service at a command prompt (with administrative rights) with the command C:\Prg\mongodb\bin\mongod.exe --install.

However, any command-line parameters, such as "--dbpath, " must also appear in that installation command, which means if any of the settings—port, path to the data files and so on—change, the service must be reinstalled. Fortunately, MongoDB supports a configuration file option, given by the "--config" command-line option, so typically the best approach is to pass the full config file path to the service install and do all additional configuration from there:

```
C:\Prg\mongodb\bin\mongod.exe --config C:\Prg\mongodb\bin\mongo.cfg --install
net start MongoDB
```

As usual, the easiest way to test to ensure the service is running successfully is to connect to it with the mongo.exe client that ships with the MongoDB download. And, because the server communicates with the clients via sockets, you need to poke the required holes in the firewall to permit communication across servers.

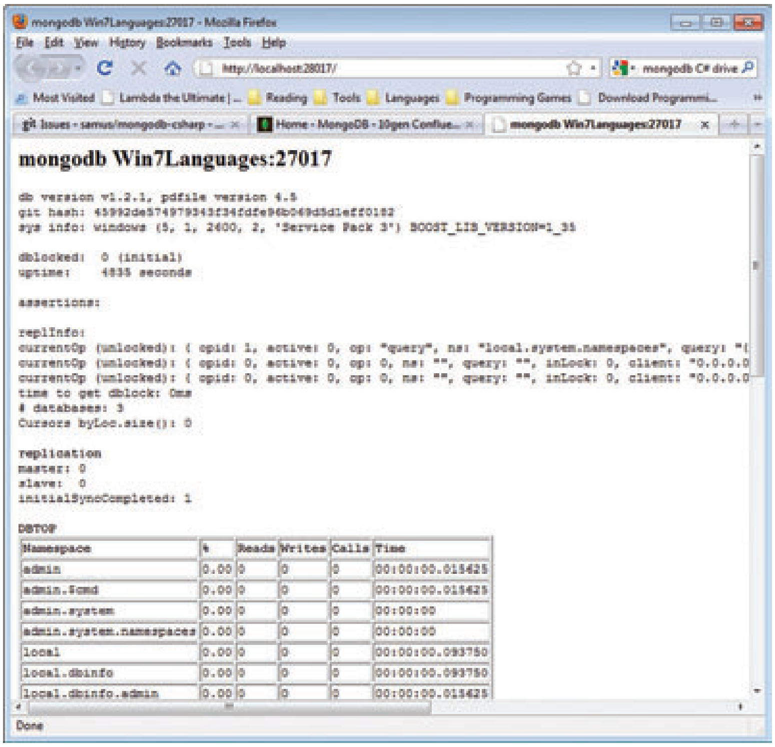
These Aren't the Data Droids You're Looking For

Of course, unsecured access to the MongoDB server isn't likely to be a good thing, so securing the server against unwanted visitors becomes a key feature. MongoDB supports authentication, but the security system isn't anywhere near as sophisticated as that found with "big iron" databases such as SQL Server.

Typically, the first step is to create a database admin login by connecting to the database with the mongo.exe client and adding an

Figure 3 **An Example of LINQ Support**

```
[TestMethod]
public void LINQQuery()
{
  var fg = db["exploretests"]["familyguy"];
  var results =
    from d in fg.Linq()
    where ((string)d["lastname"]) == "Brown"
    select d;
  bool found = false;
  foreach (var d in results)
  {
    found = true;
    TestContext.WriteLine("Found {0}", d);
  }
  Assert.IsTrue(found, "No Browns found?");
}
```

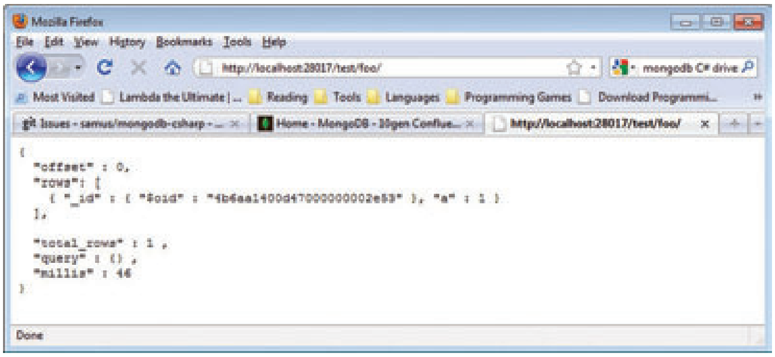

mongodb Win7Languages:27017 - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:28017/
mongodb C# drive
Most Visited
Lambda the Ultimate | ...
Reading
Tools
Languages
Programming Games
Download Programmi...
Issues - samus/mongodb-csharp - ...
Home - MongoDB - 10gen Conflue...
mongodb Win7Languages:27017
Done

mongodb Win7Languages:27017

```
db version v1.2.1, pdfile version 4.5
git hash: 45992de574979343f34fdfe96b049d5d1eff0182
sys info: windows (5, 1, 2600, 2, 'Service Pack 3') BOOST_LIB_VERSION=1_35

dblocked:  0 (initial)
uptime:    4835 seconds

assertions:

replInfo:
currentOp (unlocked): { opid: 1, active: 0, op: "query", ns: "local.system.namespaces", query: "{
currentOp (unlocked): { opid: 0, active: 0, op: 0, ns: "", query: "", inLock: 0, client: "0.0.0.0
currentOp (unlocked): { opid: 0, active: 0, op: 0, ns: "", query: "", inLock: 0, client: "0.0.0.0
time to get dblock: 0ms
# databases: 3
Cursors byLoc.size(): 0

replication
master: 0
slave:  0
initialSyncCompleted: 1

DBTOP
```

Namespace	%	Reads	Writes	Calls	Time
admin	0.00	0	0	0	00:00:00.015625
admin.$cmd	0.00	0	0	0	00:00:00.015625
admin.system	0.00	0	0	0	00:00:00
admin.system.namespaces	0.00	0	0	0	00:00:00
local	0.00	0	0	0	00:00:00.093750
local.dbinfo	0.00	0	0	0	00:00:00.093750
local.dbinfo.admin	0.00	0	0	0	00:00:00.015625

Figure 4 **The HTTP Interface for Interacting with MongoDB**


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:28017/test/foo/
mongodb C# drive
Most Visited
Lambda the Ultimate | ...
Reading
Tools
Languages
Programming Games
Download Programmi...
Issues - samus/mongodb-csharp - ...
Home - MongoDB - 10gen Conflue...
http://localhost:28017/test/foo/
Done

```
{
  "offset" : 0,
  "rows": [
    { "_id" : { "$oid" : "4b6aa1400d47000000002e53" }, "a" : 1 }
  ],

  "total_rows" : 1 ,
  "query" : {} ,
  "millis" : 46
}
```

Figure 5 **The HTTP URL for Accessing a Collection's Contents**

admin user to the admin database (a database containing data for running and administering the entire MongoDB server), like so:

```
> use admin
> db.addUser("dba", "dbapassword")
```

Once this is done, any further actions, even within this shell, will require authenticated access, which is done in the shell by explicit authentication:

```
> db.authenticate("dba", "dbapassword")
```

The DBA can now add users to a MongoDB database by changing databases and adding the user using the same addUser call shown earlier:

```
> use mydatabase
> db.addUser("billg", "password")
```

When connecting to the database via the Mongo.Driver, pass the authentication information as part of the connection string used to create the Mongo object and the same authentication magic will happen transparently:

```
var mongo = new Mongo("Username=billg;Password=password");
```

Naturally, passwords shouldn't be hardcoded directly into the code or stored openly; use the same password discipline as befits any database-backed application. In fact, the entire configuration (host, port, password and so on) should be stored in a configuration file and retrieved via the ConfigurationManager class.

Reaching Out to Touch Some Code

Periodically, administrators will want to look at the running instance to obtain diagnostic information about the running server. MongoDB supports an HTTP interface for interacting with it, running on a port numerically 1,000 higher than the port it's configured to use for normal client communication. Thus, because the default MongoDB port is 27017, the HTTP interface can be found on port 28017, as shown in **Figure 4**.

This HTTP interface also permits a more REST-style communication approach, as opposed to the native driver in MongoDB.Driver and MongoDB.Linq; the MongoDB Web site has full details, but essentially the HTTP URL for accessing a collection's contents is given by adding the database name and collection name, separated by slashes, as shown in **Figure 5**.

For more details on creating a REST client using WCF, refer to the MSDN article "REST in Windows Communication Foundation (WCF)" at msdn.microsoft.com/netframework/cc950529.

A Word from Yoda

MongoDB is a quickly evolving product and these articles, while exploring core parts of MongoDB's functionality, still leave major areas unexamined. While MongoDB isn't a direct replacement for SQL Server, it's proving to be a viable storage alternative for areas where the traditional RDBMS doesn't fare so well. Similarly, just as MongoDB is an evolution in progress, so is the mongodb-csharp project. At the time of this writing, many new improvements were going into beta, including enhancements for working with strongly typed collections using plain objects, as well as greatly improved LINQ support. Keep an eye on both.

In the meantime, however, it's time to wave farewell to MongoDB and turn our attention to other parts of the developer's world that the working programmer may not be familiar with (and arguably should be). For now, though, happy coding, and remember, as the great DevGuy Master Yoda once said, "A DevGuy uses the Source for knowledge and defense; never for a hack." ■

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, INETA speaker and the author and coauthor of a dozen books, including "Professional F# 2.0" (Wrox, 2010). He consults and mentors regularly. Reach him at ted@tedneward.com and read his blog at blogs.tedneward.com.*

THANKS to the following technical experts for reviewing this article:
Sam Corder and Craig Wilson

UI FRONTIERS

CHARLES PETZOLD

The Fluid UI in Silverlight 4

The term "fluid UI" has recently become common to describe UI design techniques that avoid having visual objects suddenly pop into view or jump from one location to another. Instead, visually fluid objects make more graceful entrances and transitions—sometimes as if emerging from fog or sliding into view.

In the past two installments of this column, I've discussed some techniques implementing fluid UI on your own. I was partially inspired by the upcoming introduction of a fluid UI feature in Silverlight 4. Now that Silverlight 4 has been officially released, that's what I'll be covering here. Silverlight 4's foray into fluid UI is rather narrowly confined—it's restricted to the loading and unloading of items in a ListBox—but it gives us some important hints on how to extend fluid UI techniques with our own implementations. More fluid UI behaviors are available in Expression Blend 4.

Templates and the VSM

If you don't know exactly where to find the new fluid UI feature in Silverlight 4, you might search for many hours. It's not a class. It's not a property. It's not a method. It's not an event. It's actually implemented as three new visual states on the ListBoxItem class. **Figure 1** shows the documentation for that class, with the TemplateVisualState attribute items slightly rearranged in accordance with the group names.

The Visual State Manager (VSM) is one of the most significant changes made to Silverlight as it was being adapted from Windows Presentation Foundation. In WPF, a style or a template (almost always defined in XAML) can include elements called *triggers*. These triggers are defined to detect either a property change or an event, and then initiate an animation or a change to another property.

For example, a style definition for a control can include a trigger for the IsMouseOver property that sets the background of the control to a blue brush when the property is true. Or a trigger for the MouseEnter and MouseLeave events can initiate a couple of brief animations when those events occur.

In Silverlight, triggers have been largely banished and replaced with the VSM, partially to provide a more structured approach to dynamically changing the characteristics of a control at run time, and partially to avoid dealing with all the different combinations of possibilities when multiple triggers are defined. The VSM is considered to be such an improvement over triggers that it has become part of WPF in the Microsoft .NET Framework 4.

As you can see in **Figure 1**, the ListBoxItem control supports 11 visual states, but they're apportioned into four groups. Within any group, one and only one visual state is active at any time. This simple rule greatly reduces the number of possible combinations. For example, you don't have to figure out how the ListBoxItem should appear when the mouse is hovering over a selected but unfocused item; each group can be handled independently of the others.

The code part of ListBoxItem is responsible for changing visual states through calls to the static VisualStateManager.GoToState method. The control template for ListBoxItem is responsible for responding to these visual states. The template responds to a particular visual state change with a single Storyboard containing one or more animations that target elements in the visual tree. If you want the control to respond to a visual state change immediately without an animation, you can simply define the animation with a duration of 0. But why bother? It's just as easy to use an animation to help make the control's visuals more fluid.

> If you don't know exactly where to find the new fluid UI feature in Silverlight 4, you might search for many hours.

The new visual states for supporting fluid UI are BeforeLoaded, AfterLoaded and BeforeUnloaded, all part of the LayoutStates group. By associating animations to these visual states, you can make items in your ListBox fade in, or grow or glide into view when they're first added to the ListBox, and do something else when they're removed from the ListBox.

Adapting the ListBoxItem Template

Most programmers will probably access the fluid UI feature of ListBoxItem through Expression Blend, but I'm going to show you how to do it directly in markup.

The default control template for ListBoxItem has no animations associated with the visual states in the LayoutStates group. That's your job. Unfortunately, you can't just "derive from" the existing ListBoxItem template and supplement it with your own stuff. You must include the whole template in your program. Fortunately, it's a simple matter of copy and paste. In the Silverlight 4 documentation, look in the Controls section, and then Control Customization, and

Code download available at code.msdn.microsoft.com/mag201007UIFrontiers.

Figure 1 **The ListBoxItem Class Documentation**

```
[TemplateVisualStateAttribute(Name = "Normal", GroupName =
  "CommonStates")]
[TemplateVisualStateAttribute(Name = "MouseOver", GroupName =
  "CommonStates")]
[TemplateVisualStateAttribute(Name = "Disabled", GroupName =
  "CommonStates")]
[TemplateVisualStateAttribute(Name = "Unselected", GroupName =
  "SelectionStates")]
[TemplateVisualStateAttribute(Name = "Selected", GroupName =
  "SelectionStates")]
[TemplateVisualStateAttribute(Name = "SelectedUnfocused", GroupName =
  "SelectionStates")]
[TemplateVisualStateAttribute(Name = "Unfocused", GroupName =
  "FocusStates")]
[TemplateVisualStateAttribute(Name = "Focused", GroupName =
  "FocusStates")]
[TemplateVisualStateAttribute(Name = "BeforeLoaded", GroupName =
  "LayoutStates")]
[TemplateVisualStateAttribute(Name = "AfterLoaded", GroupName =
  "LayoutStates")]
[TemplateVisualStateAttribute(Name = "BeforeUnloaded", GroupName =
  "LayoutStates")]
public class ListBoxItem : ContentControl
```

Control Styles and Templates, and ListBox Styles and Templates. You'll find the default style definition for ListBoxItem (which includes the template definition) in the markup that begins:

```
<Style TargetType="ListBoxItem">
```

Under the Setter element for the Template property, you'll see the entire ControlTemplate used to build a visual tree for each ListBoxItem. The root of the visual tree is a single-cell Grid. The VSM markup occupies a large part of the template at the top of the Grid definition. At the bottom are the actual contents of the Grid: three Rectangle shapes (two filled and one just stroked) and a ContentPresenter, like so:

```
<Grid ... >
  ...
  <Rectangle x:Name="fillColor" ... />
  <Rectangle x:Name="fillColor2" ... />
  <ContentPresenter x:Name="contentPresenter" ... />
  <Rectangle x:Name="FocusVisualElement" ... />
</Grid>
```

> The Visual State Manager is one of the most significant changes made to Silverlight as it was being adapted from WPF.

The first two filled Rectangle objects are used to provide background shading for mouse-over and selection (respectively). The third displays a stroked rectangle to indicate input focus. The visibility of these rectangles is controlled by the VSM markup. Notice how each visual group gets its own element to manipulate. The ContentPresenter hosts the item as it's displayed in the ListBox. Generally, the content of the ContentPresenter is another visual tree defined in a DataTemplate that's set to the ItemTemplate property of ListBox.

The VSM markup consists of elements of type VisualStateManager.VisualStateGroups, VisualStateGroup and VisualState, all with an XML namespace prefix of "vsm." In earlier versions of Silverlight, it was necessary to define a namespace declaration for that prefix:

```
xmlns:vsm="clr-namespace:System.Windows;assembly=System.Windows"
```

However, in Silverlight 4 you can simply delete all the vsm prefixes and forget about this namespace declaration. To make changes to this template, you'll want to copy that whole section of markup into a resource section of a XAML file and give it a key name:

```
<Style x:Key="listBoxItemStyle" TargetType="ListBoxItem">
  ...
</Style>
```

You then set this style to the ItemContainerStyle property of the ListBox:

```
<ListBox ... ItemContainerStyle="{StaticResource listBoxItemStyle}" ....
```

The "item container" is the object the ListBox creates as a wrapper for each item in the ListBox, and that's an object of type ListBoxItem.

Once you have this ListBoxItem style and template in your program, you can make changes to it.

Fade in, Fade Out

Let's see how this works in the context of a simple demo program. The downloadable code for this article is a solution entitled FluidUserInterfaceDemo. It consists of two programs, which you can run from my Web site at charlespetzold.com/silverlight/FluidUserInterfaceDemo. Both programs are on the same HTML page, each occupying the whole browser window.

The first program is FluidListBox. Visually, it consists of a ListBox and two buttons to add and remove items. I've used the same collection of grocery produce that I've used in my last two columns, so MainPage.xaml also contains a DataTemplate named produceDataTemplate.

I decided I wanted to start off simple and have the items fade into view when they're added to the ListBox and fade out when they're removed. This involves animating the Opacity property of the Grid that forms the root of the visual tree. To be the target of an animation, that Grid needs a name:

```
<Grid Name="rootGrid" ...>
```

First insert a new VisualStateGroup within the VisualStateManager.VisualStateGroups tags:

```
<VisualStateGroup x:Name="LayoutStates">
  ...
</VisualStateGroup>
```

That's where the markup goes for the BeforeLoaded, AfterLoaded and BeforeUnloaded states in the LayoutStates group.

The fade-in is the easier of the two jobs. When an item is first added to the visual tree, it's said to be "loaded" into the visual tree. Prior to being loaded, the item has a visual state of BeforeLoaded, and then the visual state becomes AfterLoaded.

There are several ways to define the fade-in. The first requires initializing the Opacity to 0 in the Grid tag:

```
<Grid Name="rootGrid" Opacity="0" ... >
```

You then provide an animation for the AfterLoaded state to increase the Opacity property to 1 over the course of 1 second:

```
<VisualState x:Name="AfterLoaded">
  <Storyboard>
    <DoubleAnimation Storyboard.TargetName="rootGrid"
                     Storyboard.TargetProperty="Opacity"
                     To="1" Duration="0:0:1" />
  </Storyboard>
</VisualState>
```

Or you can leave the Grid opacity at its default value of 1 and provide animations for both BeforeLoaded and AfterLoaded:

```
<VisualState x:Name="BeforeLoaded">
  <Storyboard>
    <DoubleAnimation Storyboard.TargetName="rootGrid"
                     Storyboard.TargetProperty="Opacity"
                     To="0" Duration="0:0:0" />
  </Storyboard>
</VisualState>

<VisualState x:Name="AfterLoaded">
  <Storyboard>
    <DoubleAnimation Storyboard.TargetName="rootGrid"
                     Storyboard.TargetProperty="Opacity"
                     To="1" Duration="0:0:1" />
  </Storyboard>
</VisualState>
```

Notice that the Duration on the BeforeLoaded state is 0, which effectively just sets the Opacity property to 0. Using a whole Storyboard and DoubleAnimation just to set a property might seem like overkill, but it also demonstrates the flexibility of animations. The overhead is actually not very much.

The approach I personally prefer—primarily because it's the simplest—is to leave the Opacity property of the Grid at its default value of 1 and provide only an animation for the AfterLoaded state with a From value specified, rather than a To value:

```
<VisualState x:Name="AfterLoaded">
  <Storyboard>
    <DoubleAnimation Storyboard.TargetName="rootGrid"
                     Storyboard.TargetProperty="Opacity"
                     From="0" Duration="0:0:1" />
  </Storyboard>
</VisualState>
```

Now the animation goes from the value of 0 to its base value, which is 1. You can use this identical technique with the BeforeLoaded state. But watch out: The BeforeLoaded state occurs after the ListBoxItem is created and initialized, but before it's added to the visual tree, at which point the AfterLoaded state occurs. That's just a tiny gap of time. You'll get into trouble if you define an animation for BeforeLoaded but also define an empty VisualState tag for AfterLoaded:

```
<VisualState x:Name="BeforeLoaded">
  <Storyboard>
    <DoubleAnimation Storyboard.TargetName="rootGrid"
                     Storyboard.TargetProperty="Opacity"
                     From="0" Duration="0:0:1" />
  </Storyboard>
</VisualState>

<VisualState x:Name="AfterLoaded" />
```

As soon as the item is loaded, the storyboard for BeforeLoaded is terminated and you'll get no fade-in effect. However, you can make that markup work if you also add the following:

```
<VisualStateGroup.Transitions>
  <VisualTransition From="BeforeLoaded"
                    To="AfterLoaded"
                    GeneratedDuration="0:0:1" />
</VisualStateGroup.Transitions>
```

This defines a one-second transition period between the BeforeLoaded and the AfterLoaded states. That transition period gives the BeforeLoaded animation time to complete before the AfterLoaded state shuts it off.

The fade-out process isn't quite as straightforward. When the item is about to be removed from the ListBox, the BeforeUnloaded state is set, but then the item is immediately removed so any animation that began won't be visible! I've found two approaches that work. The first defines an animation for the BeforeUnloaded state together with a transition for that state:

```
<VisualState x:Name="BeforeUnloaded">
  <Storyboard>
    <DoubleAnimation Storyboard.TargetName="rootGrid"
                     Storyboard.TargetProperty="Opacity"
                     To="0" Duration="0:0:1" />
  </Storyboard>
</VisualState>

<VisualStateGroup.Transitions>
  <VisualTransition From="AfterLoaded"
                    To="BeforeUnloaded"
                    GeneratedDuration="0:0:1" />
</VisualStateGroup.Transitions>
```

Because the fluid UI feature is implemented as visual states on ListBoxItem, it isn't available in the ItemsControl.

The second approach defines an empty tag for the BeforeUnloaded state and an animation for the VisualTransition:

```
<VisualState x:Name="BeforeUnloaded" />

<VisualStateGroup.Transitions>
  <VisualTransition From="AfterLoaded"
                    To="BeforeUnloaded"
                    GeneratedDuration="0:0:1">
    <Storyboard>
      <DoubleAnimation Storyboard.TargetName="rootGrid"
                       Storyboard.TargetProperty="Opacity"
                       To="0" Duration="0:0:1" />
    </Storyboard>
  </VisualTransition>
</VisualStateGroup.Transitions>
```

Figure 2 shows the completed markup for the AfterLoaded and BeforeUnloaded states as they appear in the ListBoxItem template in the MainPage.xaml file of the FluidListBox project.

One more warning: By default, the ListBox stores its items in a VirtualizingStackPanel. This means the actual items and their containers aren't generated until they're required to be visually displayed. If you define an animation for the AfterLoaded state, and then fill the ListBox up with items, the items will fade in as they're scrolled into view. This is probably undesirable. The easy solution is to replace the VirtualizingStackPanel with a regular StackPanel. The required markup on the ListBox is trivial:

```
<ListBox.ItemsPanel>
  <ItemsPanelTemplate>
    <StackPanel />
  </ItemsPanelTemplate>
</ListBox.ItemsPanel>
```

Extending to ItemsControl

Because the fluid UI feature is implemented as visual states on ListBoxItem, it isn't available in the ItemsControl. As you know, ItemsControl simply displays a collection of items and lets the user navigate through them. There's no concept of selection or input focus among the items. For that reason, ItemsControl doesn't require a special class like ListBoxItem to host the items. It just uses a ContentPresenter. Because ContentPresenter derives from FrameworkElement rather than Control, it doesn't have a template in which to define the behavior of visual states.

What you can do, however, is derive a class from ItemsControl that uses ListBoxItem to host its items. This is actually much easier than you might assume. **Figure 3** shows the entire code for FluidableItemsControl.

> It's time that we application developers started considering implementing our own visual states for custom behavior.

The crucial method is GetContainerForItemOverride. This method returns the object used to wrap each item. ItemsControl returns ContentPresenter, but ListBox returns ListBoxItem, and that's what FluidableItemsControl returns as well. This ListBoxItem must have a style applied, and for that reason FluidableItemsControl also defines the same ItemContainerStyle property as ListBox.

The other method that should be implemented is IsItemItsOwnContainerOverride. If the item in the ItemsControl is already the same type as its container (in this case, a ListBoxItem), then there's no reason to put it in another container. Now you can set a ListBoxItem style definition to the ItemContainerStyle property of FluidableItemsControl. The template within the style definition can be drastically simplified. It doesn't need logic for mouse-over, selection or input focus, so all those visual states can be eliminated, as well as the three Rectangle objects.

Figure 2 **An Excerpt from the ListBoxItem Template in FluidListBox**

```
<ControlTemplate TargetType="ListBoxItem">
  <Grid Name="rootGrid" Background="{TemplateBinding Background}">
    <VisualStateManager.VisualStateGroups>

      <!-- Additions to standard template -->
      <VisualStateGroup x:Name="LayoutStates">

        <VisualState x:Name="AfterLoaded">
          <Storyboard>
            <DoubleAnimation Storyboard.TargetName="rootGrid"
                             Storyboard.TargetProperty="Opacity"
                             From="0" Duration="0:0:1" />
          </Storyboard>
        </VisualState>

        <VisualState x:Name="BeforeUnloaded" />

          <VisualStateGroup.Transitions>
            <VisualTransition From="AfterLoaded"
                              To="BeforeUnloaded"
                              GeneratedDuration="0:0:1">
              <Storyboard>
                 <DoubleAnimation Storyboard.TargetName="rootGrid"
                                  Storyboard.TargetProperty="Opacity"
                                  To="0" Duration="0:0:1" />
              </Storyboard>
            </VisualTransition>
          </VisualStateGroup.Transitions>
        </VisualStateGroup>
        <!-- End of additions to standard template -->
            ...
  </Grid>
</ControlTemplate>
```

Figure 3 **The FluidableItemsControl Class**

```
using System.Windows;
using System.Windows.Controls;

namespace FluidItemsControl
{
  public class FluidableItemsControl : ItemsControl
  {
    public static readonly DependencyProperty ItemContainerStyleProperty =
      DependencyProperty.Register("ItemContainerStyle",
      typeof(Style),
      typeof(FluidableItemsControl),
      new PropertyMetadata(null));

    public Style ItemContainerStyle
    {
      set { SetValue(ItemContainerStyleProperty, value); }
      get { return (Style)GetValue(ItemContainerStyleProperty); }
    }

    protected override DependencyObject GetContainerForItemOverride()
    {
      ListBoxItem container = new ListBoxItem();

      if (ItemContainerStyle != null)
        container.Style = ItemContainerStyle;

      return container;
    }

    protected override bool IsItemItsOwnContainerOverride(object item)
    {
      return item is ListBoxItem;
    }
  }
}
```

The FluidItemsControl program shows the result. It's pretty much the same as FluidListBox but with all the ListBox selection logic absent. The default panel for ItemsControl is a StackPanel, so that's another simplification. To compensate for these simplifications, I've enhanced the animations for loading and unloading items. Now there's an animation on the PlaneProjection transform that makes it appear as if the items are swiveling into and out of view.

Limitations and Suggestions

Even with the facility to define animations on items in an ItemsControl or ListBox, there still exists a crucial limitation: If the control incorporates a ScrollViewer, you can't define transforms that take the item out of the box. The ScrollViewer imposes a severe clipping region that simply can't be transgressed (as far as I've been able to determine). This means that techniques such as those I demonstrated in last month's column are still valid and important in Silverlight 4.

But the use of the VSM to implement this fluid UI feature in Silverlight 4 is a good indication that the VSM is likely to play an increasingly important role in the future to link code and XAML. It's time that we application developers started considering implementing our own visual states for custom behavior. ■

CHARLES PETZOLD *is a longtime contributing editor to* MSDN Magazine*. He's currently writing "Programming Windows Phone 7 Series," which will be published as a free downloadable e-book in the fall of 2010. A preview edition is currently available through his Web site, charlespetzold.com.*

DAVID S. PLATT

Rejectionists Rejected

The response to my last column, calling on Microsoft to publish UI standards for Windows Presentation Foundation (WPF) and Silverlight, was quite gratifying, both in praise and scorn. Many readers loved it; others detested it and said so quite loudly. Here are the most strident objections, with my refutations.

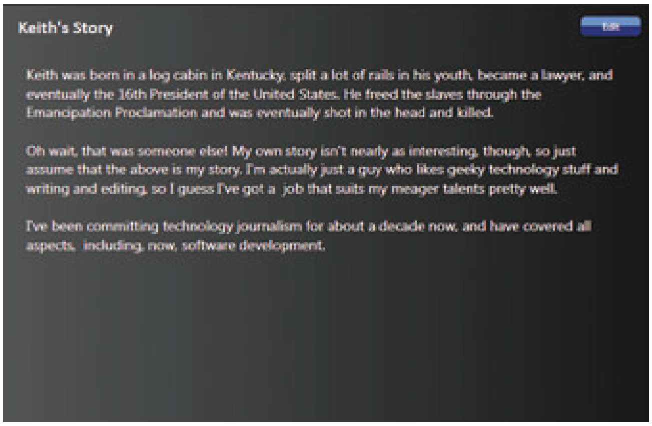
Some readers hated the idea of any sort of standard. "Plattski, you Luddite, shut up," they wrote, "we don't need no stinkin' standards. That's so 20th century. We'll do things that are cool and users will love them because we love cool and users are just like us." No they're not, and no they won't. As I've said before in this space, users don't care about your software in and of itself. Never have, never will; not even your mother. They don't want cool, they want finished.

Almost all the readers who said this are under age 35. I picture them rolling their eyes at me, as my daughter, now 10, practices daily for her approaching teen years. They've grown up with UI commonality as they've grown up with the measles vaccine: Never experiencing—and rarely even thinking about—the absence of either. But I've experienced the world both ways, and let me tell you: Learning the UI peccadilloes of different applications at best consumes time and effort that could be used more productively, and at worst drives a user barking mad when the Save command of one program is the Delete command of another. And even among otherwise healthy patients in developed countries, measles kills one or two out of 1,000 patients and permanently damages more. We're far better off today having both UI commonality and the measles vaccine, and giving up either one of them is a bad idea.

A second cohort wrote: "We don't want Microsoft to prescribe standards. We want standards to evolve naturally out of the use of WPF in our applications." My response: WPF has been out for four years now. Pioneer companies have spent eons of programmer time and mountains of money on WPF, some of which made users happier and some of which made them less happy.

The Family.Show sample genealogy application from Vertigo offers spectacular examples of both, including excellent subconscious right-brain communication, down to and including the speedy infliction of physical pain. (See my article, "Using WPF for Good and Not Evil" at tinyurl.com/27anuy7, for details.) We darn well better have learned something from examples like this. Microsoft is the only entity that can gather the community experiences, combine it with their own extensive data and promulgate it industry-wide.

A third cohort wrote: "Standards cramp innovation and are a huge barrier to progress, the classic example being the QWERTY keyboard." Poppycock. Standards raise the bar for what's a useful innovation and what isn't. If an alternate keyboard layout were that much more efficient, we'd use it. If you can make users happier by violating a standard, more power to you. An excellent example is Microsoft OneNote, which automatically saves documents without needing user action. If users like it, it'll become the new standard. Following most of the standards allows the rest of your application to work while you present your new innovation to users for their approval or disapproval. Just know what you're doing and why you're doing it.



Keith's Story

Keith was born in a log cabin in Kentucky, split a lot of rails in his youth, became a lawyer, and eventually the 16th President of the United States. He freed the slaves through the Emancipation Proclamation and was eventually shot in the head and killed.

Oh wait, that was someone else! My own story isn't nearly as interesting, though, so just assume that the above is my story. I'm actually just a guy who likes geeky technology stuff and writing and editing, so I guess I've got a job that suits my meager talents pretty well.

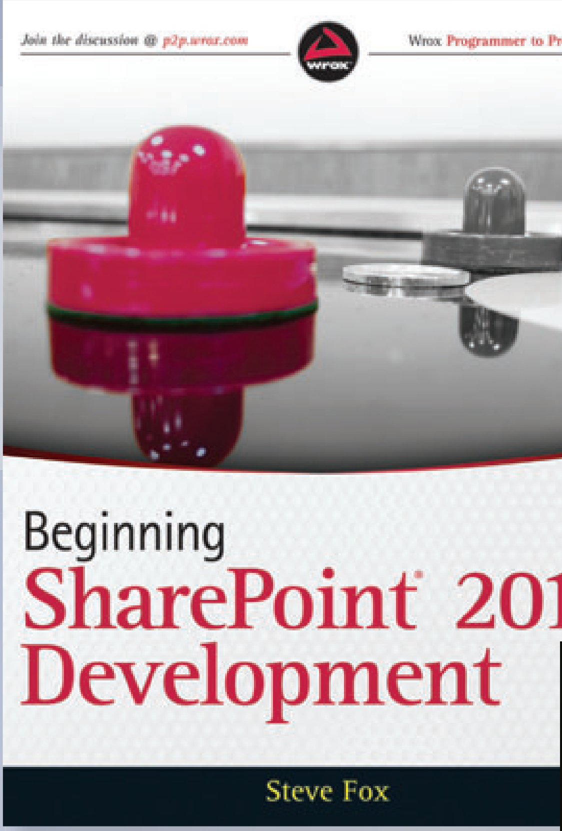
I've been committing technology journalism for about a decade now, and have covered all aspects, including, now, software development.

From the Family.Show application. This is what can happen without standards.

Social manners, such as shaking hands or bowing, are behavioral conventions that help people live and work together harmoniously. As technology advances, we invent new behavioral conventions to cover the innovations; for example, turning off cell phones in a theater. Similarly, UI standards are conventions that help people and their computer programs live and work together harmoniously. As UI technology advances, we need new conventions as to how and when to employ its new features to make users more happy—not less. And we need them right now, as WPF and Silverlight development transitions from pioneer to mainstream. ■

David S. Platt teaches Programming .NET at Harvard University Extension School and at companies all over the world. He is the author of 11 programming books, including "Why Software Sucks" (Addison-Wesley Professional, 2006) and "Introducing Microsoft .NET" (Microsoft Press, 2002). Microsoft named him a Software Legend in 2002. He wonders whether he should tape down two of his daughter's fingers so she learns how to count in octal. You can contact him at rollthunder.com.

SharePoint 2010 from the experts.



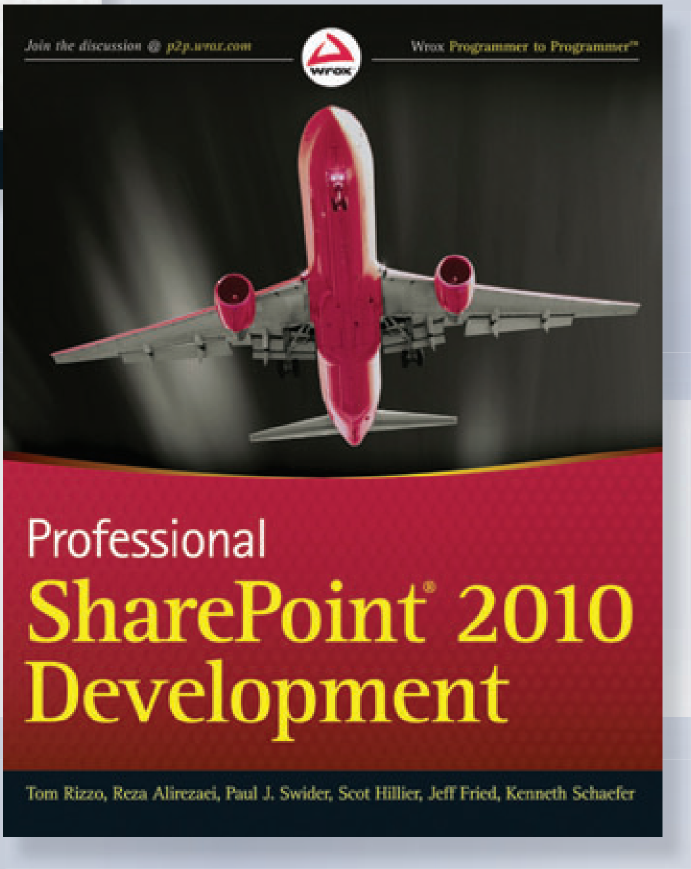
Join the discussion @ p2p.wrox.com

Wrox Programmer to Programmer™

Beginning SharePoint® 2010 Development

Steve Fox

From beginner to professional, we have SharePoint 2010 covered.



Join the discussion @ p2p.wrox.com

Wrox Programmer to Programmer™

Professional SharePoint® 2010 Development

Tom Rizzo, Reza Alirezaei, Paul J. Swider, Scot Hillier, Jeff Fried, Kenneth Schaefer

Join us on facebook where you can connect with Wrox and learn more about the latest books on SharePoint 2010!

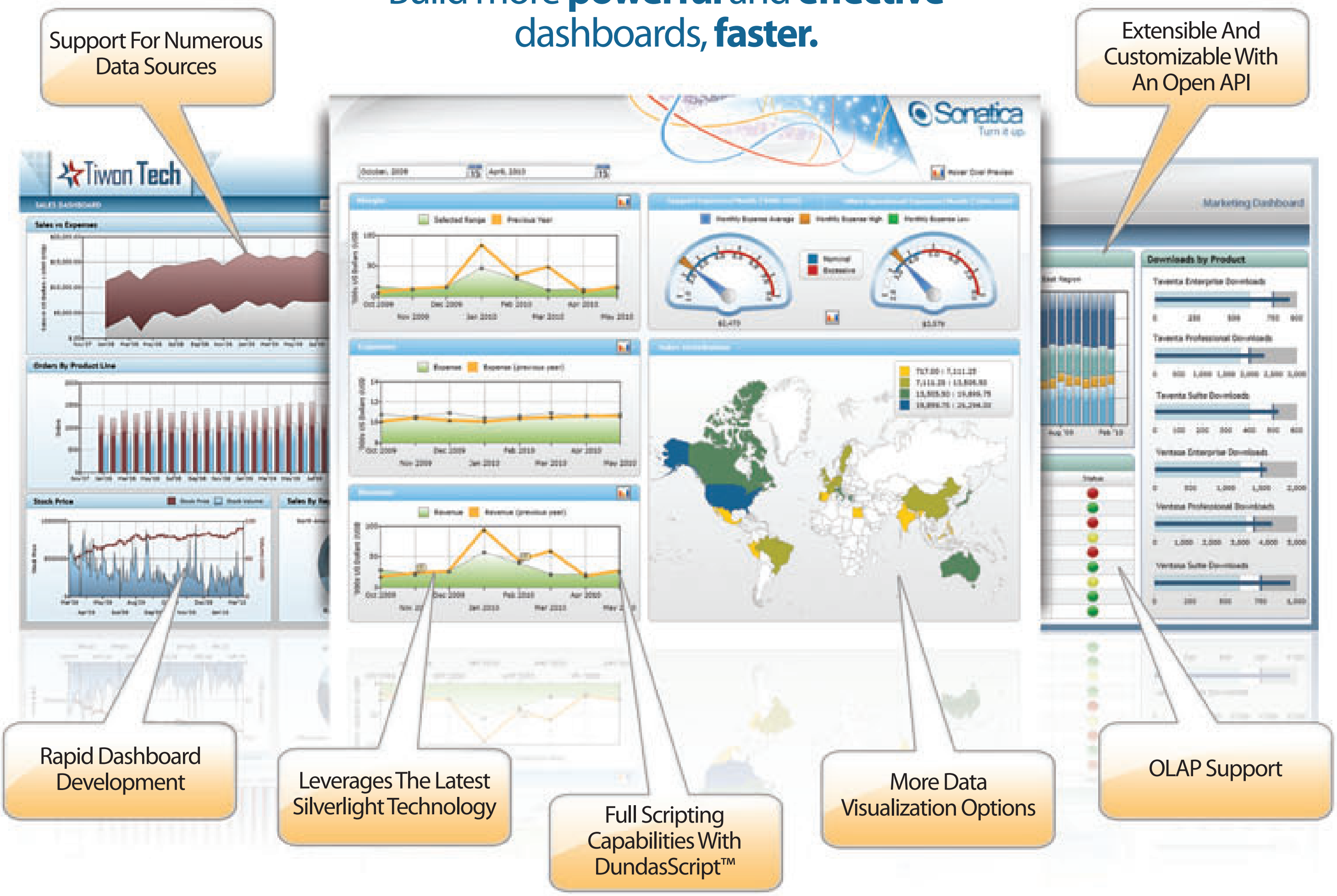
facebook.com/wroxpress

wrox™

Dundas Dashboard

The Dashboard Platform for Developers like you

Build more **powerful** and **effective** dashboards, **faster.**



Support For Numerous Data Sources

Extensible And Customizable With An Open API

Rapid Dashboard Development

Leverages The Latest Silverlight Technology

Full Scripting Capabilities With DundasScript™

More Data Visualization Options

OLAP Support

Dundas Dashboard is a ground-breaking, extensible solution that uses a revolutionary approach to dashboard creation, providing you with a unified view of key metrics and a new level of strategic insight and decision-making.

Powered by Microsoft Silverlight

Dundas
Data Visualization

www.dundas.com/dashboard
(416) 467-5100 • (800) 463-1492