The Microsoft Journal for Developers

msdn® magazine

IIS SMOOTH STREAMING

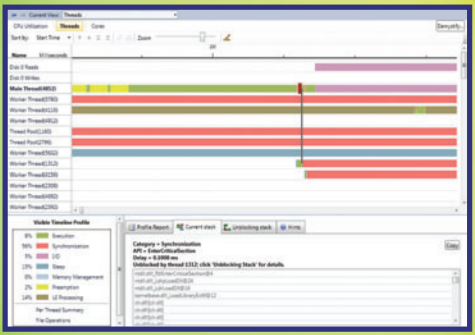
Enhancing Silverlight Video Experiences with Contextual Data

Jit Ghosh page 32

FINGER STYLE

Exploring Multi-Touch Support in Silverlight

Charles Petzold page 46



THREAD DIAGNOSTICS

Performance Tuning with the Concurrency Visualizer in Visual Studio 2010

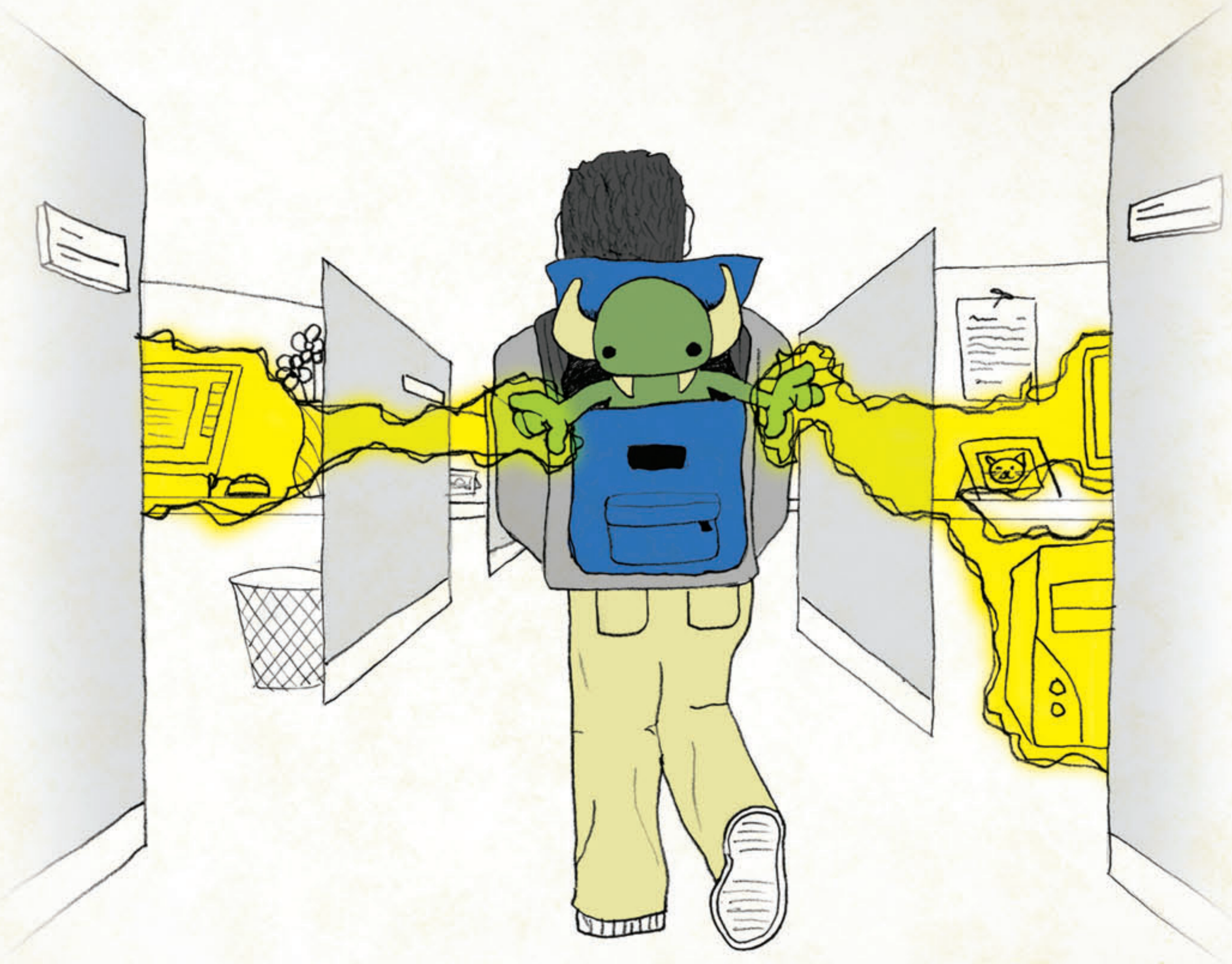
Hazim Shafi page 56

msdn

Microsoft®

MARCH 2010 VOL 25 NO 3

COLUMNS



ADD SOME EXTREME TO YOUR TEAM

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

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

Infragistics®

KILLER APPS. NO EXCUSES.

dtSearch®

Instantly Search Terabytes of Text

- 25+ full-text and fielded data search options
- Built-in file parsers and converters highlight hits in popular file types
- Spider supports static and dynamic web data; highlights hits with links, formatting and images intact
- API supports C++, .NET, Java, SQL, etc. .NET Spider API. Includes 64-bit (Win/Linux)
- Fully-functional evaluations available

Content extraction only licenses also available

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

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

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

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

1-800-IT-FINDS • www.dtSearch.com

The Smart Choice for Text Retrieval® since 1991

Microsoft® msdn® magazine

MARCH 2010 VOLUME 25 NUMBER 3

LUCINDA ROWLEY Director
DIEGO DAGUM Editorial Director
KERI GRASSL Site Manager

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

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

CONTRIBUTING EDITORS 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
Christopher M. Coates Vice President, Finance & Administration
Abraham M. Langer Vice President, Digital Media, Audience Marketing
Erik A. Lindgren Vice President, Information Technology & Web Operations

Jeffrey S. Klein Chairman of the Board

MSDN Magazine (ISSN 1528-4859) is published monthly by 1105 Media, Inc., 9201 Oakdale Avenue, Ste. 101, Chatsworth, CA 91311. Periodicals postage paid at Chatsworth, CA 91311-9998, and at additional mailing offices. Annual subscription rates payable in U.S. funds: U.S. $35; Canada $45; International $60. Single copies/back issues: U.S. $10, all others $12. Send orders with payment to: MSDN Magazine, P.O. Box 3167, Carol Stream, IL 60132, e-mail MSDNmag@1105service.com or call 847-763-9560. **POSTMASTER:** Send address changes to *MSDN Magazine*, P.O. Box 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.

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

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

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

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

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

BPA WORLDWIDE℠

Printed in the USA

Your best source for software development tools!

programmer's paradise®

LEADTOOLS Recognition SDK v16.5

by LEAD Technologies

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

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

Paradise # L05 26301A01

$3,214.99

programmers.com/LEAD

Multi-Edit 2008

by Multi Edit Software

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

1-49 Users
Paradise # A30 01201A01

$179.99

programmers.com/multiedit

VMware vSphere

Put time back into your day.

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

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

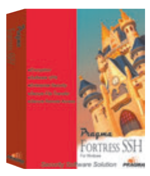
Make your time (and money) count for more with virtualization from VMware.

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

$9,234.99

programmers.com/vSphere

Pragma Fortress SSH—SSH Server & Client for Windows



by Pragma Systems

Contains SSH, SFTP, SCP servers and clients for Windows.

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

Certified for Windows 7/2008R2

Paradise # P35 04201A01

$550.99

programmers.com/pragma

gDoc Fusion

by Global Graphics

gDoc Fusion is the easiest way to assemble information from a range of sources and formats and quickly create enterprise-quality PDF, XPS or Microsoft Word documents for sharing, printing or collaboration.

- One-click PDF/XPS creation
- Merge and combine documents quickly
- Flick through large documents like a book
- Context sensitive text editing

1-4 Users Includes Maint & Support*
Paradise # F39 ZSP0008

$181.99

* Price per user.

programmers.com/globalgraphics

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

FarPoint Spread for Windows Forms

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

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

NEW VERSION 5!

Paradise # F02 01101A01

$936.99

programmers.com/grapecity

Enterprise Architect 7.5

Visualize, Document and Control Your Software Project

by Sparx Systems

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

Corporate Edition
1-4 Users
Paradise # SP6 03101A02

$182.99

programmers.com/sparxsystems

TX Text Control 15.1

Word Processing Components

NEW RELEASE!

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

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

Professional Edition
Paradise # T79 02101A02

$848.99

Download a demo today.

programmers.com/theimagingsource

AdminStudio & Application Virtualization Pack

by Flexera Software

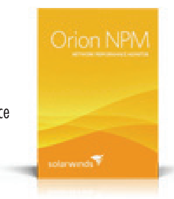
One Application Software Deployment Tool for Reliable MSI Packaging, Application Virtualization, and Windows 7 Migration. Top choice of Microsoft®, Novell®, LANDesk® and other software management solutions. Cut MSI packaging time by up to 70%, Deploy software to desktops with 99% success or better. AdminStudio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

Professional Upgrade from any Active AS Pro + Silver Mtn
Paradise # I21 09401S05

$4,228.99

programmers.com/flexera

Orion Network Performance Monitor



by Solarwinds

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

Paradise # S4A 08201E02

$4,606.99

programmers.com/solarwinds

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

$3,951.99

programmers.com/ca

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

Prices subject to change. Not responsible for typographical errors.

Coming out the Other Side

Typing in "ASP.NET" at the popular employment site Monster.com brings up 1,848 hits. Typing in "C#" brings up 3,247 matches; "SQL programmer" returns 874 openings.

Do figures like that mean anything in this job market, with many companies still handing out pink slips left and right, and continuing to outsource development jobs? In short—is the recession really over, as the experts claim, or are the hard times going to loiter awhile, like a 19-year-old on a Baltimore street corner?

The software development community, thought to be mostly immune to the unemployment line, has felt the stinging lash of the bleak economic picture as well. If you haven't lost your job, chances are good you have a colleague or friend who has.

Because anecdotal evidence is of little value on matters like these, I went to an expert to get his take. Tom Silver is the senior vice president, North America, at tech job search Web site Dice.com. On the whole, he's optimistic about the outlook for software developers going forward.

Silver gave me a "state of the state" rundown of the current situation. "The market is definitely improving for software developers," he says. "There are currently nearly 8,500 open jobs being advertised on Dice.com. That's up 7 percent year-over-year and has increased more than 20 percent in the last six months. We see opportunities in a wide variety of industries, and 45 states and the District of Columbia currently have openings."

That optimism applies to salaries as well, Silver says, which are trending up—only a bit, but in this environment, that's good to hear. "Software Engineers enjoyed a slight increase (1.5 percent) in average salaries to $91,342. That compares to a 1 percent increase for tech in general between 2009 and 2010. Software Engineers continue to be paid well—16 percent above the overall tech average salary of approximately $78,000," according to Silver.

But what if you're on the outside of those rosy statistics, looking in? What do you do to make yourself more employable? For one thing, Silver says, don't focus only on your programming chops; sharpen your communication and people skills, the so-called soft skills.

"Soft skills are important, but don't replace the appropriate experience and skill-set," Silver says. "However, most companies want excellent communication and interpersonal skills. Software developers have to have the ability to present ideas in a business-friendly and user-friendly language. They have to be highly self-motivated and -directed with excellent analytical and problem-solving capabilities. If you can combine excellent technical know-how with soft skills, it's a killer combination."

In general, Silver continues, it's better to have a big toolbox with a lot of gadgets than a small one with many specialized tools. "Broadening your skill-set is the most important. Also, our research shows that having more skills typically begets a higher income."

So, what does the immediate future hold for developers? Does Silver expect 2010 to surpass last year? "Hands down, better," he gushes. He sounds a little Pollyanna-ish, but does have numbers to back up his statements. "We think 2010 is going to be about retention," Silver says. "With confidence returning and the number of available jobs improving, software developers should be willing to go fight for what they want, whether their preference is more compensation, training or career growth."

And although it may be small comfort to the developers still knocking on doors, Silver continues to believe that this industry is the right one to be in. "Companies forget that the unemployment rate for technology professionals is 4.5 percent, as compared to 10 percent overall—and well improved from the peak [6.2 percent] in this cycle. Technology professionals told us very clearly that they don't think their companies did any of the 'soft' items to keep them motivated during the downturn. It's simply a matter of time before technology professionals start looking around."

Does this outlook cheer you? Or are you out of work and as discouraged as ever? Tell me your story at mmeditor@microsoft.com.

Keith Ward

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

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

FREE! 1-User License

Axosoft customers have saved

$1,985,575,8

by shipping software OnTime!

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

See how at axosoft.com

Project Management • Bug Tracking • Scrum

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

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

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

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

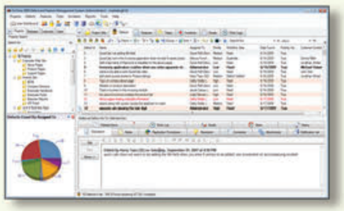
Workflow
Define a hyper-fast, lightweight, agile workflow or a robust process-enforcement system.

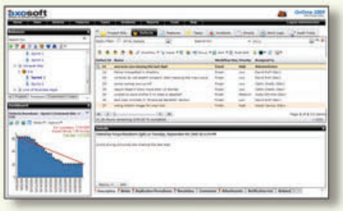
Notifications
Ensure team members are in synch with automatic email notifications and alerts.

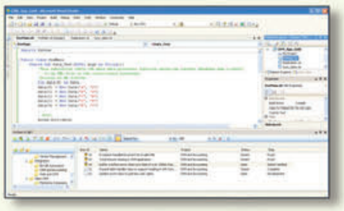
Customize
Unlimited custom fields, customize all field names, and decide when fields are visible, editable or required.

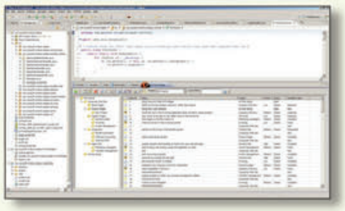
Security
Secure every aspect of the system by project, user roles, and even fields.

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


Windows


Web


Visual Studio


Eclipse

axosoft
Software for Software Development™

800.653.0024
www.axosoft.com

DINO ESPOSITO

ASP.NET Ajax Library and WCF Data Services

It's been a few years since the Web industry praised the advent of AJAX as the beginning of a new era of programming prosperity, and now a powerful set of programming tools is finally available to Web developers: the ASP.NET Ajax Library and WCF Data Services. Developers can stop relying on the browser as a distinct runtime environment and perform from the Web a number of tricks that were previously only possible through smart clients.

The ability to place calls to a remote HTTP endpoint is a common feature that many applications take advantage of today. Such applications use Windows Communication Foundation (WCF) services to download JavaScript Object Notation (JSON) data streams, and parse that content into JavaScript objects that are then rendered into the current HTML Document Object Model (DOM). However, the WCF service on the server side—and the JavaScript code on the client side—work on different data types, forcing you to create two distinct object models.

You typically need a domain model on the server side, which is how your middle tier handles and represents entities. Entity Framework and LINQ to SQL are two excellent tools for designing your server-side object model, either from scratch or by inferring it from an existing database. At some point, though, you need to transfer this data to the client as the response of a WCF service call.

One of the most popular mantras of service-oriented architecture (SOA) maintains that you should always transfer data contracts, not classes, as part of decoupling the presentation and business tiers. So you need another completely distinct object model: the view model for the presentation.

A common problem is detecting and communicating to the server any changes to the data that have occurred on the client. Communicating only changes ensures that the least amount of data possible is transferred across the wire, and optimized data access operations can be performed against the database.

An end-to-end solution for data access and manipulation is therefore required. WCF Data Services (formerly ADO.NET Data Services) and the ASP.NET Ajax Library combine to deliver a comprehensive framework that allows you to download data, work on it and return updates to the server. In this article, I'll look at the ASP.NET Ajax JavaScript components for effective client-side data access.

WCF Data Services in a Nutshell

The key idea behind the cutting-edge, end-to-end data access solution in WCF Data Services is that you build a server-side data source and expose it through a special flavor of a WCF service: a WCF Data Service. (You'll find an excellent introduction to WCF Data Services in the August 2008 edition of *MSDN Magazine*, available at msdn.microsoft.com/magazine/cc748663.)

You typically start by creating the business domain model using Entity Framework, then create a WCF service around it. I'll be using the familiar Northwind database and working on a small subset of its tables: Customers and Orders.

First, you create a new class library project and add a new item of type ADO.NET Entity Data Model. Next, you compile the project and reference the assembly from within a new ASP.NET application. In the sample code, I use an ASP.NET MVC project. You merge any connection string settings with the web.config of the host Web application and add a new "ADO.NET Data Service" item to the Web project. (The old name is still being used in Visual Studio 2008 and in the beta 2 of Visual Studio 2010.) You now have a class library with the data source and a WCF Data Service that, hosted in the ASP.NET application, exposes the content to the clients.

In the simplest case, this is all the code you need to have in the WCF Data Service:

```
public class NorthwindService : DataService<NorthwindEntities>
{
   public static void InitializeService(IDataServiceConfiguration
config)
   {
      config.SetEntitySetAccessRule("Customers", EntitySetRights.All);
    }
}
```

As you progress on your project, you'll probably need to add more access rules on entity sets and also (why not?) support for additional service operations. A WCF Data Service is a plain WCF service that is consumed in a REST way. Nothing, therefore, prevents you from adding one or more new service operations, each representing a coarse-grained operation on the data, such as a complex query or a sophisticated update. Aside from that, as shown in the previous code, the service will provide client access to the Customers entity set, with no restrictions on operations. This means that you won't be able to query for customers and orders, even though the Orders entity set does exist in the embedded entity model. A new access rule is required to enable client access to orders.

This column is based on a prerelease version of ASP.NET Ajax Library. All information herein is subject to change.

Code download available at code.msdn.microsoft.com/mag201003CuttingEdge.

Devexpress

The New **DXGRID** for WPF

by DevExpress

FEATURE COMPLETE PRESENTATION COMPONENTS • EASY-TO-USE REPORTING CONTROLS
IDE PRODUCTIVITY TOOLS • BUSINESS APP FRAMEWORKS

LEARN MORE and DOWNLOAD YOUR FREE EVALUATION COPY TODAY
VISIT **DEVEXPRESS.COM/WPFGRID**

READERS' CHOICE 2009 asp.netPRO WINNER · Visual Studio READERS CHOICE 2009 · WindowsITPro SQLServer Microsoft tech·ed North America 2009 ATTENDEES' PICK · WindowsITPro SQLServer Best of Microsoft tech·ed North America 2009

Figure 1 **Using Data Context as a Data Provider**

```
<table>
    <tr class="tableHeader">
        <td>ID</td>
        <td>Name</td>
        <td>Contact</td>
    </tr>
    <tbody sys:attach="dataview"
           class="sys-template"
           dataview:dataprovider="{{ dataContext }}"
           dataview:fetchoperation="Customers"
           dataview:autofetch="true">
         <tr>
             <td>{{ CustomerID }}</td>
             <td>{{ CompanyName }}</td>
             <td>{{ ContactName }}</td>
         </tr>
     </tbody>
</table>
```

Figure 2 **Using the AdoNetQueryBuilder Object**

```
<script type="text/javascript">
    var dataContext;
    var queryObject;

    Sys.require([Sys.components.dataView,
                Sys.components.openDataContext]);

    Sys.onReady(function() {
        dataContext = Sys.create.openDataContext(
           {
               serviceUri: "/NorthwindService.svc",
               mergeOption: Sys.Data.MergeOption.appendOnly
           });
        queryObject = new Sys.Data.OpenDataQueryBuilder("Customers");
        queryObject.set_orderby("ContactName");
        queryObject.set_filter("City eq " + "'London'");
        queryObject.set_expand("Orders");
    });
</script>
```

Until you add new REST methods to the WCF data service, the only operations allowed are generic create, read, update and delete (CRUD) operations expressed using an ad-hoc URI format. (See msdn.microsoft.com/data/cc668792 for more details on the syntax.) The URI format allows applications to query for entities, traverse the relationships between entities, and apply any changes. Each CRUD operation is mapped to a different HTTP verb: GET for queries, POST for insertions, PUT for updates and DELETE for deletions.

A client that gets a reference to a WCF Data Service receives a proxy class created by the datasvcutil.exe utility or transparently created by the Add Service Reference wizard in Visual Studio.

Invoking a WCF Data Service from a smart client platform couldn't be easier, whether Silverlight, Windows or Windows Presentation Foundation (WPF). The same can be said for server-side binding to ASP.NET. What about Web clients based on JavaScript and AJAX?

Consuming Data Services via ASP.NET Ajax Library

In the ASP.NET Ajax Library, there are two JavaScript components related to WCF Data Services: OpenDataServiceProxy and OpenDataContext.

OpenDataContext is essentially designed for managing CRUD operations from within the Web client. It can be thought of as the JavaScript counterpart of the DataServiceContext class. Defined in the System.Data.Services.Client namespace, DataServiceContext represents the runtime context of the specified WCF Data Service. OpenDataContext tracks changes to the entities being used and can intelligently generate commands against the back-end service.

The OpenDataServiceProxy class is intended as a lightweight additional proxy for a WCF Data Service. It essentially manages read-only scenarios but can be used to invoke additional service operations exposed in the service. You initialize the OpenDataServiceProxy class as follows:

```
var proxy = new Sys.Data.OpenDataServiceProxy(url);
```

At this point, the class is up and running. You typically need to spend more time configuring an OpenDataContext object. As far as the connection to the service is concerned, though, it happens in a similar way:

```
var dataContext = new Sys.Data.OpenDataContext();
dataContext.set_serviceUri(url);
```

Both classes can be used as data providers for a DataView. Which one you use depends on what you want to do. If any CRUD activity has to happen via the service, you're probably better off going with a proxy. If you have logic on the client and intend to perform a bunch of CRUD operations before you apply changes, then data context is preferable. Let's focus on OpenDataContext.

Using the OpenDataContext Class

Here's how to create and initialize an instance of the OpenDataContext class:

```
<script type="text/javascript">
  var dataContext;

  Sys.require([Sys.components.dataView, Sys.components.
openDataContext]);

  Sys.onReady(function() {
    dataContext = Sys.create.openDataContext(
      {
         serviceUri: "/NorthwindService.svc",
         mergeOption: Sys.Data.MergeOption.appendOnly
      });

    });
</script>
```

Note the use of the Sys.require function to dynamically link only script files that serve the purpose of components being used. If you opt for the Sys.require approach, the only script file you need to link in the traditional way is start.js:

```
 (see code <script src="../../Scripts/MicrosoftAjax/Start.js"
  type="text/javascript">
</script>
```

All files you use, though, must be available on the server or referenced through the Microsoft Content Delivery Network (CDN).

Looking ahead to **Figure 2**, you can see that in the document's ready event, you create a new instance of the OpenDataContext class. Note again the use of the newest abbreviated syntax to define code for common events and instantiate common objects. The factory of the OpenDataContext class receives the URL of the service and some additional settings. At this point, you're ready to use the data context as the data provider to some DataView UI components in the page, as shown in **Figure 1**.

An instance of the DataView component is created and used to populate the template it's attached to. The DataView provides the glue code necessary to download data via the WCF Data Service and bind it to the HTML template. Where is the decision made about the data to be downloaded? In other words, how do you specify the query string for the data you want back?

DevExpress

The **ASPx GRIDVIEW** for ASP.NET

by DevExpress

FEATURE COMPLETE PRESENTATION COMPONENTS • EASY-TO-USE REPORTING CONTROLS
IDE PRODUCTIVITY TOOLS • BUSINESS APP FRAMEWORKS

LEARN MORE and DOWNLOAD YOUR FREE EVALUATION COPY TODAY
VISIT **DEVEXPRESS.COM/ASPXGRIDVIEW**

READERS' CHOICE 2009 asp.netPRO WINNER · Visual Studio READERS 2009 CHOICE · WindowsITPro SQLServer Microsoft tech·ed North America 2009 ATTENDEES' PICK · WindowsITPro SQLServer Best of Microsoft tech·ed North America 2009

The fetchoperation property of the DataView component indicates the name of the service operation to be invoked. If the data provider is a plain proxy for the service, then the fetchoperation property takes the name of a public method on the service. If you instead go through the OpenDataContext class, the value for fetchoperation is expected to be a string that the runtime of the WCF Data Service can understand. It can be an expression like any of these:

```
Customers
Customers('ALFKI')
Customers('ALFKI')?$expand=Orders
Customers('ALFKI')?$expand=Orders&$orderBy=City
```

If you simply specify the name of a valid entity set, you'll get the entire list of entities. Other keywords, such as $expand, $orderBy and $filter, allow you to include related entity sets (sort of an inner join), order on a property and filter returned entities based on a Boolean condition.

You can compose the query manually as a string, being respectful of the underlying URI format. Or you can use the built-in OpenDataQueryBuilder JavaScript object, as shown in **Figure 2**.

The query builder can be used to build complete URLs, or just the query part of the URL. In this case, the query builder gets the name of the entity set to query. It also offers a bunch of properties for setting any required expansion, filters and orders. The criteria set via the query builder object must then be serialized to an effective query string when that fetchoperation is set, as shown here:

```
<tbody sys:attach="dataview"
       class="sys-template"
       dataview:dataprovider="{{ dataContext }}"
       dataview:fetchoperation="{{ queryObject.toString() }}"
       dataview:autofetch="true">
```

You use the toString method to extract the query string from the query builder. In the sample code, the resulting query string is

```
Customers?$expand=Orders&$filter="City eq 'London'"&$orderby=ContactName
```

The service returns a collection of composite objects that embed customer demographics plus some order information. **Figure 3** shows the output.

The numbers in the last column indicate the number of orders that have been placed by the customer. Because of the $expand attribute in the query, the JSON data stream contains an array of orders. The HTML template refers to the length of the array and populates the column like this:

```
 <td>{{ Orders.length }}</td>
```

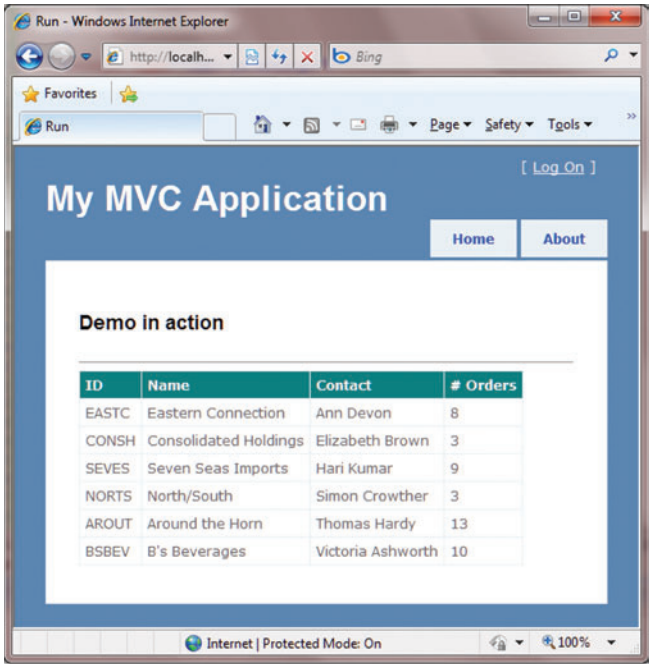
Note that in order to successfully retrieve order information, you should first go back to the source code of the WCF Data Service and enable access to the Orders entity set:

```
public static void InitializeService(
  IDataServiceConfiguration config)
{
  config.SetEntitySetAccessRule(
    "Customers", EntitySetRights.All);
  config.SetEntitySetAccessRule(
    "Orders", EntitySetRights.All);
}
```

Let's see how it works if you have more ambitious plans and want to update data on the client before returning it to the service.

Dealing with Updates

Figure 4 shows the HTML template for a page fragment used to query for a customer. The user enters the ID, clicks the button and gets fresh, editable data.



My MVC Application

Demo in action

ID	Name	Contact	# Orders
EASTC	Eastern Connection	Ann Devon	8
CONSH	Consolidated Holdings	Elizabeth Brown	3
SEVES	Seven Seas Imports	Hari Kumar	9
NORTS	North/South	Simon Crowther	3
AROUT	Around the Horn	Thomas Hardy	13
BSBEV	B's Beverages	Victoria Ashworth	10

Figure 3 **Querying Data Using a WCF Data Service**

Data loading occurs on demand. Here's the code that takes care of invoking the data service:

```
function doLoad() {
  var id = Sys.get("#CustomerID").value;

  // Prepare the query
  var queryObject = new Sys.Data.OpenDataQueryBuilder("Customers");
  queryObject.set_filter("CustomerID eq '" + id + "'");
  var command = queryObject.toString();

  // Set up the DataView
  var dataView = Sys.get("$Editor").component();
  dataView.set_fetchOperation(command);
  dataView.fetchData();
}
```

You first get the input data you need—specifically, the text the user typed in the input field. Next, you prepare the query using a new OpenDataQueryBuilder object. Finally, you instruct the DataView (in turn configured to use the WCF Data Service) to download data for the query.

Any data retrieved is displayed using ASP.NET Ajax Library live binding, which guarantees live updates to any involved JavaScript objects (see **Figure 5**). The text box in which you edit the address of the customer is defined as:

```
<td class="caption">Address</td>
<td><%= Html.SysTextBox("Address", "{binding Address}") %></td>
```

In addition to the use of the {binding} expression, note the custom HTML helper being used in ASP.NET MVC. You might have a similar situation if you attempt to use live binding and AJAX templates in the context of a Web Forms application. So what's the problem?

For data binding to work, involved attributes must be prefixed with the sys: namespace. Therefore, to bind some text to a text box, you need to ensure that the following HTML markup is emitted:

```
<input type="text" ... sys:value="{binding Address}" />
```

In both ASP.NET MVC and Web Forms, you can brilliantly solve the problem by entering HTML literals. Otherwise, you need an adapted version of the tools that the ASP.NET framework of choice

The XTRAGRID for WINFORMS

by DevExpress

FEATURE COMPLETE PRESENTATION COMPONENTS • EASY-TO-USE REPORTING CONTROLS
IDE PRODUCTIVITY TOOLS • BUSINESS APP FRAMEWORKS

LEARN MORE and DOWNLOAD YOUR FREE EVALUATION COPY TODAY
VISIT **DEVEXPRESS.COM/XTRAGRID**

offers for abstracting pieces of markup: HTML helpers or server controls. In particular, in ASP.NET MVC, you may resort to a custom HTML helper that emits the sys:value attribute, as shown in **Figure 6**.

Changes to the address of the displayed customer are recorded as they happen and are tracked by the data context object. Note that this is possible only if you use the data context object as the data provider of the DataView used for rendering. This is the additional work that the OpenDataContext object can do for you with respect to the aforementioned OpenDataServiceProxy object.

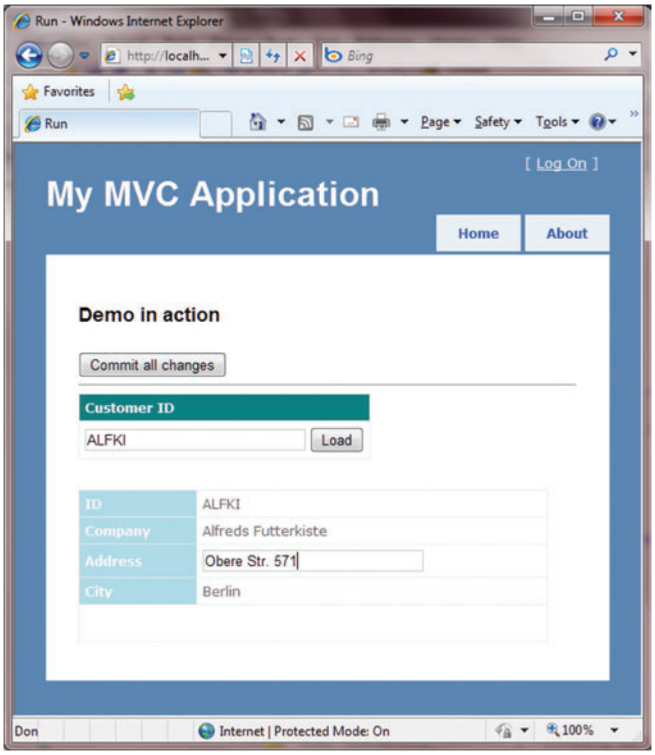
How can you save changes? To ensure that the modified delta of the downloaded data is served back to the data service, all you need to do is invoke the saveChanges method on the data context instance. Depending on the type of application you are building, though, you might want to add some extra layers of control. For example, you might want to add a "commit" button that first summarizes what's going on and asks users to confirm that they want to save pending changes. **Figure 7** shows the JavaScript code for such a commit button.

The function checks with the current data context to see if there are any pending changes. If so, it builds a summary of detected changes. The get_changes method on the data context returns an array of objects with information about the type of action (insert, remove or update) and the local object that was involved in the change. **Figure 8** shows the dialog box that results from the preceding code when you attempt to commit pending changes.

It should be noted that every time you select a new customer, you lose the changes of the previous one. This is because the data context is emptied and refilled with other data. Persisting changes in some other object just doesn't make sense—you'll be rewriting a clone of the data context yourself.

Figure 4 **Querying a Data Service**

```
<div id="Demo2">
<table>
   <tr class="tableHeader"><td>Customer ID</td></tr>
   <tr><td>
     <%= Html.TextBox("CustomerID", "ALFKI") %>
     <input type="button" value="Load" onclick="doLoad()" />
   </td></tr>
 </table>

 <br /><br />

 <table sys:attach="dataview" id="Editor"
       class="sys-template"
       dataview:dataprovider="{{ dataContext }}"
       dataview:autofetch="false">
   <tr>
     <td class="caption">ID</td>
     <td>{{ CustomerID }}</td>
   </tr>
   <tr>
     <td class="caption">Company</td>
     <td>{{ CompanyName }}</td>
   </tr>
   <tr>
     <td class="caption">Address</td>
     <td>
        <%=Html.SysTextBox("Address", "{binding Address}")%></td>
   </tr>
   <tr>
     <td class="caption">City</td>
     <td>{{ City }}</td>
    </tr>
</table>
</div>
```



My MVC Application

Demo in action

Commit all changes

Customer ID	
ALFKI	Load

ID	ALFKI
Company	Alfreds Futterkiste
Address	Obere Str. 571
City	Berlin

Figure 5 **Editing an Object Locally**

The power of the client-side proxy of a WCF Data Service doesn't show up really well through a single-object user interface. In the ASP.NET Ajax Library Beta kit, you'll find an excellent way to test this feature: the ImageOrganizer example. However, I can give you the gist of what I mean by simply extending the present example a bit. Let's suppose you have a master-detail view and can switch from one customer's view to the next without leaving the page and without necessarily having to save changes. The download occurs only once (or periodically), and for the time it remains in memory, all changes your user interface allows are correctly tracked (see **Figure 9**).

Insertions and Deletions

So far, I've only focused on updates. But what about insertions and deletions? These have some slight differences, and require a bit more work. First and foremost, you can't rely on data binding to make changes to the underlying object being displayed. Your responsibility is to update the in-memory collection (or object) you received from the data context being used within the user interface. For inserts, you

Figure 6 **A Custom HTML Helper**

```
public static string SysTextBox(this HtmlHelper htmlHelper,
    string name,
    string value,
   IDictionary<string, object> htmlAttributes)
{
    var builder = new TagBuilder("input");
    builder.MergeAttributes(htmlAttributes);
    builder.MergeAttribute("type", "text");
    builder.MergeAttribute("name", name, true);
    builder.MergeAttribute("id", name, true);
    builder.MergeAttribute("sys:value", value, true);
    return builder.ToString(TagRenderMode.SelfClosing);
}
```

LEADTOOLS®

The World Leader in Imaging Development SDKs

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

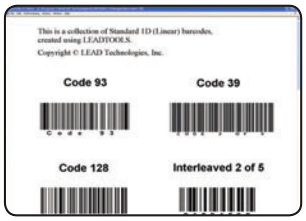
Develop your application with the same robust imaging technologies used by **Microsoft, HP, Sony, Canon, Kodak, GE, Siemens,** the **US Air Force** and **Veterans Affairs Hospitals.**

LEADTOOLS provides developers easy access to decades of expertise in color, grayscale, document, medical, vector and multmedia 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.

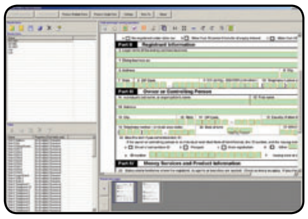
Multimedia

Barcode



This is a collection of Standard 1D (Linear) barcodes, created using LEADTOOLS.
Copyright © LEAD Technologies, Inc.

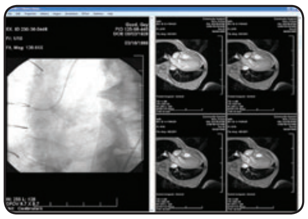
Form Recognition & Processing



Mark-up

Document

DICOM Medical



High Level Design • Low Level Control

LEAD TECHNOLOGIES INCORPORATED

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

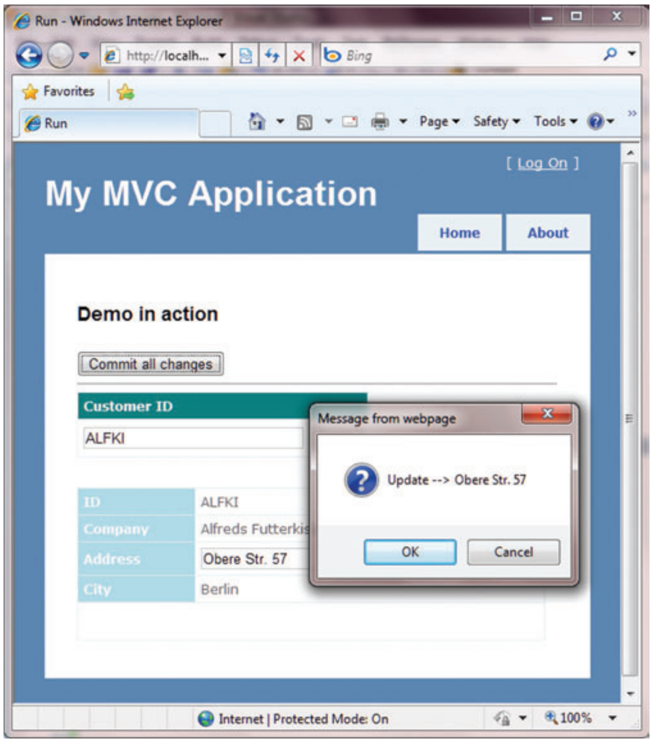
Figure 7 **A Commit Button to Confirm Changes**

```
function doCommit() {
    var pendingChanges = dataContext.get_hasChanges();
    if (pendingChanges !== true) {
        alert("No pending changes to save.");
        return;
    }

    var changes = dataContext.get_changes();
    var buffer = "";
    for (var i = 0; i < changes.length; i++) {
        ch = changes[i];

      // Function makeReadable just converts the action to readable text
        buffer += makeReadable(ch.action) +
                  " --> " +
                  ch.item["Address"];
        buffer += "\n";
    }
    if (confirm(buffer))
        dataContext.saveChanges();
}
```

simply need to create a new local instance of the object that is good for display and add it to the bound collection. At this point, if your user interface is fully data-bound, it should be able to reflect the change. Next, you need to inform the data context that a new object has been added to an entity set and needs to be tracked for persistence. Here's the typical code you need to attach to the JavaScript button that inserts an object:

```
// Create a new local object
var newCustomer = { ID: "DINOE", CompanyName: "...", ... };

// Add it to the collection used for data binding
dataView.get_data().add(newCustomer);

// Inform the data context object
dataContext.insertEntity(newCustomer, "Customers");
```

Removing an object is even simpler. You remove the object from



Message from webpage

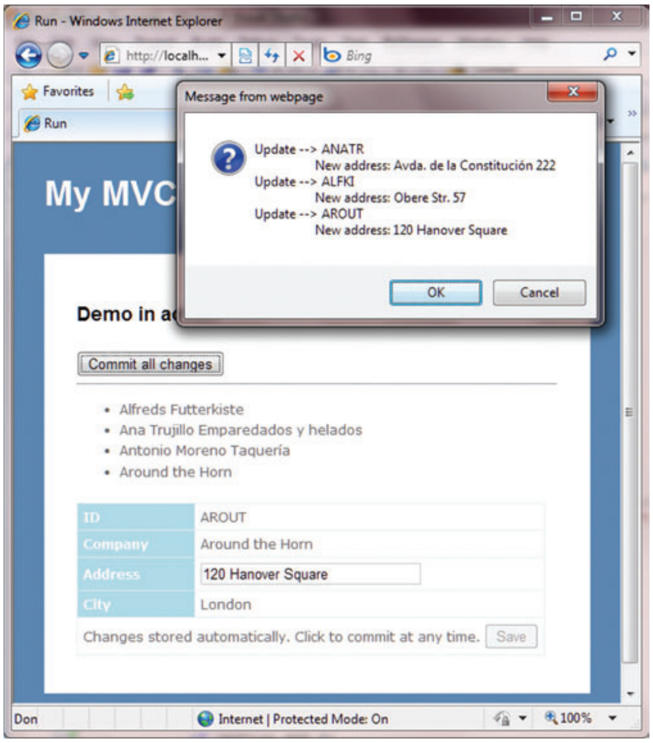
Update --> Obere Str. 57

OK Cancel

ID	ALFKI
Company	Alfreds Futterki
Address	Obere Str. 57
City	Berlin

Customer ID: ALFKI

Figure 8 **Pending Changes Detected**



Message from webpage

Update --> ANATR
New address: Avda. de la Constitución 222
Update --> ALFKI
New address: Obere Str. 57
Update --> AROUT
New address: 120 Hanover Square

OK Cancel

- Alfreds Futterkiste
- Ana Trujillo Emparedados y helados
- Antonio Moreno Taquería
- Around the Horn

ID	AROUT
Company	Around the Horn
Address	120 Hanover Square
City	London

Changes stored automatically. Click to commit at any time. Save

Figure 9 **Tracking Client-Side Changes**

the in-memory collection and call the removeEntity method on the data context.

```
var index = customerList.get_selectedIndex();
var customer = dataView.get_data()[index];
dataContext.removeEntity(customer);
imageData.remove(customer);
```

Avoid Confusion

The OpenDataContext and DataView objects work well together but should not be confused with each other. The OpenDataContext object represents the client-side proxy of a remote WCF Data Service. It's a very special type of proxy, however. It implements the "Unit of work" pattern on the client side as it tracks changes being made to any entities it helped retrieve. The data context is an excellent data provider for the DataView component. The DataView component is exclusively concerned with rendering. It offers plug-ins for templates to invoke remote operations easily, but that's just a facility for developers. No such CRUD and data management logic belong to the DataView.

This article didn't delve into the intricacies of WCF Data Services and didn't touch on aspects such as concurrency, lazy loading and security. It didn't discuss data transfers, either. Hopefully, this article serves as an up-to-date summary of how to do some important things with the ASP.NET Ajax Library and WCF Data Services. The remainder is good fodder for future articles. Stay tuned! ■

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

THANKS to the following technical experts for reviewing this article: *Boris Rivers-Moore and Stephen Walther*

/n software

We didn't invent the Internet...

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

applications

powered by

connectivity

powered by

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

Each day, as you click around the Web or use any connected application, chances are that directly or indirectly some bits are flowing through applications that use our components, on a server, on a device, or right on your desktop. It's your code and our code working together to move data, information, and business. We give you the most robust suite of components for adding Internet Communications, Security, and E-Business Connectivity to any application, on any platform, anywhere, and you do the rest. Since 1994, we have had one goal: to provide the very best connectivity solutions for our professional developer customers. With more than 100,000 developers worldwide using our software and millions of installations in almost every Fortune 500 and Global 2000 company, our business is to connect business, one application at a time.

To learn more please visit our website → www.nsoftware.com

Microsoft .NET, | Java | ActiveX | C++ | Delphi | C++ Builder | PHP | Objective-C | Windows | Windows Mobile | Mac OS X | Linux/Unix | iPhone

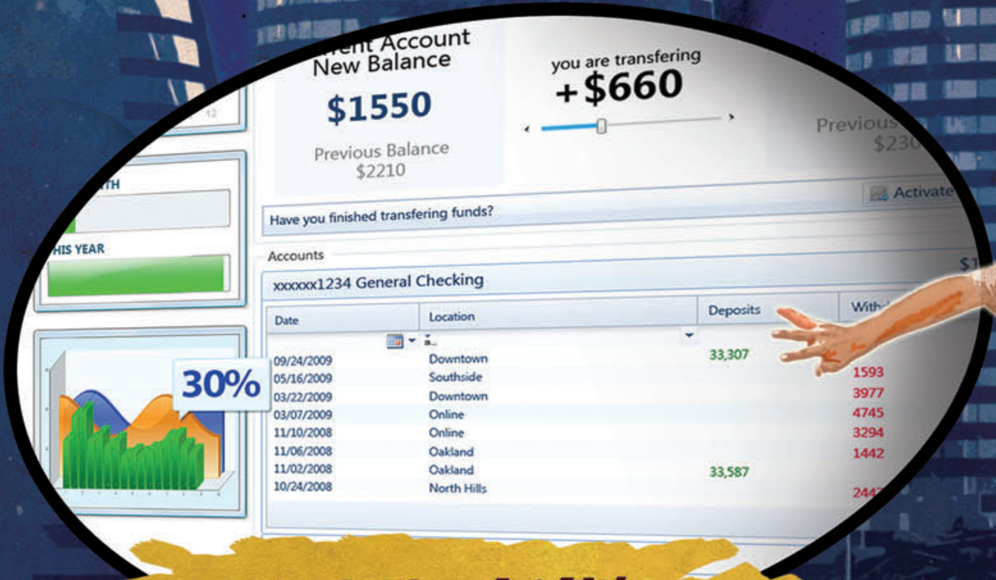
WINDOWS 7 IS HERE... VISUAL STUDIO 2010 IS COMING
PREPARE FOR THE WORLD OF
TOMORROW
WITH STUDIO ENTERPRISE 2009 v3

100'S OF CONTROLS, 7 PLATFORMS, FEATURING

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

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

DEVELOPER APPROVED FOR 20+ YEARS!



Have you finished transfering funds?

Accounts

xxxxxx1234 General Checking

Date	Location	Deposits	With
09/24/2009	Downtown	33,307	
05/16/2009	Southside		1593
03/22/2009	Downtown		3977
03/07/2009	Online		4745
11/10/2008	Online		3294
11/06/2008	Oakland		1442
11/02/2008	Oakland	33,587	
10/24/2008	North Hills		244

ASP.NET! • AJAX!

WPF!

SILVERLIGHT!

DOWNLOAD YOUR FREE TRIAL AT

WWW.COMPONENTONE.COM/PREPARE

ComponentOne **ComponentOne Sales:** 1.800.858.2739 or 1.412.681.4343

Migrating an APTCA Assembly to the .NET Framework 4

In the Microsoft .NET Framework 4, the common language runtime (CLR) security model has undergone some substantial changes. One of these changes, the adoption of Level2 transparency (much like Silverlight's security model), is likely to impact authors of AllowPartiallyTrustedCallers (APTCA) libraries.

The reason for the impact is that the underlying workings of APTCA have changed in CLR v4. The APTCA attribute retains the ability to make a fully trusted library available to partially trusted callers, but the details of how that happens are different, and some modifications to APTCA library code are likely necessary as a result.

Note: references to v2 in this article refer to CLR v2, which includes everything from the .NET Framework versions 2.0 through 3.5 SP1.

APTCA Before V4

Prior to v4, all signed assemblies were protected from partially trusted callers by implicit link demands for full trust on all entry points. This meant any partially trusted code attempting to access a strong-named assembly would fail with a security exception. This prevented (potentially dangerous) fully trusted code from being called maliciously from partial trust.

The new CLR v4 transparency system provides the same protection of full trust code as the old link demands.

Adding the AllowPartiallyTrustedCallers attribute to a signed full-trust library made it available to partial trust by removing these implicit link demands. Consequently, APTCA libraries allowed partially trusted code controlled access to privileged operations through APTCA-exposed methods. It was the responsibility of APTCA authors to ensure that only safe operations were exposed to partial trust in this way, and that any potentially dangerous operations were protected with explicit link demands or full demands.

APTCA in V4

The AllowPartiallyTrustedCallers attribute has changed. In v4, it no longer has anything to do with link demands. In fact, the implicit link demand that was present on signed libraries in v2 is gone. Instead, all fully trusted assemblies in v4 are, by default, SecurityCritical. On the other hand, all partially trusted assemblies are automatically Security-Transparent in v4. As explained in the transparency overview in the next section, SecurityCritical code can't be called from Security-Transparent code.

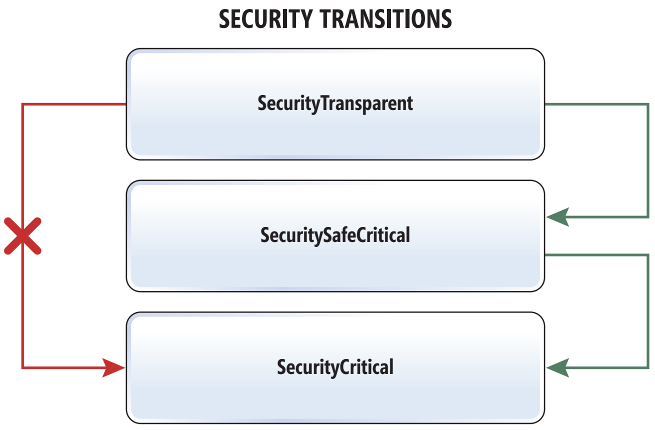


Figure 1 **Interactions of SecurityTransparent, SecuritySafeCritical and SecurityCritical Code**

Thus, the new v4 transparency system provides the same protection of full trust code as the old link demands; because of the automatic SecurityCritical and SecurityTransparent transparency levels, partially trusted code can't call into fully trusted libraries by default.

As you may have guessed, the v4 change to AllowPartiallyTrusted-Callers is related to this. In v4, the effect of APTCA is to remove the automatic SecurityCritical behavior from the assembly to which it's applied. The assembly then defaults to SecurityTransparent, but allows the APTCA assembly author to apply more granular SecurityCritical and SecuritySafeCritical attributes to specific types and methods as necessary.

Crash Course in Transparency

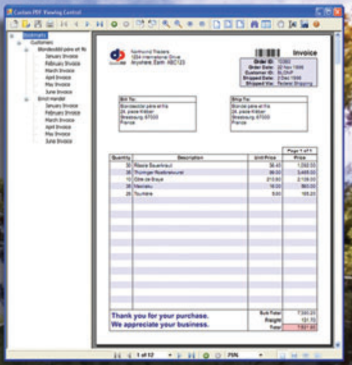
The effect of transparency attributes like SecurityTransparent and SecurityCritical will be known to readers familiar with the

This article discusses a prerelease version of the Microsoft .NET Framework 4. All information is subject to change.

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

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.

Try our three new products FREE today!

Fully functional and never expiring evaluation editions available at www.cete.com/download

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.

ceTesoftware
INFINITE POSSIBILITIES

ceTe Software has been delivering quality software applications and components to our customers for over 10 years. Our DynamicPDF product line has proven our commitment to delivering innovative software components and our ability to respond to the changing needs of software developers. We back our products with a first class support team trained to provide timely, accurate and thorough responses to any support needs.

www.cete.com
info@cete.com

800.631.5006
+1 410.772.8620

Silverlight security model, because the new v4 transparency model is quite similar.

Let's take a look at the three primary transparency attributes: SecurityTransparent, SecuritySafeCritical and SecurityCritical.

SecurityTransparent Code marked as SecurityTransparent is safe from a security perspective. It's unable to complete any dangerous operations, such as asserting a permission, executing unverifiable code or calling native code. It's also unable to call SecurityCritical code directly.

Prior to CLR v4, all signed assemblies were protected from partially trusted callers by implicit link demands for full trust on all entry points.

- As noted, all partial trust code is forced to be SecurityTransparent for security reasons. It's also the default transparency of APTCA libraries.

SecurityCritical SecurityCritical code, by contrast, is able to perform any operations it wishes. It can assert, call native code and more. It can call other methods regardless of transparency markings.

- Only fully trusted code can be SecurityCritical. And, in fact, (non-APTCA) fully trusted code is assumed to be SecurityCritical, by default, to protect it from transparent, partially trusted callers.

SecuritySafeCritical SecuritySafeCritical code acts as a bridge, allowing transparent code to call into critical methods. SecuritySafeCritical code has all the same rights as SecurityCritical code, but it's callable from SecurityTransparent code. It is, therefore, extremely important that SecuritySafeCritical code expose underlying SecurityCritical methods only in a safe manner (lest some malicious, partially trusted code attempts to exploit the methods through the SecuritySafeCritical layer).

- Like SecurityCritical code, SecuritySafeCritical code must be fully trusted.

Figure 1 illustrates the interactions of SecurityTransparent, SecuritySafeCritical and SecurityCritical code.

Note that in addition to the transitions shown in the diagram, all transparency levels can access themselves and any less critical code (for example, SecuritySafeCritical code can access SecurityTransparent code). Because the AllowPartiallyTrustedCallers attribute causes the entire assembly to be SecurityTransparent by default, the assembly's author must specifically mark methods needing to perform privileged operations as SecurityCritical or SecuritySafeCritical. Without such marking, the APTCA author will find that his code fails with MethodAccessExceptions, TypeAccessExceptions and other errors indicating that the APTCA library is attempting to call dangerous APIs from SecurityTransparent code.

This is just a brief introduction to the model; you'll find a more thorough examination in MSDN documentation and in a previous CLR Inside Out article by Andrew Dai, available at msdn.microsoft.com/magazine/8e75546c-416a-44e1-8462-e39205fb942a.

Migrating from V2 to V4: Which Attributes to Apply

Most of the work necessary to migrate a v2 APTCA assembly to v4 involves identifying and applying the correct transparency attributes to methods that need them. Following are guidelines indicating when each of the attributes is appropriate.

SecurityTransparent Code that does not perform any security-sensitive operations should be SecurityTransparent.

Unlike the other transparency settings, SecurityTransparent behavior is the default in an APTCA assembly and, therefore, does not need to be marked explicitly. Code is considered transparent in the absence of other attributes.

One advantage of transparent code is that it's safe (because dangerous operations are disallowed) and, as a result, it does not need as thorough a security review as SecurityCritical or, especially, SecuritySafeCritical code. It's recommended that as much code as possible be SecurityTransparent.

The following are disallowed in SecurityTransparent code:

- Calling SecurityCritical methods
- Asserting a permission or permission set
- Unverifiable code
- Calling unmanaged code
- Overriding SecurityCritical virtual methods
- Implementing SecurityCritical interfaces
- Deriving from any type that is not SecurityTransparent

SecuritySafeCritical Code that is callable from partial trust but needs to be able to call potentially dangerous APIs should

Figure 2 **Output from Security Annotator**

```
<requiredAnnotations>
  <assembly name="Logging">
    <type name="Logging">
      <method name="MethodA()">
        <annotations>
          <safeCritical>
            <rule name="MethodsMustOverrideWithConsistentTransparency">
              <reason pass="2" sourceFile="d:\repro\aptca\logging.
cs" sourceLine="67">Critical method Logging.MethodA()' is overriding
transparent or safe critical method 'Logging.MethodA()' in violation of
method override rules.  Logging.MethodA()' must become transparent or
safe-critical in order to override a transparent or safe-critical virtual
method or implement a transparent or safe-critical interface method.</
reason>
            </rule>
          </safeCritical>
          <critical>
            <rule name="TransparentMethodsMustNotSatisfyLinkDemands">
              <reason pass="1" sourceFile="d:\repro\aptca\logging.cs"
sourceLine="68">Security transparent method Logging.MethodA()' satisfies
a LinkDemand for 'FileIOPermissionAttribute' on method 'Logging.set_
LogLocation(System.String)'.  Logging.MethodA()' should become critical
or safe-critical in order to call 'Logging.set_LogLocation(System.
String)'.</reason>
            </rule>
          </critical>
        </annotations>
      </method>
    </type>
  </assembly>
</requiredAnnotations>
```

Figure 3 **V2 APTCA Library**

```
using System;
using System.IO;
using System.Security;
using System.Security.Permissions;

// This assembly is meant to be representative of a simple v2 APTCA
assembly
// It has some dangerous code protected with demands/link demands
// It exposes some dangerous code in a controlled way with an assert

[assembly: AllowPartiallyTrustedCallers]
public class Logging
{
    private string logLocation = @"C:\temp\firstfoo.txt";

    public virtual string Usage()
    {
        return "This is a helpful string";
    }

    public virtual string LogLocation
    {
        get
        {
            return logLocation;
        }

        [FileIOPermissionAttribute(SecurityAction.LinkDemand,
Unrestricted=true)]
        set
        {
            logLocation = value;
        }
    }

    public virtual void SetLogLocation(int index)
    {
        switch (index)
        {
            case 1:
                LogLocation = @"C:\temp\foo.txt";
                break;
            case 2:
                LogLocation = @"D:\temp\foo.txt";
                break;
            case 3:
                LogLocation = @"D:\repro\temp\foo.txt";
                break;
            default:
                break;
        }
    }


    public virtual void DeleteLog()
    {
        FileIOPermission fp = new FileIOPermission(FileIOPermissionAcce
ss.AllAccess, LogLocation);
        fp.Assert();
        if (File.Exists(LogLocation)) { File.Delete(LogLocation); }
        SecurityPermission.RevertAll();
    }

    // TODO : Put other APIs (creating log, writing to log, etc) here
}

public class OtherLogging : Logging
{
    public override string Usage()
    {
        LogLocation = null;
        return "This is a different useful string";
    }

    // TODO : Put other APIs (creating log, writing to log, etc) here
}
```

be marked as SecuritySafeCritical. Often, methods that demand permissions will fall into this category because they represent a protected boundary between partially trusted code and privileged operations.

Because SecuritySafeCritical code allows partially trusted callers to indirectly access dangerous APIs, it's a very powerful attribute and should be applied carefully and sparingly. It's important that SecuritySafeCritical code expose SecurityCritical functionality to its callers only in specific, safe ways. It's usually a good idea for SecuritySafeCritical code to contain demands to ensure that callers can access particular resources that the SecuritySafeCritical code will be using. It's also important for SecuritySafeCritical code to validate both inputs and outputs (to be sure that invalid values are not passed through, and that any returned information is safe to give to partial trust).

Because of the potential security risks, it's recommended that SecuritySafeCritical code be kept to a minimum.

SecurityCritical Code that is not safe to expose to partially trusted callers should be marked as SecurityCritical. Methods that previously were protected by a link demand are likely to require this attribute.

SecurityCritical code is less dangerous than SecuritySafe-Critical because it's not directly callable from transparent (partially trusted) callers. However, the code can perform many high-security operations so, in order to keep the need for security reviews to a minimum, it's a good idea to keep SecurityCritical code to a minimum, as well.

Good general guidance is that any code that can be Security-Transparent should be. Other code should be SecurityCritical unless it's specifically expected that transparent code will access SecurityCritical code through it, in which case SecuritySafe-Critical is appropriate.

There are three primary transparency attributes: SecurityTransparent, SecuritySafeCritical and SecurityCritical.

Using SecAnnotate.exe

To help with the correct application of transparency attributes, there is a new .NET Framework SDK tool, the Security Annotator (SecAnnotate.exe). This tool consumes a user's binary (or collection of binaries) and provides guidance on where transparency attributes should be applied. It can be very helpful when migrating an APTCA library to v4.

SecAnnotate works by making several passes through the target binary, looking for methods that, according to the CLR's rules, need to be marked with a transparency attribute. On subsequent passes, the tool looks for attributes that are necessary because of modifications suggested in previous passes. For example, consider this short code snippet (which is assumed to come from an APTCA assembly):

```
static void Method1()
{
      Console.WriteLine("In method 1!");
      Method2();
}

static void Method2()
{
      PermissionSet ft = new PermissionSet(PermissionState.
Unrestricted);
      ft.Assert();
      DangerousAPI();
      PermissionSet.RevertAssert();
}
```

SecAnnotate.exe would immediately notice that Method2 can't be transparent because it asserts for some permissions. After the first pass, the tool would know that Method2 must be either SecurityCritical or SecuritySafeCritical (with SecurityCritical being preferred unless transparent code needs to specifically access this method).

It's recommended that as much code as possible be SecurityTransparent.

On the first pass through the binary, Method1 would not seem interesting to the tool. On the second pass, however, it would be noted that Method1 is calling Method2, which, during the first pass, SecAnnotate suggested become SecurityCritical. Because of this, Method1 would also need to be SecurityCritical (or, at the author's discretion, SecuritySafeCritical). After two passes, the guidance to mark as SecurityCritical would be given for both methods.

Understanding SecAnnotate.exe Output

Security Annotator's output is an XML file that contains the problems it has identified and the recommended fixes. Sometimes, Security Annotator reverses an earlier recommendation after subsequent passes. In such cases, both recommendations appear in the XML. You'll need to look at the pass number in these cases to understand which recommendation is more recent and, therefore, correct.

For example, consider the output from Security Annotator in **Figure 2**. Notice that there are two elements under the annotations tag for method Logging.MethodA—a SecuritySafeCritical tag and a SecurityCritical tag. This means that SecAnnotate recommended both SecurityCritical and SecuritySafeCritical attributes for this method during its analysis.

The SecurityCritical element's explanation says that because this method is calling something protected with a link demand, it must be either SecurityCritical or SecuritySafeCritical. SecAnnotate.exe defaults to recommending SecurityCritical because it's more secure. Note that the pass attribute has a value of 1 here, meaning this suggestion resulted from SecAnnotate.exe's first pass through the code.

The next recommendation—for SecuritySafeCritical—notes that MethodA is overriding a transparent base method and so must be SecurityTransparent or SecuritySafeCritical (it must have the same accessibility as the base method). Putting this information together with the previous recommendation, SecAnnotate.exe suggests that MethodA should be SecuritySafeCritical.

Note that pass="2" means this recommendation came during SecAnnotate.exe's second pass through the code. This is because during the first pass, the tool didn't know that MethodA could not be transparent, so it wasn't aware of this SecuritySafeCritical requirement.

Because the SecuritySafeCritical recommendation was made during the second (more recent) pass, it is the correct annotation in this case.

SecAnnotate.exe Best Practices

In cases where SecurityCritical and SecuritySafeCritical would both be correct markings, Security Annotator first prefers any attribute already on the code, and then SecurityCritical because it's less risky. Unfortunately, this often results in code that is secure but unusable in a sandbox because all entry points are blocked to partially trusted callers.

Remember that SecuritySafeCritical is appropriate on APIs that are meant to be called directly from transparent/partial-trust code and have been reviewed for security with that in mind. Because Security Annotator can't know which APIs are meant to be called from partial trust or are safe to be called in that way, it will mark very little as SecuritySafeCritical. The library's author must *manually* apply the SecuritySafeCritical attribute to some methods, even when using Security Annotator.

Because a single action prohibited in transparent code can "spider web" out into many SecurityCritical markings in Security Annotator's successive passes without strategic placement of the SecuritySafeCritical attribute, it's good practice to use SecAnnotate.exe with the /p command-line switch. The switch /p:x (where x is a number) instructs Security Annotator to run only x passes, instead of running until no more changes are necessary. Here is good way to use Security Annotator:

1. Run SecAnnotate.exe /p:1 /d:<Path to referenced assemblies> <FileName.dll>
   a. This adds transparency attributes where needed, but only with a single pass. Stopping at this point allows the author to manually check the attributes.
   b. By default, SecAnnotate.exe looks only in the GAC for dependencies of the assembly it's annotating. Other assemblies must have their paths specified with the /d switch.
2. Update the library's source files with the suggested attributes. Consider cases with multiple possible attributes, though, and decide which is correct. In some cases, SecuritySafeCritical will be the correct attribute, despite SecAnnotate favoring SecurityCritical.
3. Rebuild the assemblies and repeat at step 1 without /p:1. You could rerun the program using /p:1 repeatedly, but you shouldn't need to—the necessary SecuritySafeCritical attributes are already present after the first iteration of step 2.

Figure 4 **V4 APTCA Library**

```
using System;
using System.IO;
using System.Security;
using System.Security.Permissions;

// This assembly is meant to be representative of a simple v2 APTCA
assembly
// It has some dangerous code protected with demands/link demands
// It exposes some dangerous code in a controlled way with an assert

[assembly: AllowPartiallyTrustedCallers]
public class Logging
{
    private string logLocation = @"C:\temp\firstfoo.txt";

    // This API can be transparent because it does nothing dangerous.
    // Transparent APIs need no attributes because it is the default
behavior of a v4
    // APTCA assembly
    public virtual string Usage()
    {
        return "This is a helpful string";
    }

    // Note that transparency attributes do not go directly on
properties.
    // Instead, they go on the getters and setters (even if the getter
and setter
    // get the same attributes)
    public virtual string LogLocation
    {
        get
        {
            return logLocation;
        }

        // This API is made critical because it sets sensitive data (the
path to write to)
        // which partial trust code should not be able to do.
        [SecurityCritical]
        // The previous LinkDemand is removed as the SecurityCritical
attribute replaces it
         //[FileIOPermissionAttribute(SecurityAction.LinkDemand,
Unrestricted=true)]
        set
        {
            logLocation = value;
        }
    }

    // This API accesses a critical member (LogLocation) and, therefore,
cannot be transparent
    // However, the access is done in a limited, safe way and we expect
transparent code
    // should be able to call this API. Therefore, it is
SecuritySafeCritical
    [SecuritySafeCritical]
    public virtual void SetLogLocation(int index)
    {
        switch (index)
        {
            case 1:
                LogLocation = @"C:\temp\foo.txt";
                break;
            case 2:
                LogLocation = @"D:\temp\foo.txt";
                break;
            case 3:
                LogLocation = @"D:\repro\temp\foo.txt";
                break;
            default:
                break;
        }
    }

    // This API is potentially dangerous; it asserts which means it can't
be transparent
    // Because setting LogLocation is protected, however, partial trust
code can safely
    // call this API. In fact, it is intended that it is safe for partial
trust code
    // to call this method. Therefore, it is SecuritySafeCritical
    [SecuritySafeCritical]
    public virtual void DeleteLog()
    {
        FileIOPermission fp = new FileIOPermission(FileIOPermissionAcce
ss.AllAccess, LogLocation);
        fp.Assert();
        if (File.Exists(LogLocation)) { File.Delete(LogLocation); }
        SecurityPermission.RevertAll();
    }

    // TODO : Put other APIs (creating log, writing to log, etc) here
}

public class OtherLogging : Logging
{
    // The logic for attributing this method is complicated and it is an
example of when
    // SecAnnotate.exe can be very helpful. This API cannot be
transparent because it
    // calls a critical member (LogLocation). However, because it
overrides a transparent
    // method (Usage) it cannot be critical. Therefore, the only possible
annotation here
    // is SecuritySafeCritical and it is the author's responsibility to
make sure that
    // a malicious caller cannot abuse that access.
    [SecuritySafeCritical]
    public override string Usage()
    {
        LogLocation = null;
        return "This is a different useful string";
    }

    // TODO : Put other APIs (creating log, writing to log, etc) here
}
```

This iterative process with manual developer interaction will result in a correctly annotated assembly that maximizes transparent code.

Identifying and Reviewing SecuritySafeCritical APIs

As noted earlier, it's common for SecAnnotate.exe to recommend that an API should be either SecurityCritical or SecuritySafe-Critical. The key differentiator is whether the API can safely be called from partial trust. If the API does all validation necessary to be sure that underlying critical or native APIs will be called safely (through demands or input and output validation, for example), then it can be SecuritySafeCritical, which is sometimes desirable because it allows callers of the API to be transparent. If, on the other hand, there is any way that malicious code could access protected resources through the API, the API must remain SecurityCritical.

It's important that all SecuritySafeCritical code be carefully reviewed for security impact of exposure to partial trust. Although both SecuritySafeCritical and SecurityCritical code should be minimized, if there is doubt as to which attribute is correct, Security-Critical is the safer option.

Applying Transparency Attributes

Applying transparency attributes is as simple as applying any other .NET attributes in code. Documentation regarding usage for the attributes can be found in MSDN documentation for these types:

- SecurityTransparentAttribute

Note that this attribute may be applied only at the assembly level. Its meaning, in that case, is that all types and methods in the assembly are transparent. It's unnecessary at the type or method level because it's the default transparency setting in APTCA assemblies.

- SecuritySafeCriticalAttribute
- SecurityCriticalAttribute

In C#, applying the attributes looks like this:

```
[SecurityCritical]
public static void Method1()
{ /* Do something potentially dangerous*/ }

[SecuritySafeCritical]
public static void Method2()
{ /* Do something potentially dangerous in a safe way that can be called
from partial trust */ }
```

To help with the correct application of transparency attributes, there is a new .NET Framework SDK tool, the Security Annotator (SecAnnotate.exe).

Level1 and Level2

One final note regarding transparency and APTCA is that it's possible, through the use of an assembly-level attribute, to use the old v2 APTCA behavior instead of the new v4 behavior. This is not recommended because the new model is more secure, easier to audit and common between Silverlight and desktop CLRs. Still, sometimes compatibility is needed in the short term until you can migrate. In these cases, the SecurityRules attribute can be used to force an assembly to use old v2 rules.

The SecurityRules attribute takes a parameter of the Security-RuleSet enum type. SecurityRuleSet.Level1 specifies compatibility. SecurityRuleSet.Level2 specifies the new model, but a Level2 attribute is not necessary because it's the default. It can be useful, however, to explicitly indicate which transparency rule set is in use and to protect against any future changes regarding which rule set is the .NET Framework's default.

In C#, application of this attribute appears like this:

```
[assembly:SecurityRules(SecurityRuleSet.Level1)]
```

Common Pitfalls

Following are a few common "gotchas" that APTCA library authors should be wary of as they migrate from v2 to v4:

- SecAnnotate.exe will recommend that LinkDemands become SecurityCritical attributes (which are very similar to a Link-Demands for FullTrust). If, however, a type (rather than a method) was protected with a LinkDemand, this is not the same as applying SecurityCritical to a type in v4. It's better to apply SecurityCritical to all members of the type, as this will act more like a v2 type-level LinkDemand.
- Note that some low-permissions LinkDemands that some partial trust code is expected to be able to satisfy may not be best translated into SecurityCritical. If a LinkDemand is for a low permission (for example, read permission to a particular safe path), it's better to remove the LinkDemand and replace it with a full demand for the same permission. This allows partial trust code to still call the API (but the demand will make sure that only partial trust code with high enough permissions succeeds in making the call).
- In general, type-level transparency attributes apply to members of the type they modify, as well. And the outermost attribute trumps others. So, applying [SecurityCritical] to a method is ineffectual if the type it's in has [SecuritySafeCritical] applied to it, for example. Typically, [SecuritySafeCritical] is not a useful attribute on the type level. It's too likely that someone will later introduce a new member to the type and not realize that it's SecuritySafeCritical (thanks to the type-level attribute), potentially resulting in a security hole.

Although type-level attributes apply to new slot members of the types they modify, they do not apply to overridden members. Be sure that if you use type-level transparency attributes, you also add attributes to overridden members specifically, as necessary.

Migration Example

Figure 3 is a simple (and incomplete) logging library written in v2. The same library is also shown in **Figure 4**, migrated to v4 with comments (in italics) explaining the changes.

Syncing CLR and the Silverlight CoreCLR Security Systems

Although the merging of APTCA and transparency in v4 may seem complex, it ultimately supplies straightforward and effective protection to sensitive system resources from partially trusted callers. What's more, the change aligns the desktop CLR and Silverlight CoreCLR security systems.

SDK tools such as SecAnnotate.exe and FxCop rules (which can validate transparency) help to make migration easier. V4 APTCA assemblies are much easier to audit—looking closely at Security-SafeCritical APIs (and the SecurityCritical calls they make) is all that is needed to have confidence that the assembly is secure.

Because transparent code often makes up 80 percent to 90 percent or more of an assembly, this is a great relief of audit burden. Readers interested in delving further into transparency can find more complete explanations in MSDN documentation. ■

MIKE ROUSOS *has been a software design engineer in test on Microsoft's CLR team since 2005. His work primarily deals with ensuring quality in the design and implementation of CLR security systems.*

THANKS to the following technical experts for reviewing this article: *Andrew Dai, Cristian Eigel and Shawn Farkas*

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.

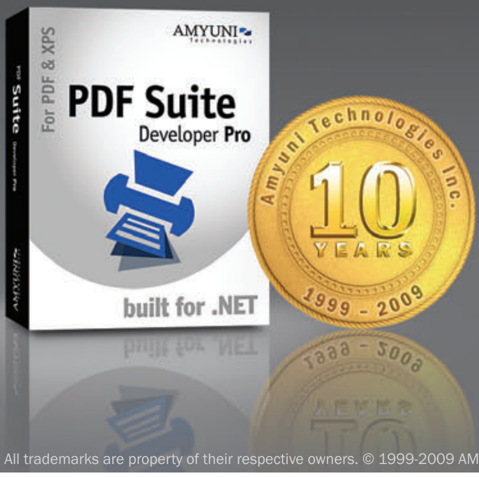
Customization

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

XCEED
DataGrid
for WPF

Serious recognition

Microsoft®
Visual Studio® Team System 2010

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

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

IBM®
U2 SystemBuilder™

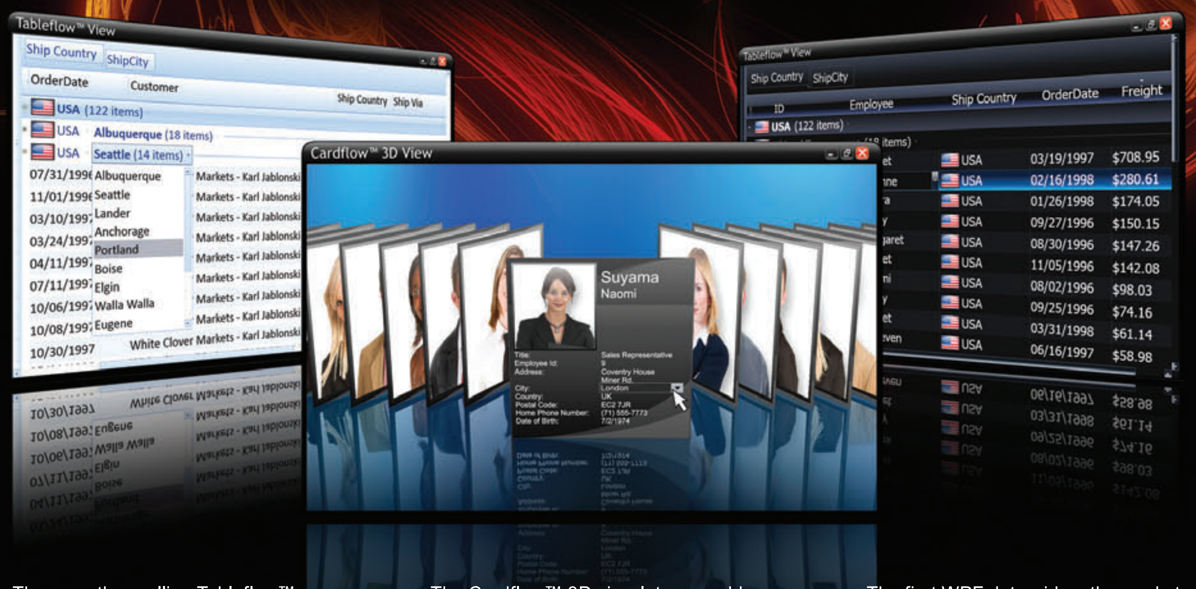
"IBM U2 researched existing third-party solutions and identified Xceed DataGrid for WPF as the most suitable tool. The datagrid's performance and solid foundation take true advantage of WPF and provide the extensibility required for IBM U2 customers to take their applications to the next level in presentation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

Vincent Smith,
U2 Tools Product Manager at IBM

XCEED
MULTI-TALENTED COMPONENTS

Datagrids, transformed

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



OrderDate	Customer	Ship Country	Ship Via
USA (122 items)			
USA Albuquerque (18 items)			
USA Seattle (14 items)			
07/31/1996	Albuquerque	Markets - Karl Jablonski	
11/01/1996	Seattle	Markets - Karl Jablonski	
03/10/1997	Lander	Markets - Karl Jablonski	
03/24/1997	Anchorage	Markets - Karl Jablonski	
04/11/1997	Portland	Markets - Karl Jablonski	
07/11/1997	Boise	Markets - Karl Jablonski	
10/06/1997	Elgin	Markets - Karl Jablonski	
10/08/1997	Walla Walla	Markets - Karl Jablonski	
	Eugene	Markets - Karl Jablonski	
10/30/1997	White Clover	Markets - Karl Jablonski	

Title: Sales Representative
Employee Id: 9
Address: Coventry House Miner Rd.
City: London
Country: UK
Postal Code: EC2 7JR
Home Phone Number: (71) 555-7773
Date of Birth: 7/2/1974

ID	Employee	Ship Country	OrderDate	Freight
USA (122 items)				
(18 items)				
	et	USA	03/19/1997	$708.95
	nne	USA	02/16/1998	$280.61
	a	USA	01/26/1998	$174.05
	y	USA	09/27/1996	$150.15
	garet	USA	08/30/1996	$147.26
	et	USA	11/05/1996	$142.08
	ni	USA	08/02/1996	$98.03
	y	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

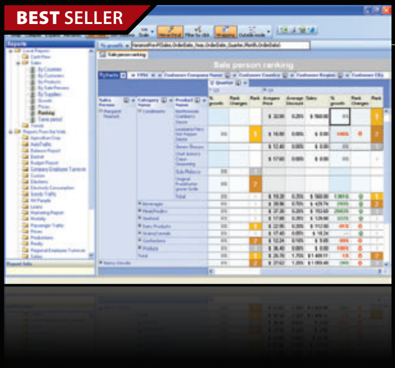
ComponentSource®
The Definitive Source of Software Components
www.componentsource.com

Janus WinForms Controls Suite | from $757.44

Janus systems

Add Outlook style interfaces to your WinForms applications.

- Janus GridEX for .NET (Outlook style grid)
- Janus Schedule for .NET and Timeline for .NET (Outlook style calendar view and journal)
- Janus ButtonBar for .NET and ExplorerBar for .NET (Outlook style shortcut bars)
- Janus UI Bars and Ribbon Control (menus, toolbars, panels, tab controls and Office 2007 ribbon)
- Now includes Office 2007 visual style for all controls



ContourCube | from $900.00

Contour Components

OLAP component for interactive reporting and data analysis.

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

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

TX TEXT CONTROL word processing components

Word processing components for Visual Studio .NET.

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

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

MasterCard VISA DISCOVER
GSA Schedule Contract GS-35F-0188R

Precompiling LINQ Queries

When using LINQ to SQL or LINQ to Entities in your applications, it's important to consider precompiling any query you create and execute repeatedly. I often get caught up in completing a particular task and neglect to leverage precompiled queries until I'm fairly far along in the development process. This is much like "exception handling disease," where devs try to shoehorn exception handling into an app after the fact.

However, even after you've implemented this important performance-enhancing technique, there's a good chance you're losing its benefit. You may notice that the promised performance gain isn't being realized, but the reason (and fix) might escape you.

In this column, I'll first explain how to precompile queries and then focus on the problem of losing the precompilation benefits in Web applications, services and other scenarios. You'll learn how to ensure that you're getting the performance benefit across postbacks, short-lived service operations and other code where critical instances are going out of scope.

Precompiling Queries

The process of transforming your LINQ query into the relevant store query (for example, the T-SQL executed by the database) is relatively expensive within the larger umbrella of query execution. **Figure 1** shows the involved process that transforms a LINQ to Entities query into a store query.

The Entity Framework team's blog post, "Exploring the Performance of the ADO.NET Entity Framework - Part 1" (blogs.msdn.com/adonet/archive/2008/02/04/exploring-the-performance-of-the-ado-net-entity-framework-part-1.aspx), breaks down the process and shows the relative time taken by each step. Note that this post was based on the Microsoft .NET Framework 3.5 SP1 version of Entity Framework, and the time-per-step distribution has likely shifted in the new version. Nevertheless, precompilation is still an expensive part of the query execution process.

By precompiling your query, Entity Framework and LINQ to SQL can reuse the store query and skip the redundant process of figuring it out each time. For example, if your app frequently retrieves different customers from the data store, you might have a query such as this:

```
Context.Customers.Where(c=>c.CustomerID==_custID)
```

When nothing but the _custID parameter changes from one execution to the next, why waste the effort of transposing this to a SQL command over and over again?

LINQ to SQL and Entity Framework both enable query precompilation; however, because of some differences between the processes in the two frameworks, they each have their own CompiledQuery class. LINQ to SQL uses System.Data.LINQ.CompiledQuery while Entity Framework uses System.Data.Objects.CompiledQuery. Both forms of CompiledQuery allow you to pass in parameters, and both require that you pass in the current DataContext or ObjectContext being used. Essentially, from the coding perspective, they're equal.

The CompiledQuery.Compile method returns a delegate in the form of a Func that can, in turn, be invoked on demand.

Here is a simple query being compiled by Entity Framework's CompiledQuery class, which is static and therefore doesn't require instantiation:

```
C#:  var _custByID = CompiledQuery.Compile<SalesEntities, int, Customer>
   ((ctx, id) =>ctx.Customers.Where(c=> c.ContactID == id).Single());
VB: Dim _custByID= CompiledQuery.Compile(Of SalesEntities, Integer,
Customer)
   (Function(ctx As ObjectContext, id As Integer)
    ctx.Customers.Where(Function(c) c.CustomerID = custID).Single)
```

You can use either LINQ methods or LINQ operators in the query expression. These queries are built with LINQ methods and lambdas.

The syntax is a little more confusing than your typical generic method, so I'll break it down. Again, the goal of the Compile method is to create a Func (delegate) that can be invoked at a later time, as shown here:

```
C#: CompiledQuery.Compile<SalesEntities, int, Customer>
VB: CompiledQuery.Compile(Of SalesEntities, Integer, Customer)
```

Because it's generic, the method must be told what types are being passed in as arguments, as well as what type will be returned when the delegate is invoked. Minimally, you must pass in some type of ObjectContext, or DataContext for LINQ to SQL. You can specify a System.Data.Objects.ObjectContext or something that derives from it. In my case, I'm explicitly using the derived class SalesEntities that is associated with my Entity Data Model.

You can also define multiple arguments, which must come directly after the context. In my example, I'm telling Compile that the resulting precompiled query should also take an int/Integer parameter. The last type describes what the query will be returning, in my case a Customer object:

```
C#: ((ctx, id) =>ctx.Customers.Where(c => c.ContactID == id).Single())
VB: Function(ctx As ObjectContext, id As Integer)
   ctx.Customers.Where(Function(c) c.CustomerID = custID).Single
```

The result of the previous compile method is the following delegate:

```
C#: private System.Func<SalesEntities, int, Customer> _custByID
VB: Private _custByID As System.Func(Of SalesEntities, Integer, Customer)
```

Code download available at code.msdn.microsoft.com/mag201003DataPoints.

Once the query has been compiled, you simply invoke it whenever you want to execute that query, passing in the ObjectContext or DataContext instance and any other required parameters. Here I have an instance named _commonContext and a variable named _custID:

```
Customer cust  = _custByID.Invoke(_commonContext, _custID);
```

The first time the delegate is invoked, the query is translated to the store query and that translation is cached for reuse on subsequent calls to Invoke. LINQ can skip the task of compiling the query and go right to execution.

Ensuring that Your Precompiled Query Is Truly Being Used

There's a not-so-obvious, and not widely known, problem with precompiled queries. Many developers presume that the query is cached in the application process and will stick around. I certainly made this assumption, because there was nothing I found to indicate otherwise—except for some unimpressive performance numbers. However, when the object where you instantiated the compiled query goes out of scope, you also lose the precompiled query. It will need to be precompiled again for each use, so you completely lose the benefit of the precompiling. In fact, you're paying a higher price than you would if you were simply executing a LINQ query, due to some of the extra effort the CLR must make with respect to the delegate.

Rico Mariani digs into the cost of using the delegate in his blog post, "Performance Quiz #13—Linq to SQL compiled query cost—solution" (blogs.msdn.com/ricom/archive/2008/01/14/performance-quiz-13-linq-to-sql-compiled-query-cost-solution.aspx). The discussion in the comments is equally enlightening.

I've seen blog reports about LINQ to Entities' "terrible performance" in Web apps "even with precompiled queries." The reason is that every time a page posts back, you're getting a newly instantiated context and *re*-precompiling the query. The precompiled query is never getting reused. You'll have the same problem anywhere you have short-lived contexts. This could happen in an obvious place, such as a Web or Windows Communication Foundation (WCF) service, or even in something less obvious, such as a repository that will instantiate a new context on the fly if an instance hasn't been provided.

You can avoid the loss of the delegate by using a static (Shared, in VB) variable to retain the query across processes, and then invoking it using whatever context is currently available.

Here's a pattern I've successfully used with Web applications, WCF services and repositories, where the ObjectContext goes out of scope frequently and I want the delegate to be available throughout the application process. You need to declare a static delegate in the constructor of the class where you'll be invoking queries. Here I'm declaring a delegate that matches the compiled query I previously created:

```
C#: static Func<ObjectContext, int, Customer> _custByID;
VB: Shared _custByID As Func(Of ObjectContext, Integer, Customer)
```

There are a few possible places to compile the query. You can do it in a class constructor or just prior to invoking it. Here is a method that is designed to perform a query and return a Customer object:

```
    public static Customer GetCustomer( int ID)
     {
       //test for an existing instance of the compiled query
       if (_custByID == null)
       {
         _custByID = CompiledQuery.Compile<SalesEntities, int, Customer>
          ((ctx, id) => ctx.Customers.Where(c => c.CustomerID == id).
Single());
       }
       return _custByID.Invoke(_context, ID);
     }
```

The method will use the compiled query. First it will compile the query on the fly, but only if necessary, which I determine by testing to see if the query has been instantiated yet. If you are compiling in your class constructor, you'll need to perform the same test to be sure you're only using resources to compile when necessary.

Because the delegate, _custByID, is static, it will remain in memory even when its containing class goes out of scope. Therefore, as long as the application process itself is in scope, the delegate will be available; it won't be null, and the compilation step will be skipped.

Precompiled Queries and Projections

There are some other speed bumps to be aware of that are much more discoverable. The first revolves around projections, but isn't specific to the problem of unwittingly recompiling your precompiled query. When you project columns in a query, rather than returning specific types, you'll always get an anonymous type as a result.

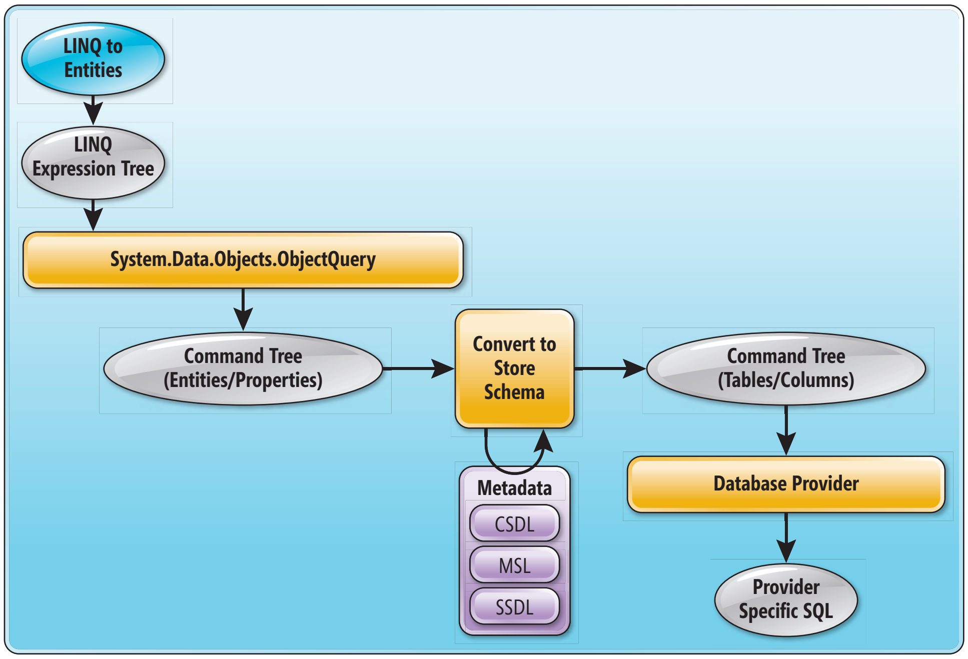


Figure 1 **Transforming a LINQ Query into a Relevant Store Query**

Figure 2 **Defining LoadOptions Along with a Delegate**

```
public Customer GetRandomCustomerWithOrders()
    {
      if (Load_Customer_Orders_Details == null)
      {
        Load_Customer_Orders_Details = new DataLoadOptions();
        Load_Customer_Orders_Details.LoadWith<Customer>(c => c.Orders);
        Load_Customer_Orders_Details.LoadWith<Order>(o => o.Details);
      }
      if (_context.LoadOptions == null)
      {
        _context.LoadOptions = Load_Customer_Orders_Details;
      }
      if (_CustWithOrders == null)
      {
        _CustWithOrders = CompiledQuery.Compile<DataClasses1DataContext,
Customer>
               (ctx => ctx.Customers.Where(c => c.Orders.Any()).
FirstOrDefault());
      }
      return _CustWithOrders.Invoke(_context);
    }
```

When defining the query, specifying its return type is impossible because there's no way to say "type of anonymous type." You'll have the same problem if you want to have the query inside of a method that returns the results, because you can't specify what will be returned by the method. Developers using LINQ hit this latter limitation frequently.

If you focus on the fact that an anonymous type is an on-the-fly type that isn't meant to be reused, these limitations, while frustrating, make sense. Anonymous types aren't meant to be passed around from method to method.

What you'll need to do for your precompiled query is define a type that matches the projection. Note that in Entity Framework you must use a class, not a struct, as LINQ to Entities won't allow you to project into a type that doesn't have a constructor. LINQ to SQL does allow structs to be the target of a projection. So, for Entity Framework you can only use a class, but for LINQ to SQL you can use either a class or a struct to avoid the limitations surrounding anonymous types.

Precompiled Queries and LINQ to SQL Prefetching

Another potential problem with precompiled queries involves prefetching, or *eager loading*, but the problem only arises with LINQ to SQL. In Entity Framework, you use the Include method to eager load, which results in a single query being executed in the database. Because Include can be part of a query, such as context.Customer.Include("Orders"), it's not an issue here. However, with LINQ to SQL, the eager loading is defined within the DataContext, not the query itself.

DataContext.LoadOptions has a LoadWith method that lets you specify what related data should get eager loaded along with a particular entity.

You can define LoadWith to load Orders with any Customer that is queried:

```
Context.LoadOptions.LoadWith<Customer>(c => c.Orders)
```

Then you can add a rule that says to load details with any orders that are loaded:

```
Context.LoadOptions.LoadWith<Customer>(c => c.Orders)
Context.LoadOptions.LoadWith<Order>(o =>o.Details)
```

You can define the LoadOptions directly against your DataContext instance or create a DataLoadOptions class, define LoadWith rules in this object, and then attach it to your context:

```
DataLoadOptions _myLoadOptions = new DataLoadOptions();
_myLoadOptions.LoadWith<Customer>(c => c.Orders)
Context.LoadOptions= myLoadOptions
```

There are caveats with the general use of LoadOptions and the DataLoadOptions class. For example, if you define and then attach DataLoadOptions, once a query has been executed against the DataContext, you can't attach a new set of DataLoadOptions. There's a lot more you can learn about the various load options and their caveats, but let's take a look at a basic pattern for applying some LoadOptions to a precompiled query.

The key to the pattern is that you can predefine the DataLoadOptions without associating them with a particular context.

In the class declarations where you declare the static Func variables to contain the precompiled queries, declare a new DataLoadOptions variable. It's critical to make this variable static so it, too, remains available along with the delegates:

```
static DataLoadOptions Load_Customer_Orders_Details = new
DataLoadOptions();
```

Then in the method that compiles and invokes the query, you can define the LoadOptions along with the delegate (see **Figure 2**). This method is valid in the .NET Framework 3.5 and .NET Framework 4.

Because the DataLoadOptions are static, they're defined only when necessary. Based on the logic of your class, the DataContext may or may not be new. If it's a context that's being reused, then it will have the previously assigned LoadOptions. Otherwise you'll need to assign them. Now you'll be able to invoke this query repeatedly and still get the benefit of LINQ to SQL's prefetching capabilities.

Keep Precompiling at the Top of Your Checklist

In the scope of LINQ query execution, query compilation is an expensive part of the process. Any time you're adding LINQ query logic to your LINQ to SQL- or Entity Framework-based applications, you should consider precompiling the queries and reusing them. But don't assume you're finished there. As you've seen, there are scenarios where you may not be benefiting from the precompiled query. Use some type of profiler, such as SQL Profiler or one of the profiling tools from Hibernating Rhinos, which include L2SProf (l2sprof.com/.com) and EFProf (efprof.com). You may need to leverage some of the patterns shown here to ensure that you're getting the edge that precompiled queries promise.

Danny Simmons, from the Microsoft Entity Framework team, explains how to control merge options when precompiling queries—and lists some additional gotchas to watch out for—in his blog post at blogs.msdn.com/dsimmons/archive/2010/01/12/ef-merge-options-and-compiled-queries.aspx. ■

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). You can follow her on Twitter at Twitter.com/julielermanvt.*

THANKS to the following technical expert for reviewing this article:
Danny Simmons

Enhancing Silverlight Video Experiences with Contextual Data

Jit Ghosh

There are two primary requirements for enabling a glitch-free viewing experience in Web-based, high-definition digital video delivery. First, the video provider needs to support high video delivery bit rates over the network. Second, the client computer needs to support continuous availability of processing capacity to decode the video at its fullest resolution.

The reality, however, is that network bandwidth for connected home computers can fluctuate significantly over time, and in certain parts of the world high bandwidth comes at a very high premium or is unavailable to many consumers. Along with that, the processing capacity of the client computer can vary, depending on CPU load at any given moment. As a result, consumers are susceptible to degradation in the quality of their viewing experience when a video stutters or freezes while the player is waiting to buffer enough data to show the next set of video frames, or waiting for the CPU cycles to decode those frames.

Adaptive streaming is a video delivery approach that addresses the problem of smooth content delivery and decoding. With adaptive streaming, video content is encoded at a range of bit rates and made available through a specialized streaming server. An adaptive streaming player constantly monitors various resource utilization metrics on the client computer and uses that information to compute the appropriate bit rate that the client can most efficiently decode and display at given the current resource constraints.

The player requests chunks of video encoded at that currently appropriate bit rate, and the streaming server responds with content from the video sources encoded at that bit rate. As a result, when resource conditions degrade, the player can continue displaying the video without any significant disruptions, with only a slight degradation in overall resolution, until an improvement or further degradation in conditions causes a different bit rate to be requested.

This kind of a continuous collaboration between the player and the server requires a special implementation of processing logic on both the streaming server and the client runtime implementing the player. Internet Information Server (IIS) Smooth Streaming is the server-side implementation of adaptive streaming over HTTP

This article is based on a prerelease version of the IIS Smooth Streaming Player Development Kit beta 2. All information is subject to change.

This article discusses:

- How Smooth Streaming works
- Streams, tracks, and manifests
- Clip scheduling
- Editing with composite manifests

Technologies discussed:

IIS Smooth Streaming, Silverlight

Code download available at:

code.msdn.microsoft.com/mag201003Silverlight from Microsoft. The client-side implementation is provided as an extension to Microsoft Silverlight.

The IIS Smooth Streaming Player Development Kit is a Silverlight library that lets applications consume content being streamed over IIS Smooth Streaming. It also provides a rich API that offers programmatic access to various aspects of the Smooth Streaming logic.

In this article I will walk you through the basics of Smooth Streaming, and explain how you can use the IIS Smooth Streaming Player Development Kit to build rich user experiences around video. Specifically, I will look at using the Player Development Kit to consume a stream, with a close examination of the client-side data model for streams and tracks. I will show you how to consume additional data streams, such as closed captions and animations, and merge external data streams with an existing presentation. You'll see how to schedule external clips such as advertisements within a presentation, handle variable playback rates and build composite manifests that lend to robust editing scenarios.

How Smooth Streaming Works

You can encode video for Smooth Streaming by using one of the supplied profiles in Expression Encoder 3.0. For one source video file, several files are created in the destination folder. **Figure 1** shows the files created for a source video named FighterPilot.wmv.

Each of the files with an .ismv extension contains the video encoded at a specific bit rate. For example, the FighterPilot_331.ismv contains the video encoded at a bit rate of 331 kbps, while FighterPilot_2056.ismv contains the video encoded at 2 mbps.

For each bit rate, the video content is broken into two-second fragments, and the .ismv files store these fragments in a file format called Protected Interoperable File Format (PIFF). Note that you can have additional audio tracks (or just audio in case the presentation is audio only) encoded in similar files that have an .isma extension.

The FighterPilot.ism file is a server manifest, which is structured in Synchronized Multimedia Integration Language (SMIL) format and contains a mapping of quality levels and bit rates to the .ismv and .isma files. This mapping in the server manifest is used by the server to access the right disk files to create the next fragment of content encoded at the right bit rate, before responding to a client side request. **Figure 2** shows an excerpt of a server manifest file.

The server manifest also contains a mapping to a client manifest file (identified by the extension .ismc), which in my example is FighterPilot.ismc. The client manifest contains all the information that the Silverlight client will need to access the various media and data streams, as well as metadata about those streams, such as quality levels, available bit rates, timing information, codec initialization data and so on. The client-side logic will use this metadata to sample and decode the fragments and request bit rate switches based on prevailing local conditions.

At run time, the presentation begins with the client requesting the client manifest from the server. Once the client receives the manifest, it checks to see what bit rates are available and requests fragments of content starting at the lowest available bit rate. In response, the server prepares and sends the fragments by reading the data from the disk file encoded at that bit rate (using the mapping in the server manifest). The content is then displayed on the client.

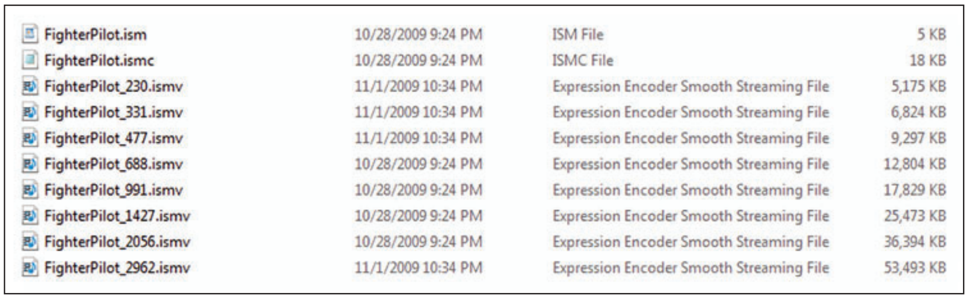
The client gradually requests higher bit rates as allowed by the resource-monitoring logic, and eventually reaches the highest allowable bit rate as determined by the prevailing resource conditions. This interchange continues until the client's monitoring logic senses a change in resource conditions resulting in a different lower desired bit rate. Subsequent client requests are for media encoded at the new bit rate, and the server again responds accordingly. This goes on until the presentation completes or is stopped.

Smooth Streaming with Silverlight

Getting video to play in Silverlight is a fairly uncomplicated effort. At a fundamental level, all you really need to do is add an instance of the MediaElement type to your XAML file, set the appropriate properties to control the MediaElement behavior, and make sure that the MediaElement.Source property points to a valid media source URI. For example, this XAML will play the FighterPilot.wmv video automatically as soon as the Silverlight page is launched, in a 640x360 rectangle:

```
<MediaElement AutoPlay="True"
  Source="http://localhost/Media/FighterPilot.wmv"
  Width="640" Height="360" />
```

The System.Windows.Controls.MediaElement type also exposes an API that allows you to control the behavior of the play



FighterPilot.ism	10/28/2009 9:24 PM	ISM File	5 KB
FighterPilot.ismc	10/28/2009 9:24 PM	ISMC File	18 KB
FighterPilot_230.ismv	11/1/2009 10:34 PM	Expression Encoder Smooth Streaming File	5,175 KB
FighterPilot_331.ismv	11/1/2009 10:34 PM	Expression Encoder Smooth Streaming File	6,824 KB
FighterPilot_477.ismv	11/1/2009 10:34 PM	Expression Encoder Smooth Streaming File	9,297 KB
FighterPilot_688.ismv	10/28/2009 9:24 PM	Expression Encoder Smooth Streaming File	12,804 KB
FighterPilot_991.ismv	10/28/2009 9:24 PM	Expression Encoder Smooth Streaming File	17,829 KB
FighterPilot_1427.ismv	10/28/2009 9:24 PM	Expression Encoder Smooth Streaming File	25,473 KB
FighterPilot_2056.ismv	10/28/2009 9:24 PM	Expression Encoder Smooth Streaming File	36,394 KB
FighterPilot_2962.ismv	11/1/2009 10:34 PM	Expression Encoder Smooth Streaming File	53,493 KB

Figure 1 **Files Generated for Smooth Streaming by Expression Encoder**

Getting the Smooth Streaming Environment

To try out the examples discussed in this article, you will need to prepare a Smooth Streaming environment on your development machines.

The server-side ingredient is straightforward: you'll need to download and install IIS Media Services 3.0 for IIS7 from iis.net/media using the Microsoft Web Platform installer.

You will need a copy of Microsoft Expression Encoder 3.0 to prepare video for Smooth Streaming. While there is a free evaluation version of Expression Encoder 3.0, that version does not include support for Smooth Streaming. You will need a licensed installation of Expression Encoder to create your own video.

For additional details on preparing your environment, please visit learn.iis.net/page.aspx/558/smooth-streaming-for-iis-70---getting-started.

Figure 2 **Sample Server Manifest**

```
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
  <head>
    <meta name="clientManifestRelativePath"
      content="FighterPilot.ismc" />
  </head>
  <body>
    <switch>
      <video src="FighterPilot_2962.ismv"
        systemBitrate="2962000">
        <param name="trackID"
          value="2" valuetype="data" />
      </video>
      <video src="FighterPilot_2056.ismv"
        systemBitrate="2056000">
        <param name="trackID"
          value="2" valuetype="data" />
      </video>
      ...
      <audio src="FighterPilot_2962.ismv"
        systemBitrate="64000">
        <param name="trackID"
          value="1" valuetype="data" />
      </audio>
    </switch>
  </body>
</smil>
```

experience in code and to build a player complete with standard controls like Play, Pause, Seek and so on. This approach works great with either progressively downloaded or HTTP streamed media as long as the container format and the encoding used is one that the Silverlight runtime has built-in support for.

What about file formats or codecs that are not supported out of the box by Silverlight? The MediaStreamSource (MSS) type enables an extensibility mechanism that allows you to take control of the media file parsing and decoding process by introducing your own custom parser and decoder into the Silverlight media pipeline. To do this, you need to implement a concrete type extending the abstract System.Windows.Media.MediaStream-Source, and then pass an instance of it to MediaElement using the MediaElement.SetSource method.

The MSS implementation will need to handle every aspect of the media consumption process short of the actual rendering—from receiving the media stream from a remote location, to parsing the container and associated metadata, to sampling individual audio and video samples and passing them to MediaElement for rendering.

Because the logic required to decode Smooth Streaming was not built into Silverlight, the first version of Smooth Streaming (part of IIS Media Services 2.0) was accompanied by a custom MSS implementation that handled all of the communication, parsing, and sampling logic, and also implemented the machine and network state-monitoring functionality.

For the most part, this approach worked well for Smooth Streaming, but there were a few shortcomings. The MSS is essentially a black box in that the only API it exposes directly is to facilitate interchange of raw audio and video samples between itself and a MediaElement. As a Silverlight developer, you do not have a direct way to interface with the MSS while in action. If the content being consumed had additional data like embedded text, animation or secondary camera angles, or if the streaming solution allowed for finer-grained control over the streams like variable playback rates, there was no way for you to programmatically access that additional data in a structured way because you were limited to interfacing with the fixed API set that MediaElement always exposes.

For Smooth Streaming, this poses a challenge. As you will see later in this article, the Smooth Streaming manifests and wire/file formats are pretty rich in terms of the additional content and metadata that can be carried, and with the MSS approach you could not get at that information. You need a Silverlight API that offers more control over and access to the Smooth Streaming solution.

IIS Smooth Streaming Player Development Kit

And that brings me to the IIS Smooth Streaming Player Development Kit. The Player Development Kit consists of a single assembly named Microsoft.Web.Media.SmoothStreaming.dll. At its heart is a type named Microsoft.Web.Media.SmoothStreaming.SmoothStreaming-MediaElement (SSME). Using SSME in your code is almost identical to the way you would use a regular MediaElement:

```
<UserControl x:Class="SSPlayer.Page"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:ss="clr-namespace:Microsoft.Web.Media.SmoothStreaming;assembly=M
icrosoft.Web.Media.SmoothStreaming">
  <Grid x:Name="LayoutRoot" Background="White">
    <ss:SmoothStreamingMediaElement AutoPlay="True"
      Width="640" Height="360"
      SmoothStreamingSource="http://localhost/SmoothStreaming/Media/
FighterPilot/FighterPilot.ism/manifest"/>
  </Grid>
</UserControl>
```

The SmoothStreamingSource property points SSME to a valid Smooth Streaming presentation. In general, the SSME API is a superset of the MediaElement API; this property is one of the few differences. SSME exposes the Source property just like Media-Element does, but SSME also exposes the SmoothStreamingSource property to attach to smooth streams. If you are authoring players that need the ability to consume both smooth streams and the other formats traditionally supported by MediaElement, you can safely use SSME, but you will likely need to author some code to set the right property to attach to the media source. Something like this:

```
private void SetMediaSource(string MediaSourceUri,
  SmoothStreamingMediaElement ssme) {

  if (MediaSourceUri.Contains(".ism"))
    ssme.SmoothStreamingSource = new Uri(MediaSourceUri);
  else
    ssme.Source = new Uri(MediaSourceUri);
}
```

The other major difference to keep in mind is that SSME does not expose a SetSource overload that accepts a MediaStream-Source type. If you need to use a custom MSS, you should do that through MediaElement.

Streams and Tracks

The Smooth Streaming client manifest contains rich metadata about the presentation and it can be useful to have programmatic access to that metadata inside your player application. SSME exposes parts of this metadata through a well-defined API in an arrangement of streams and tracks within each stream.

A stream represents the overall metadata for tracks of a specific type—video, audio, text, advertisements and so on. The stream also acts as a container for multiple tracks of the same underlying type. In a client manifest (see **Figure 3**), each StreamIndex entry represents a stream. There can be multiple streams in the presentation, as depicted by the multiple StreamIndex entries. There can also be multiple streams of the same type. In such cases, the stream name can be used to disambiguate among multiple occurrences of the same type.

The StreamInfo type represents the stream in your Silverlight code. Once SSME downloads the client manifest, it raises the SmoothStreamingMediaElement.ManifestReady event. At this point the SmoothStreamingMediaElement.AvailableStreams collection property contains a StreamInfo instance for each StreamIndex entry in the client manifest.

For a given video stream in the client manifest, the video track is broken into many fragments of two-second duration, and each c element in the manifest represents metadata for the fragment. In this case, the fragments in the track are contiguous and define the entire duration of the video track without any breaks in between—in other words, the stream is not sparse.

For a closed caption stream, the track includes only two fragments, each with individual timing information (the t attribute on the c element). Further, the ParentStreamIndex attribute is set to "video," parenting the closed caption stream with the video stream. This causes the closed caption stream to align with the timing information from the video stream—the closed caption stream starts and ends exactly with its parent video stream, and the first caption is displayed 10 seconds into the video stream while the second is displayed 15 seconds into the video. A stream in which the timeline is based on a parent stream and the fragments are non-contiguous is called a sparse stream.

A track is a timed sequence of fragments of content of a specific type—video, audio or text. Each track is represented using an instance of a TrackInfo type, and all the tracks in a stream are made available through the StreamInfo.AvailableTracks collection property.

Each track in a client manifest is uniquely identified via a QualityLevel. A QualityLevel is identified by the associated bit rate, and is exposed through the TrackInfo.Bitrate property. For example, a video stream in a client manifest may have several QualityLevels, each with a unique bit rate. Each represents a unique track of the same video content, encoded at the bit rate specified by the QualityLevel.

Custom Attributes and Manifest Output

Custom attributes are a way to add additional stream- or track-specific information to the manifest. Custom attributes are specified using a CustomAttribute element, which can contain multiple data elements expressed as key/value pairs. Each data element is expressed as an Attribute element, with Key and Value attributes specifying the data element key and the data element value. In cases where distinct quality levels do not apply, such as multiple tracks within a stream with the same track name and bit rate, a custom attribute can also be used to disambiguate tracks from each other. **Figure 4** shows an example of custom attribute usage.

For a given video stream in the client manifest, the video track is broken into many fragments.

Custom attributes added to a manifest do not affect any SSME behavior automatically. They are a way for the production workflow to introduce custom data into the manifest that your player code can receive and act upon. For example, in **Figure 4**, you may want to look for the AccessLevel custom attribute key in the video stream custom attributes collection, and expose that video stream only to paying subscribers as instructed in the attribute's value.

The StreamInfo.CustomAttributes collection property exposes a dictionary of string key/value pairs for all custom attributes applied at the stream level (as direct CustomAttribute child elements to the StreamIndex element). The TrackInfo.CustomAttributes property exposes the same for all custom attributes applied at the track level (as direct children to the QualityLevel element).

Figure 3 **Excerpt from a Client Manifest**

```
<SmoothStreamingMedia MajorVersion="2" MinorVersion="0"
  Duration="1456860000">
  <StreamIndex Type="video" Chunks="73" QualityLevels="8"
    MaxWidth="1280" MaxHeight="720"
    DisplayWidth="1280" DisplayHeight="720"
    Url="QualityLevels({bitrate})/Fragments(video={start time})">
    <QualityLevel Index="0" Bitrate="2962000" FourCC="WVC1"
      MaxWidth="1280" MaxHeight="720"
      CodecPrivateData="250000010FD37E27F1678A27F859E80490825A645A6440000
0010E5A67F840" />
    <QualityLevel Index="1" Bitrate="2056000" FourCC="WVC1"
      MaxWidth="992" MaxHeight="560"
      CodecPrivateData="250000010FD37E1EF1178A1EF845E8049081BEBE7D7CC0000
0010E5A67F840" />
    ...
    <c n="0" d="20020000" />
    <c n="1" d="20020000" />
    ...
    <c n="71" d="20020000" />
    <c n="72" d="15010001" />
  </StreamIndex>
  <StreamIndex Type="audio" Index="0" FourCC="WMAP"
    Chunks="73" QualityLevels="1"
    Url="QualityLevels({bitrate})/Fragments(audio={start time})">
    <QualityLevel Bitrate="64000" SamplingRate="44100" Channels="2"
      BitsPerSample="16" PacketSize="2973" AudioTag="354"
      CodecPrivateData="1000030000000000000000000000E00042C0" />
    <c n="0" d="21246187" />
    <c n="1" d="19620819" />
    ...
    <c n="71" d="19504762" />
    <c n="72" d="14900906" />
  </StreamIndex>
  <StreamIndex Type="text" Name="ClosedCaptions" Subtype="CAPT"
    TimeScale="10000000" ParentStreamIndex="video"
    ManifestOutput="TRUE" QualityLevels="1" Chunks="2"
    Url="QualityLevels({bitrate},{CustomAttributes})/
  Fragments(ClosedCaptions={start time})">
    <QualityLevel Index="0" Bitrate="1000"
      CodecPrivateData="" FourCC=""/>
    <c n="0" t="100000000">
      <f>...</f>
    </c>
    <c n="1" t="150000000">
      <f>...</f>
    </c>
  </StreamIndex>
  ...
</SmoothStreamingMedia>
```

Figure 4 **Using Custom Attributes in the Client Manifest**

```
<StreamIndex Type="video" Chunks="12" QualityLevels="2"
  MaxWidth="1280" MaxHeight="720"
  DisplayWidth="1280" DisplayHeight="720"
  Url="QualityLevels({bitrate})/Fragments(video={start time})">
  <CustomAttributes>
    <Attribute Key="CameraAngle" Value="RoofCam"/>
    <Attribute Key="AccessLevel" Value="PaidSubscription"/>
  </CustomAttributes>
  <QualityLevel Index="0" Bitrate="2962000" FourCC="WVC1"
    MaxWidth="1280" MaxHeight="720"
    CodecPrivateData="250000010FD37E27F1678A27F859E80490825A645A64400000
010E5A67F840">
    <CustomAttributes>
      <Attribute Name = "hardwareProfile" Value = "10000" />
    </CustomAttributes>
  </QualityLevel>
...
</StreamIndex>
```

When the ManifestOutput attribute on the stream (the Stream-Index element) is set to TRUE, the client manifest can actually contain the data representing each fragment for the tracks within the stream. **Figure 5** shows an example.

Note the nested content within the f elements—each represents caption item data to be displayed at the time specified by the containing chunk. The client manifest specification requires that the data be represented as a base64-encoded string version of the original data item.

The TrackInfo.TrackData collection property contains a list of TimelineEvent instances—one for each f element corresponding to the track. For each TimelineEvent entry, TimelineEvent.Event-Time represents the time point in the sequence and the Timeline-Event.EventData provides the base64-encoded text string. TrackInfo also supports Bitrate, CustomAttributes, Index, Name and ParentStream properties.

Selecting Streams and Tracks

There are many interesting ways you can use the streams and tracks metadata and API in your application code.

It may be useful to have the ability to select specific tracks within a stream and filter out the rest. A common scenario is a graded viewing experience based on a subscriber's level of access, where for a basic or free level you serve the low-resolution version of the content, and expose the high-definition version only to premium-level subscribers:

```
if (subscriber.AccessLevel != "Premium") {
  StreamInfo videoStream =
    ssme.GetStreamInfoForStreamType("video");
  List<TrackInfo> allowedTracks =
    videoStream.AvailableTracks.Where((ti) =>
    ti.Bitrate < 1000000).ToList();
  ssme.SelectTracksForStream(
    videoStream, allowedTracks, false);
}
```

GetStreamInfoForStreamType accepts a stream type literal and returns the matching StreamInfo instance. A LINQ query on Stream-Info.AvailableTracks retrieves a list of tracks that offer a bit rate of less than 1 mbps—in other words, a standard-definition video for non-premium subscribers. The SelectTracksForStream method can then be used to filter down the list of tracks in that stream to only the tracks you want to expose.

The last parameter to SelectTracksForStream, when set to true, indicates to SSME that any data stored in the look-ahead buffers should be cleaned out immediately. To get the current selected list of tracks at any time, you can use the StreamInfo.SelectedTracks property, while the StreamInfo.AvailableTracks property continues to expose all the available tracks.

Remember that Smooth Streaming allows multiple streams of the same type to coexist in the client manifest. In the current beta of the IIS Smooth Streaming Player Development Kit, the GetStreamInfoForStreamType method returns the first occurrence of a stream of the specified type in case there are multiple streams of that type, which may not be what you desire. However, there is nothing stopping you from bypassing this method and instead using a query on the AvailableStreams collection directly to get the right StreamInfo. The following snippet shows a LINQ query that gets a text stream named "ticker":

```
StreamInfo tickerStream =
  ssme.AvailableStreams.Where((stm) =>
  stm.Type == "text" &&
  stm.Name == "ticker").FirstOrDefault();
```

Using Text Streams

An audio/video presentation may need to display additional content that is timed along the primary video sequence at specific time points. Examples could be closed captions, advertise-ments, news alerts, overlay animations and so on. A text stream is a convenient place to expose such content.

One approach to include a text stream in your presentation would be to mux in the text tracks alongside the video tracks during the video encoding, so that the content chunks for the text track are delivered from the server, appropriately timed with the video.

Another option is to utilize the manifest output feature discussed earlier to author the text content into the client manifest itself. Let's take a closer look at this second approach.

To start, you need to prepare a client manifest with the text streams. In a production media workflow, there can be many different ways to inject such content into the manifest during or after encoding, and the data could be coming from several different sources, like ad-serving platforms and caption generators. But, for this example, I am going to use a simple XML data file as the data source, use some LINQ over XML queries to manufacture the text streams, and insert them into an existing client manifest.

Figure 5 **Manifest Output**

```
<StreamIndex Type="video" Chunks="12" QualityLevels="2"
  MaxWidth="1280" MaxHeight="720"
  DisplayWidth="1280" DisplayHeight="720"
  Url="QualityLevels({bitrate})/Fragments(video={start time})">
  <CustomAttributes>
    <Attribute Key="CameraAngle" Value="RoofCam"/>
    <Attribute Key="AccessLevel" Value="PaidSubscription"/>
  </CustomAttributes>
  <QualityLevel Index="0" Bitrate="2962000" FourCC="WVC1"
    MaxWidth="1280" MaxHeight="720"
    CodecPrivateData="250000010FD37E27F1678A27F859E80490825A645A64400000
010E5A67F840">
    <CustomAttributes>
      <Attribute Name = "hardwareProfile" Value = "10000" />
    </CustomAttributes>
  </QualityLevel>
...
</StreamIndex>
```

Figure 6 **Client Manifest Excerpt with Text Content Streams**

```
<SmoothStreamingMedia MajorVersion="2" MinorVersion="0"
  Duration="1456860000">
  <StreamIndex Type="video" Chunks="73" QualityLevels="8"
    MaxWidth="1280" MaxHeight="720"
    DisplayWidth="1280" DisplayHeight="720"
    Url="QualityLevels({bitrate})/Fragments(video={start time})">
    <QualityLevel Index="0" Bitrate="2962000" FourCC="WVC1"
      MaxWidth="1280" MaxHeight="720"
      CodecPrivateData="250000010FD37E27F1678A27F859E80490825A645A6440000
0010E5A67F840" />
    <QualityLevel Index="1" Bitrate="2056000" FourCC="WVC1"
      MaxWidth="992" MaxHeight="560"
      CodecPrivateData="250000010FD37E1EF1178A1EF845E8049081BEBE7D7CC0000
0010E5A67F840" />
    ...
    <c n="0" d="20020000" />
    <c n="1" d="20020000" />
    ...
    <c n="71" d="20020000" />
    <c n="72" d="15010001" />
  </StreamIndex>
  <StreamIndex Type="audio" Index="0" FourCC="WMAP"
    Chunks="73" QualityLevels="1"
    Url="QualityLevels({bitrate})/Fragments(audio={start time})">
    <QualityLevel Bitrate="64000" SamplingRate="44100" Channels="2"
      BitsPerSample="16" PacketSize="2973" AudioTag="354"
      CodecPrivateData="1000030000000000000000000000E00042C0" />
    <c n="0" d="21246187" />
    <c n="1" d="19620819" />
    ...
    <c n="71" d="19504762" />
    <c n="72" d="14900906" />
  </StreamIndex>
  <StreamIndex Type="text" Name="ClosedCaptions" Subtype="CAPT"
    TimeScale="10000000" ParentStreamIndex="video"
    ManifestOutput="TRUE" QualityLevels="1" Chunks="2"
    Url="QualityLevels({bitrate},{CustomAttributes})/
Fragments(ClosedCaptions={start time})">
    <QualityLevel Index="0" Bitrate="1000"
      CodecPrivateData="" FourCC=""/>
    <c n="0" t="100000000">
      <f>...</f>
    </c>
    <c n="1" t="150000000">
      <f>...</f>
    </c>
  </StreamIndex>
  ...
</SmoothStreamingMedia>
```

The structure of the data does not need to be complex. (You can find the full file in the code download for this article. I will show excerpts here for illustration.) The data file begins with a Tracks element, then contains two ContentTrack elements. Each ContentTrack entry will ultimately result in one distinct text stream in the client manifest. The first ContentTrack element is for the captions:

```
<ContentTrack Name="ClosedCaptions" Subtype="CAPT">
```

The second is for animations:

```
<ContentTrack Name="Animations" Subtype="DATA">
```

Smooth Streaming allows multiple streams of the same type to coexist in the client manifest.

Each ContentTrack contains multiple Event elements, with the time attributes specifying the time points on the video's timeline when these text events need to occur. The Event elements in turn contain the actual caption events defined in XML, or the XAML for the animation as CDATA sections:

```
<Event time="00:00:10">
  <![CDATA[<Caption Id="{DE90FACD-BC01-43f2-A4EC-6A01A49BAFBB}"
    Action="ADD">
    Test Caption 1
  </Caption>]]>
</Event>
<Event time="00:00:15">
  <![CDATA[<Caption Id="{DE90FACD-BC01-43f2-A4EC-6A01A49BAFBB}"
    Action="REMOVE"/>]]>
</Event>
```

Note that for each added closed caption event, there is a corresponding event that indicates the time point when the previously added caption needs to be removed. The Caption element contained within the CDATA section for a closed caption event defines an Action attribute with a value of Add or Remove to indicate appropriate action.

My LINQ over XML code transforms the XML data into appropriate entries for a client manifest, and inserts them into an existing client manifest file. You can find an example in the code download for this article, but note that the data format demonstrated is not a part of the Smooth Streaming Player Development Kit or the Smooth Streaming specification, and neither is it prescriptive in any way. You can define whatever data structure suits the needs of your application, as long as you can transform it into the appropriate format required by the Smooth Streaming client manifest specification, which includes encoding the text content in the CDATA sections to a base64 format.

Once the transformation is executed, the resulting client manifest file will contain the text streams as shown in **Figure 6**.

The video and the audio streams already existed in the client manifest shown in **Figure 6**, and I added the two text streams, named ClosedCaptions and Animations, respectively. Note that each stream uses the video stream as its parent and sets ManifestOutput to true. The former is because the text streams are sparse in nature and parenting them to the video stream ensures correct timing of each text content entry (the c elements) along the video stream's timeline. The latter is to ensure that the SSME reads the actual data (the base64-encoded strings within the f elements) from the manifest itself.

TimelineEvent and TimelineMarker

Now let's look at making use of the additional text content in SSME. SSME exposes the additional text streams as StreamInfo instances in the AvailableStreams property, with each StreamInfo containing the track data as a TrackInfo instance. The TrackInfo.TrackData collection property will contain as many instances of the TimelineEvent type as there are text events in each text track. The TimelineEvent.EventData property exposes a byte array representing the string content (decoded from its base64-encoded format), while the TimelineEvent.EventTime property exposes the time point where this event needs to occur.

When you start playing the presentation, as these events are reached, SSME raises the TimelineEventReached event. **Figure 7**

Figure 7 **Handling the TimelineEventReached Event**

```
ssme.TimelineEventReached +=
  new EventHandler<TimelineEventArgs>((s, e) => {
  //if closed caption event
  if (e.Track.ParentStream.Name == "ClosedCaptions" &&
    e.Track.ParentStream.Subtype == "CAPT") {

    //base64 decode the content and load the XML fragment
    XElement xElem = XElement.Parse(
      Encoding.UTF8.GetString(e.Event.EventData,
      0, e.Event.EventData.Length));

    //if we are adding a caption
    if (xElem.Attribute("Action") != null &&
      xElem.Attribute("Action").Value == "ADD") {

      //remove the text block if it exists
      UIElement captionTextBlock = MediaElementContainer.Children.
      Where((uie) => uie is FrameworkElement &&
        (uie as FrameworkElement).Name == (xElem.Attribute("Id").Value)).
        FirstOrDefault() as UIElement;
        if(captionTextBlock != null)
          MediaElementContainer.Children.Remove(captionTextBlock);

      //add a TextBlock
      MediaElementContainer.Children.Add(new TextBlock() {
        Name = xElem.Attribute("Id").Value,
        Text = xElem.Value,
        HorizontalAlignment = HorizontalAlignment.Center,
        VerticalAlignment = VerticalAlignment.Bottom,
        Margin = new Thickness(0, 0, 0, 20),
        Foreground = new SolidColorBrush(Colors.White),
        FontSize = 22
      });
    }
    //if we are removing a caption
    else if (xElem.Attribute("Action") != null &&
      xElem.Attribute("Action").Value == "REMOVE") {

      //remove the TextBlock
      MediaElementContainer.Children.Remove(
        MediaElementContainer.Children.Where(
        (uie) => uie is FrameworkElement &&
        (uie as FrameworkElement).Name ==
        (xElem.Attribute("Id").Value)).FirstOrDefault()
        as UIElement);
    }
  }

  //Logic for animation event
  ...
});
```

shows a sample of handling the closed caption and animation tracks that were added to the client manifest in **Figure 6**.

As each TimelineEvent is handled, you either insert a TextBlock into the UI to display a caption or load the animation XAML string and start the animation (see the downloadable code for details of the animation-handling logic).

Note that because the text content is base-64 encoded, it is decoded to its original state. Also note that the code checks the Action attribute on the Caption element to decide whether it is adding a caption to the UI or removing an existing caption. For animation events, you can rely on an animation's own completion handler to remove it from the UI.

Figure 8 shows a screenshot of a caption being displayed and an ellipse being animated overlaid on a playing video. While this approach works well, there is one issue you need to consider before using this technique. The current release of SSME handles TimlineEvents at two-second boundaries. To understand this better, let's say you had a closed caption timed at the 15.5-second time point along the video timeline. SSME would raise the TimelineEventReached event for this closed caption at the closest previous time point that is a multiple of 2—in other words, at approximately 14 seconds.

If your scenario demands greater accuracy and you can't position your content chunks close to two-second boundaries, using the TimelineEventReached to handle the content tracks may not be the right way. You can, however, use the TimelineMarker class (as used in the standard MediaElement type) to add markers to your timeline that can raise the MarkerReached event at any granularity you may need. The code download for this article includes the outline of an AddAndHandleMarkers method that adds TimelineMarkers for each content event and responds to them in the MarkerReached event handler.

Merging External Manifests

Earlier you saw an example of adding additional streams of content to a client manifest. That approach works well if you have access to the client manifest, but you may encounter situations where direct access to the client manifest to make the necessary additions is not possible. You may also encounter situations where the additional content streams are conditionally dependent on other factors (for example, closed captions in different languages for different locales). Adding the data for all possible conditions to the client manifest causes SSME to spend more time parsing and loading the manifest.

SSME solves this problem by allowing you to merge external manifest files at run time into the original client manifest, giving you the ability to bring in additional data streams and act upon the data as shown before, without having to modify the original client manifest.

Here is an example of manifest merging:

```
ssme.ManifestMerge += new
  SmoothStreamingMediaElement.ManifestMergeHandler((sender) => {
  object ParsedExternalManifest = null;
  //URI of the right external manifest based on current locale
  //for example expands to
  string UriString =
    string.Format(
    "http://localhost/SmoothStreaming/Media/FighterPilot/{0}/CC.xml",
    CultureInfo.CurrentCulture.Name);
  //parse the external manifest - timeout in 3 secs
  ssme.ParseExternalManifest(new Uri(UriString), 3000,
    out ParsedExternalManifest);
  //merge the external manifest
  ssme.MergeExternalManifest(ParsedExternalManifest);
});
```

This code snippet notes the prevailing locale and uses an appropriate external manifest file (named CC.xml stored in a folder named for the language identifier for the locale) that contains closed captions in the right language for that locale. The ParseExternalManifest method accepts a URI pointing to the location of the external manifest and returns the parsed manifest as an object through the third out parameter to the method. The second parameter to the method accepts a timeout value, allowing you to avoid blocking for too long on the network call.

The MergeExternalManifest method accepts the parsed manifest object returned from the previous call and does the actual merging. Following this, the streams and tracks from any merged external manifest are made available anywhere else in your player code as StreamInfo and TrackInfo instances, and can be acted upon as shown earlier.

It is important to note that the calls to ParseExternalManifest and MergeExternalManifest can only be made in the ManifestMerge event handler. Any calls to these methods outside the scope of this event handler raise an InvalidOperationException.

Keep in mind that external manifests need to have an extension that has an associated MIME type registered with the Web server from which they are available. Using a common extension such as .xml is a good idea because the content is XML is anyway. If the external manifest files are served from the same Web server that is acting as your Smooth Streaming server, you should refrain from using the .ismc extension because the IIS Media Services handler prevents .ismc files from being accessed directly, and ParseExternal-Manifest will fail to download the external manifest.

As far as the structure of an external manifest goes, it needs to be identical to a regular client manifest: a top-level SmoothStreaming-Media element, with appropriate StreamIndex child elements to represent your data.

Clip Scheduling

You may face the need to insert additional video clips into a presentation at specific time points. Advertisement videos, breaking news or filler clips in a presentation are just a few examples. The problem can be viewed in two parts. First, acquiring the necessary content data and determining where in the timeline to insert it. Second, actually scheduling and playing the clips. SSME incorporates functionality that makes both of these tasks fairly straightforward to implement.

You can continue to use the approach of a text stream inserted into the client manifest, as illustrated in the previous sections, to make the clip data available to your code. Here is a sample data source used for clip schedule information:

```
<ContentTrack Name="AdClips" Subtype="DATA">
  <Event time="00:00:04">
    <![CDATA[<Clip Id="{89F92331-8501-41ac-B78A-F83F6DD4CB40}"
    Uri="http://localhost/SmoothStreaming/Media/Robotica/Robotica_1080.
ism/manifest"
    ClickThruUri="http://msdn.microsoft.com/en-us/robotics/default.aspx"
    Duration="00:00:20" />]]>
  </Event>
  <Event time="00:00:10">
    <![CDATA[<Clip Id="{3E5169F0-A08A-4c31-BBAD-5ED51C2BAD21}"
    Uri="http://localhost/ProgDownload/Amazon_1080.wmv"
    ClickThruUri="http://en.wikipedia.org/wiki/Amazon_Rainforest"
    Duration="00:00:25"/>]]>
  </Event>
</ContentTrack>
```

For each clip to be scheduled there is a URI for the content, a URI for a Web page the user can navigate to as a click-through on the clip, and a playback duration for the clip. The time attribute on the Event element specifies where in the timeline the clip is scheduled.

You can transform this data and add the corresponding text stream into the client manifest, using the same approach of a LINQ to XML query as outlined in the previous section. As before, the text stream is exposed to the code as a StreamInfo instance. You can then use the clip scheduling API on the SSME to utilize this information to schedule these clips. **Figure 9** shows a method that schedules the clips based on this information.

The ScheduleClip method on SSME does the actual scheduling. For each clip you want to schedule, a new instance of the Clip-Information type is inserted into the schedule with the appropriate properties derived from the clip data.

Note that clips can be either Smooth Streaming sources or other sources as supported by the Silverlight MediaElement. It is important to set the ClipInformation.IsSmoothStreamingSource property correctly to make sure the right player component is used to play the clip.

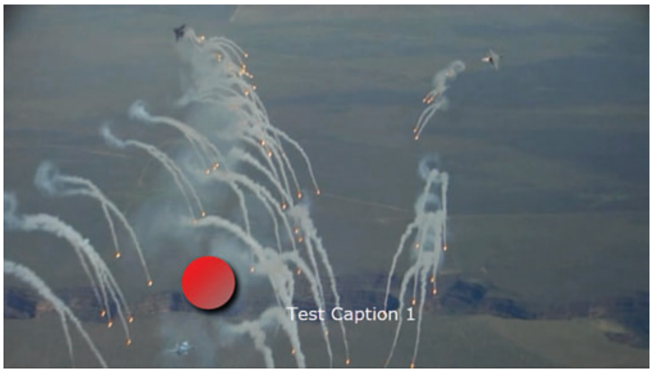


Figure 8 **Content Overlay Using Text Content Streams and TimelineEvents**

The second parameter to ScheduleClip is the time when you want the clip to play. The third parameter is used to indicate whether you want the timeline to stop progressing while the clip is playing. The last parameter is used to pass in any user data that will be made available with the various clip-related event handlers.

Sometimes clips need to be scheduled in a sequence where start-time information is applied only to the first clip in a sequence, and subsequent clips are chained so that all the scheduled clips play out in one continuous sequence. The ScheduleClip method facilitates this feature as well, as shown in **Figure 10**.

I only use an absolute time to schedule the first clip, when there is no ClipContext (in other words, the clipCtx variable is null). Each subsequent call to ScheduleClip returns a ClipContext instance that represents the scheduled state of the clip. The ScheduleClip method has an overload that accepts a ClipContext instance instead of a scheduled

Figure 9 **Schduling Clips**

```
private void ScheduleClips() {
  //get the clip data stream
  StreamInfo siAdClips = ssme.AvailableStreams.Where(
    si => si.Name == "AdClips").FirstOrDefault();

  //if we have tracks
  if (siAdClips != null && siAdClips.AvailableTracks.Count > 0) {

    //for each event in that track
    foreach (TimelineEvent te in
      siAdClips.AvailableTracks[0].TrackData) {

      //parse the inner XML fragment
      XElement xeClipData = XElement.Parse(
        Encoding.UTF8.GetString(te.EventData, 0,
        te.EventData.Length));

      //schedule the clip
      ssme.ScheduleClip(new ClipInformation {
        ClickThroughUrl = new Uri(
        xeClipData.Attribute("ClickThruUri").Value),
        ClipUrl = new Uri(xeClipData.Attribute("Uri").Value),
        IsSmoothStreamingSource =
        xeClipData.Attribute("Uri").Value.ToUpper().Contains("ism"),
        Duration = TimeSpan.Parse(xeClipData.Attribute("Duration").Value)
        },
        te.EventTime, true, //pause the timeline
        null);
    }
    //set the Clip MediaElement style
    ssme.ClipMediaElementStyle =
      this.Resources["ClipStyle"] as Style;
  }
}
```

Figure 10 **Using the ClipContext to Chain Scheduled Clips**

```
private void ScheduleClips() {
  StreamInfo siAdClips = ssme.AvailableStreams.Where(
  si => si.Name == "AdClips").FirstOrDefault();

  if (siAdClips != null && siAdClips.AvailableTracks.Count > 0) {
    ClipContext clipCtx = null;
    foreach (
      TimelineEvent te in siAdClips.AvailableTracks[0].TrackData) {
      XElement xeClipData =
        XElement.Parse(Encoding.UTF8.GetString(te.EventData, 0,
        te.EventData.Length));

      //if this is the first clip to be scheduled
      if (clipCtx == null) {
        clipCtx = ssme.ScheduleClip(new ClipInformation {
          ClickThroughUrl = new Uri(
          xeClipData.Attribute("ClickThruUri").Value),
          ClipUrl = new Uri(xeClipData.Attribute("Uri").Value),
          IsSmoothStreamingSource =
          xeClipData.Attribute("Uri").Value.ToUpper().Contains("ism"),
          Duration = TimeSpan.Parse(
          xeClipData.Attribute("Duration").Value)
        },
        te.EventTime, //pass in the start time for the clip
        true, null);
      }
      else { //subsequent clips
        clipCtx = ssme.ScheduleClip(new ClipInformation {
          ClickThroughUrl = new Uri(
          xeClipData.Attribute("ClickThruUri").Value),
          ClipUrl = new Uri(xeClipData.Attribute("Uri").Value),
          IsSmoothStreamingSource =
          xeClipData.Attribute("Uri").Value.ToUpper().Contains("ism"),
          Duration = TimeSpan.Parse(
          xeClipData.Attribute("Duration").Value)
        },
        clipCtx, //clip context for the previous clip to chain
        true, null);
      }
    }
    ssme.ClipMediaElementStyle =
      this.Resources["ClipStyle"] as Style;
  }
}
```

start time for a clip, and that schedules the clip to start right after the previously scheduled clip (represented by the passed-in ClipContext).

When the scheduled clips play, SSME hides the main video and introduces a MediaElement to play the scheduled clip. In the event that you want to customize this MediaElement, you can set the ClipMediaElementStyle property on SSME to a desired XAML style.

There are also several events of interest that are raised by SSME while a scheduled clip is playing. The ClipProgressUpdate event can be handled to track the progress of the clip. ClipPlaybackEventArgs.Progress is of the enumeration type ClipProgress, which represents the clip's progress in quartiles. The ClipProgressUpdate event is raised only at the start and end of the clip and at time points denoting 25 percent, 50 percent and 75 percent of the clip's duration. Note that the ClipContext.HasQuartileEvents boolean property indicates whether the quartile events will be raised for a clip. In certain cases, like when the duration of a clip is not known, quartile progress events may not be raised.

The ClipClickThrough event is raised when the viewer clicks on a clip while viewing it. If click-through destination was intended for this clip, ClipEventArgs.ClipContext.ClipInformation.ClickThroughUrl exposes it and you can use a technique of your choice (like interacting with the browser to open a pop-up window) to open up the Web resource targeted by the click-through URL.

You can also use the ClipError event and the ClipStateChanged event to handle any error conditions and state changes for the clip, respectively.

Playback Speed and Direction

SSME enables playing content at varying speeds and direction. The SmoothStreamingMediaElement.SupportedPlaybackRates property returns a list of supported playback speeds as double values, where 1.0 denotes the default playback speed. In the current public beta, this list contains the additional values of 0.5, 4.0, 8.0, -4.0 and -8.0. The positive values enable playback at half, 4x and 8x speeds, and the negative values enable reverse play (rewind) at 4x and 8x speeds.

The SmoothStreamingMediaElement.SetPlaybackRate method can be called to set the playback speed at any point during playback. SetPlaybackRate accepts the desired playback speed as its only parameter.

Note that controlling playback speed only works for Smooth Streaming content—so if you are using SSME to play content that is being progressively downloaded or streamed using some other technique, SetPlaybackRate will raise an exception.

Smooth Stream Edits Using Composite Manifests

Sometimes you may need to combine portions from multiple Smooth Streaming presentations into a single composite presentation. The most common scenario is using tools like rough-cut editors that allow users to specify mark-in and mark-out time points into a master source producing clips, and then having several such clips from potentially different master sources play in a linear fashion as a single presentation.

The composite manifest feature of SSME allows you to accomplish this by creating a separate manifest document that contains clip segments, where each clip segment defines a portion of a complete presentation bounded by the begin and end time points of the clip. The biggest benefit of using this approach is the ability to create different edits on existing presentations without the need to transcode the source material.

A composite manifest always ends with the extension .csm. To consume such a manifest you simply set the SmoothStreamingSource property to a valid URL pointing to a composite manifest file:

```
ssme.SmoothStreamingSource = new Uri("http://localhost/SmoothStreaming/
Media/MyCompositeSample.csm");
```

Figure 11 shows an excerpt from a composite manifest. (The entire file is included in the code download for this article.)

This manifest contains two Clip elements, each defining a clip (also called an edit) from an existing Smooth Streaming presentation. The URL attribute points to an existing Smooth Streaming presentation, and the ClipBegin and ClipEnd attributes contain the beginning and ending time values that provide the bounds to the clip. The Duration attribute on the top-level SmoothStreamingMedia element needs to be the exact sum of the durations of each clip in the manifest—you can sum the difference of the ClipEnd and ClipBegin values of each Clip entry to get the total manifest duration.

Each Clip element contains the video and the audio StreamIndex and their child QualityLevel entries, mirroring the client manifest (.ismc) files of the source presentations. The chunk metadata (c) entries for each StreamIndex entry, however, can be limited to only those chunks that are required to satisfy the ClipBegin and

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

Infragistics®
KILLER APPS. NO EXCUSES.

Figure 11 **Sample Composite Manifest**

```
<?xml version="1.0" encoding="utf-8"?>
<SmoothStreamingMedia MajorVersion="2" MinorVersion="0"
Duration="269000000">
<Clip Url="http://localhost/SmoothStreaming/Media/AmazingCaves/Amazing_
Caves_1080.ism/manifest"
  ClipBegin="81000000" ClipEnd="250000000">
<StreamIndex Type="video" Chunks="9" QualityLevels="3"
  MaxWidth="992" MaxHeight="560"
  DisplayWidth="992" DisplayHeight="560"
  Url="QualityLevels({bitrate})/Fragments(video={start time})">
  <QualityLevel Index="0" Bitrate="2056000" FourCC="WVC1"
    MaxWidth="992" MaxHeight="560"
    CodecPrivateData="250000010FD37E1EF1178A1EF845E8049081BEBE7D7CC00000
010E5A67F840"
  />
  <QualityLevel Index="1" Bitrate="1427000" FourCC="WVC1"
    MaxWidth="768" MaxHeight="432"
    CodecPrivateData="250000010FCB6C17F0D78A17F835E8049081AB8BD718400000
010E5A67F840"
  />
  <QualityLevel Index="2" Bitrate="991000" FourCC="WVC1"
    MaxWidth="592" MaxHeight="332"
    CodecPrivateData="250000010FCB5E1270A58A127829680490811E3DF8F8400000
010E5A67F840"
  />
  <c t="80130000" />
  <c t="100150000" />
  <c t="120170000" />
  <c t="140190000" />
  <c t="160210000" />
  <c t="180230000" />
  <c t="200250000" />
  <c t="220270000" />
  <c t="240290000" d="20020000" />
</StreamIndex>
<StreamIndex Type="audio" Index="0" FourCC="WMAP"
  Chunks="10" QualityLevels="1"
  Url="QualityLevels({bitrate})/Fragments(audio={start time})">
  <QualityLevel Bitrate="64000" SamplingRate="44100"
    Channels="2" BitsPerSample="16" PacketSize="2973"
    AudioTag="354" CodecPrivateData="1000030000000000000000000000E0004
2C0" />
  <c t="63506576" />
  <c t="81734240" />
  <c t="102632199" />
  <c t="121672562" />
  <c t="142106122" />
  <c t="162075283" />
  <c t="181580045" />
  <c t="202478004" />
  <c t="222447165" />
  <c t="241313378" d="20143311" />
</StreamIndex>
</Clip>
<Clip Url="http://localhost/SmoothStreaming/Media/CoralReef/Coral_Reef_
Adventure_1080.ism/manifest"
  ClipBegin="102000000" ClipEnd="202000000">
<StreamIndex Type="video" Chunks="6" QualityLevels="3"
  MaxWidth="992" MaxHeight="560"
  DisplayWidth="992" DisplayHeight="560"
  Url="QualityLevels({bitrate})/Fragments(video={start time})">
...
</Clip>
</SmoothStreamingMedia>
```

ClipEnd boundaries. In other words, the ClipBegin value needs to be greater than or equal to the start time(t attribute) value of the first c entry for the stream, and the ClipEnd value needs to be less than or equal to the sum of the start time and the duration(d attribute) values of the last c entry for that stream.

SSME enables playing content at varying speeds and direction.

Note that, in your client manifest, chunks may be defined in an indexed (n attribute) fashion with durations specified. However, for the composite manifest, the chunks need to be defined using their start times (which can be easily calculated by summing the durations of the preceding chunks). Also note that the Chunks attribute on each StreamIndex entry needs to reflect the number of chunks in the clip, but all the other attributes mirror the entries in the source client manifest.

Live Streams

SSME can play both on-demand and live streams. To play a live Smooth Streaming video stream using SSME, you can set the SmoothStreamingSource property on SSME to a live publishing point URL:

```
ssme.SmoothStreamingSource = "http://localhost/SmoothStreaming/Media/
FighterPilotLive.isml/manifest";
```

To know if SSME is playing a live stream, you can check the IsLive property, which is set to True if the content is a live source, and False otherwise.

Note that the setup and delivery of Smooth Streaming live video requires specialized infrastructure. A detailed discussion of setting up a live streaming server environment is beyond the scope of this article. You can refer to the articles at learn.iis.net/page.aspx/628/live-smooth-streaming/ for more details on how to set up IIS Media Services 3.0 for live streaming. The article at learn.iis.net/page.aspx/620/live-smooth-streaming-for-iis-70---getting-started/ will provide you with information on setting up a simulation of a live streaming environment for testing purposes.

Wrapping Up

IIS Smooth Streaming is a state-of-the-art adaptive streaming platform from Microsoft. As you've seen, the Smooth Streaming PDK (and in particular the SmoothStreamingMediaElement type) is an essential ingredient to authoring Silverlight clients that can consume both on-demand and live streams. The PDK offers extensive control on the client-side behavior of smooth streams, and allows you to write rich and immersive experiences that go beyond just audio/video streams, letting you easily combine data streams with your media in a meaningful way.

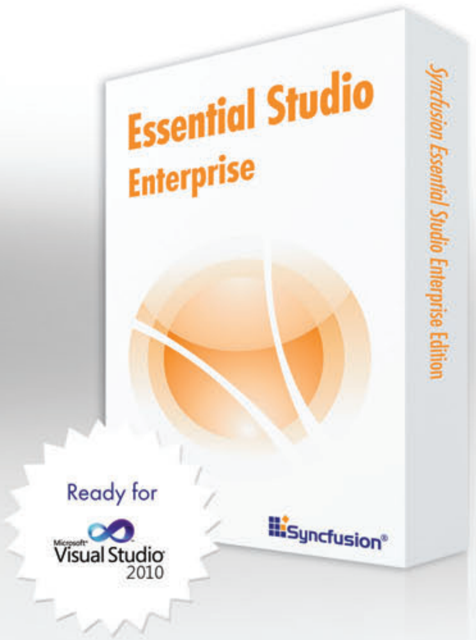
A detailed treatment of Smooth Streaming is beyond the scope of this article. You are encouraged to find more details at iis.net/media. For more guidance on media programming in Silverlight and the Silverlight MediaElement type, you can visit silverlight.net/getstarted. ■

JIT GHOSH *is an architect evangelist in the Developer Evangelism team at Microsoft, advising customers in the media industry on building cutting-edge digital media solutions. Ghosh co-authored the book "Silverlight Recipes" (APress, 2009). You can read his blog at blogs.msdn.com/jitghosh.*

THANKS to the following technical expert for reviewing this article: *Vishal Sood*

Deliver innovation with ease®

with Syncfusion's award-winning .NET components and controls


Essential Studio
Enterprise
Syncfusion Essential Studio Enterprise Edition
Syncfusion®
Ready for
Microsoft Visual Studio 2010

User interfaces, reporting, business intelligence.
Any .NET platform. Just one package.

ASP.NET ASP.NET MVC Windows Forms WPF Silverlight

Syncfusion®

www.syncfusion.com 1 888-9 DOTNET

RadControls *for* Silverlight

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

Pioneering Support for Microsoft Silverlight 4
Telerik is the first component vendor to provide native controls built on Silverlight 4. RadControls for Silverlight 4 fully match the feature set of their Silverlight 3 counterparts, and closely follow Microsoft's latest advancements in the Silverlight 4 framework, staying up to date with the latest beta releases and the fast-paced Microsoft release schedule for this technology.

Engineered for Great Performance
Telerik Silverlight controls are engineered for outstanding performance by utilizing various techniques that help reduce page loading time and speed up data operations such as streamlined themes and templates, virtualized scrolling, innovative LINQ-based data engine, built-in support for asynchronous databinding, RadCompression module and more.

Support for Visual Studio 2010 and Expression Blend
RadControls for Silverlight provide support for Visual Studio 2010 Beta 2, offering toolbox support, property browsing and WYSIWYG preview for all controls. Telerik is working closely with Microsoft to ensure best practices are followed and provide the most complete design experience, allowing for easy development in Visual Studio 2010 and styling in Expression Blend.

A Comprehensive Silverlight Toolset from the Masters of Web UI
An established leader in web interface technologies, Telerik offers a comprehensive suite of 40+ controls that bring style and interactivity to your line-of-business applications. Featuring a lightning fast DataGrid, rich data visualization controls, and a powerful Outlook-like Scheduling control, RadControls provides all the building blocks for developing next generation Rich Internet Applications (RIAs).

Code Re-Use with RadControls for WPF
RadControls for Silverlight and RadControls for WPF are derived from the same codebase and share the same API. They represent two almost mirror toolsets for building rich line-of-business web and desktop applications, allowing for substantial code and skills reuse between Silverlight and WPF development and shortening your learning curve.

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

UI COMPONENTS	DATA	PRODUCTIVITY TOOLS	AUTOMATED TESTING	TFS TOOLS	CMS
ASP.NET AJAX	OpenAccess ORM	JustCode	Web Testing Tools	Work Item Manager	Sitefinity
Silverlight	Reporting			Project Dashboard	
ASP.NET MVC					
WinForms					
WPF					

www.telerik.com

telerik

deliver more than expected

Exploring Multi-Touch Support in Silverlight

Charles Petzold

Whenever I visit the American Museum of Natural History in New York City, I always make a point to drop in on the Hall of Primates. With a large selection of skeletons and stuffed specimens, the hall presents an evolutionary panorama of the Primate order—animals ranging in size from tiny tree shrews, lemurs and marmosets, through chimpanzees, great apes and humans.

What leaps out from this exhibit is a striking characteristic common to all primates: the bone structure of the hand, including an opposable thumb. The same arrangement of joints and digits that allowed our ancestors and distant cousins to grasp and climb tree branches lets our species manipulate the world around us, and build things. Our hands may have their origins in the paws of tiny primates tens of millions of years ago, yet they are also a major factor in what makes us distinctly human.

This article discusses:

- Multi-touch events and classes
- Two-finger manipulation
- A TouchDial control
- Volume controls and music

Technologies discussed:

Silverlight

Code download available at:

code.msdn.microsoft.com/mag201003MultiTouch

Is it any wonder we reach out instinctively to point at or even touch objects on the computer screen?

In response to this human desire to bring our fingers into more intimate connection with the computer, our input devices have been evolving as well. The mouse is terrific for selecting and dragging, but hopeless for freeform sketching or handwriting. The tablet stylus lets us write but often feels awkward for stretching or moving. Touch screens are familiar from ATMs and museum kiosks, but are usually restricted to simple pointing and pressing.

I think the technology known as "multi-touch" represents a big leap forward. As the name implies, multi-touch goes beyond touch screens of the past to detect multiple fingers, and this makes a huge difference in the types of movement and gestures that can be conveyed through the screen. Multi-touch has evolved from the touch-oriented input devices of the past, but at the same time suggests an intrinsically different input paradigm.

Multi-touch has probably been most evident on television news broadcasts, with maps on large screen manipulated by the resident meteorologist or pundit. Microsoft has been exploring multi-touch in several ways—from the coffee-table-size Microsoft Surface computer to small devices like the Zune HD—and the technology is becoming fairly standard on smartphones as well.

While Microsoft Surface can respond to many simultaneous fingers (and even contains internal cameras to view objects placed on the glass), most other multi-touch devices are limited to a discrete number. Many respond to only two fingers—or touch points, as they're called. (I will be using finger and touch point fairly synonymously.) But synergy is at work here: On the computer screen, two fingers are more than twice as powerful as one.

The limitation of two touch points is characteristic of the multi-touch monitors that have become available recently for desktop PCs and laptops, as well as the customized Acer Aspire 1420P laptop distributed to attendees at the Microsoft Professional Developers Conference (PDC) last November—commonly referred to as the PDC laptop. The distribution of the PDC laptop provided a unique opportunity for thousands of developers to write multi-touch-aware applications.

The PDC laptop is the machine I used to explore multi-touch support under Silverlight 3.

Silverlight Events and Classes

Multi-touch support is becoming standard in the various Windows APIs and frameworks. Support is built into Windows 7 and the forthcoming Windows Presentation Foundation (WPF) 4. (The Microsoft Surface computer is based around WPF as well, but includes custom extensions for its very special capabilities.)

For this article I'd like to focus on the multi-touch support in Silverlight 3. The support is a little on the light side, but it's certainly adequate, and very useful for exploring basic multi-touch concepts.

If you publish a multi-touch Silverlight application to your Web site, who will be able to use it? The user will need a multi-touch monitor, of course, but will also need to be running the Silverlight application under an OS and browser that support multi-touch. For now, Internet Explorer 8 running under Windows 7 provides this support, and likely more OSes and browsers will support multi-touch in the future.

The Silverlight 3 support for multi-touch consists of five classes, one delegate, one enumeration and a single event. There is no way to determine if your Silverlight program is running on a multi-touch device or, if it is, how many touch points the device supports.

A Silverlight application that wants to respond to multi-touch must attach a handler to the static Touch.FrameReported event:

```
Touch.FrameReported += OnTouchFrameReported;
```

You can attach this event handler on machines that don't have multi-touch monitors and nothing bad will happen. The FrameReported event is the only public member of the static Touch class. The handler looks like this:

```
void OnTouchFrameReported(
  object sender, TouchFrameEventArgs args) {
  ...
}
```

You can install multiple Touch.FrameReported event handlers in your application, and all of them will report all touch events anywhere in the application.

TouchFrameEventArgs has one public property named TimeStamp that I haven't had occasion to use, and three essential public methods:

- TouchPoint GetPrimaryTouchPoint(UIElement relativeTo)
- TouchPointCollection GetTouchPoints(UIElement relativeTo)
- void SuspendMousePromotionUntilTouchUp()

The argument to GetPrimaryTouchPoint or GetTouchPoints is used solely for reporting position information in the TouchPoint object. You can use null for this argument; positioning information will then be relative to the upper-left corner of the entire Silverlight application.

Multi-touch supports multiple fingers touching the screen, and each finger touching the screen (up to the maximum number, which currently is usually two) is a touch point. The primary touch point refers to the finger that touches the screen when no other fingers are touching the screen and the mouse button is not pressed.

Touch a finger to the screen. That's the primary touch point. With the first finger still touching the screen, put a second finger on the screen. Obviously that second finger is not a primary touch point. But now, with the second finger still on the screen, lift the first finger and put it back on the screen. Is that a primary touch point? No, it's not. A primary touch point occurs only when no other fingers are touching the screen.

A primary touch point maps onto the touch point that will be promoted to the mouse. In real multi-touch applications, you should be careful not to rely on the primary touch point, because the user will typically not attach specific significance to the first touch.

Events are fired only for fingers actually touching the screen. There is no hover detection for fingers very close to the screen, but not touching.

By default, activity involving the primary touch point is promoted to various mouse events. This allows your existing applications to respond to touch without any special coding. Touching the screen becomes a MouseLeftButtonDown event, moving the finger while it's still touching the screen becomes a MouseMove, and lifting the finger is a MouseLeftButtonUp.

A multi-touch screen combines the functionality of a touch screen and a tablet.

The MouseEventArgs object that accompanies mouse messages includes a property named StylusDevice that helps differentiate mouse events from stylus and touch events. It is my experience with the PDC laptop that the DeviceType property equals TabletDeviceType.Mouse when the event comes from the mouse, and TabletDeviceType.Touch regardless of whether the screen is touched with the finger or the stylus.

Only the primary touch point is promoted to mouse events, and—as the name of the third method of TouchFrameEventArgs suggests—you can inhibit that promotion. More on this shortly.

A particular Touch.FrameReported event might be fired based on one touch point or multiple touch points. The TouchPointCollection returned from the GetTouchPoints method contains all the touch points associated with a particular event. The TouchPoint returned from GetPrimaryTouchPoint is always a primary touch point. If there is no primary touch point associated with the particular event, GetPrimaryTouchPoint will return null.

Even if the TouchPoint returned from GetPrimaryTouchPoint is non-null, it will not be the same object as one of the TouchPoint objects returned from GetTouchPoints, although all the properties will be the same if the argument passed to the methods is the same.

Figure 1 **Code for MultiTouchEvents**

```
using System;
using System.Collections.Generic;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;

namespace MultiTouchEvents {
  public partial class MainPage : UserControl {
    Dictionary<int, TextBox> touchDict =
      new Dictionary<int, TextBox>();

    public MainPage() {
      InitializeComponent();
      Touch.FrameReported += OnTouchFrameReported;
    }

    void OnTouchFrameReported(
      object sender, TouchFrameEventArgs args) {

      TouchPoint primaryTouchPoint =
        args.GetPrimaryTouchPoint(null);

      // Inhibit mouse promotion
      if (primaryTouchPoint != null &&
        primaryTouchPoint.Action == TouchAction.Down)
        args.SuspendMousePromotionUntilTouchUp();

      TouchPointCollection touchPoints =
        args.GetTouchPoints(null);

      foreach (TouchPoint touchPoint in touchPoints) {
        TextBox txtbox = null;
        int id = touchPoint.TouchDevice.Id;
        // Limit touch points to 2
        if (touchDict.Count == 2 &&
          !touchDict.ContainsKey(id)) continue;

        switch (touchPoint.Action) {
          case TouchAction.Down:
            txtbox = touchDict.ContainsValue(txtbox1) ?
              txtbox2 : txtbox1;
            touchDict.Add(id, txtbox);
            break;

          case TouchAction.Move:
            txtbox = touchDict[id];
            break;

          case TouchAction.Up:
            txtbox = touchDict[id];
            touchDict.Remove(id);
            break;
        }

        txtbox.Text += String.Format("{0} {1} {2}\r\n",
          touchPoint.TouchDevice.Id, touchPoint.Action,
          touchPoint.Position);
        txtbox.Select(txtbox.Text.Length, 0);
      }
    }
  }
}
```

The TouchPoint class defines the following four get-only properties, all backed by dependency properties:

- Action of type TouchAction, an enumeration with members Down, Move and Up.
- Position of type Point relative to the element passed as an argument to the GetPrimaryTouchPoint or GetTouchPoints method (or relative to the upper-left corner of the application for an argument of null).
- Size of type Size. Size information is not available on the PDC laptop so I didn't work with this property at all.
- TouchDevice of type TouchDevice.

You can call the SuspendMousePromotionUntilTouchUp method from the event handler only when GetPrimaryTouchPoint returns a non-null object and the Action property equals TouchAction.Down.

Multi-touch support is becoming standard in the various Windows APIs and frameworks.

The TouchDevice object has two get-only properties also backed by dependency properties:

- DirectlyOver of type UIElement—the topmost element underneath the finger.
- Id of type int.

DirectlyOver need not be a child of the element passed to GetPrimaryTouchPoint or GetTouchPoints. This property can be null if the finger is within the Silverlight application (as defined by the dimensions of the Silverlight plug-in object), but not within an area encompassed by a hit-testable control. (Panels with a null background brush are not hit-testable.)

The ID property is crucial for distinguishing among multiple fingers. A particular series of events associated with a particular finger will always begin with an Action of Down when the finger touches the screen, followed by Move events, finishing with an Up event. All these events will be associated with the same ID. (But don't assume that a primary touch point will have an ID value of 0 or 1.)

Most non-trivial multi-touch code will make use of a Dictionary collection where the ID property of TouchDevice is the dictionary key. This is how you will store information for a particular finger across events.

Examining the Events

When exploring a new input device, it's always helpful to write a little application that logs the events on the screen so you can get an idea of what they're like. Among the downloadable code accompanying this article is a project named MultiTouchEvents. This project consists of two side-by-side TextBox controls showing the multi-touch events for two fingers. If you have a multi-touch monitor you can run this program at charlespetzold.com/silverlight/MultiTouchEvents.

The XAML file consists of just a two-column Grid containing two TextBox controls named txtbox1 and txtbox2. The code file is shown in **Figure 1**.

Notice the dictionary definition at the top of the class. The dictionary keeps track of which TextBox is associated with the two touch point IDs.

The OnTouchFrameReported handler begins by inhibiting all mouse promotion. That's the only reason for the call to GetPrimary-

TouchPoint, and very often the only reason you'll be calling this method in a real program.

A foreach loop enumerates through the TouchPoint members of the TouchPointCollection returned from GetTouchPoints. Because the program contains only two TextBox controls and is only equipped to handle two touch points, it ignores any touch point where the dictionary already has two and the ID is not in that dictionary. (Just as you want your multi-touch-aware Silverlight program to handle multiple fingers, you don't want it to crash if it encounters too many fingers!) The ID is added to the dictionary on a Down event, and removed from the dictionary on an Up event.

You'll notice that at times the TextBox controls get bogged down with too much text, and you'll need to select all the text and delete it (Ctrl-A, Ctrl-X) to get the program running smoothly again.

What you'll notice from this program is that multi-touch input is captured on an application level. For example, if you press your finger on the application, and then move it off the application, the application will continue to receive Move events and eventually an Up event when you lift your finger up. In fact, once an application is getting some multi-touch input, multi-touch input to other applications is inhibited, and the mouse cursor disappears.

Touch a finger to the screen. That's the primary touch point.

This application-centric capturing of multi-touch input allows the MultiTouchEvents application to be very sure of itself. For example, on Move and Down events the program simply assumes that the ID will be in the dictionary. In a real application, you might want more bullet-proofing just in case something odd happens, but you'll always get the Down event.

Two-Finger Manipulation

One of the standard multi-touch scenarios is a photo gallery that lets you move, resize and rotate photos with your fingers. I decided to try something similar—just to give myself a little familiarity with the principles involved—but simpler as well. My version of the program has only a single item to manipulate, a text string of the word "TOUCH." You can run the TwoFingerManipulation program on my Web site at charlespetzold.com/silverlight/TwoFingerManipulation.

When you code an application for multi-touch, you'll probably always inhibit mouse promotion for multi-touch-aware controls. But to make your program usable without a multi-touch monitor, you'll also add specific mouse processing.

If you have only a mouse or a single finger, you can still move the string within the TwoFingerManipulation program, but you can change only its position—the graphical operation known as translation. With two fingers on a multi-touch screen, you can also scale the object and rotate it.

When I sat down with a pad and pen to figure out the algorithm I'd need for this scaling and rotation, it soon became obvious that there was no unique solution!

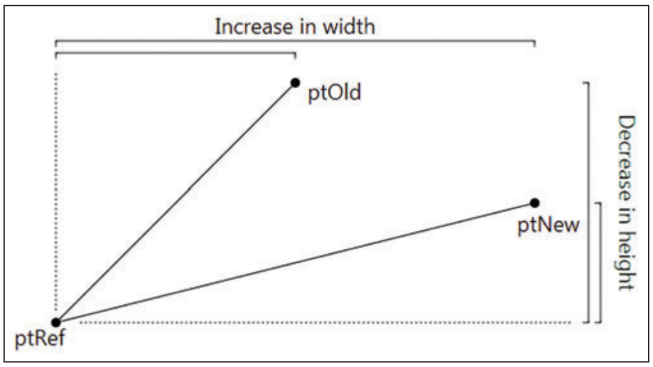


Figure 2 **Two-Finger Movement Converted to Scaling Factors**

Suppose one finger remains fixed at the point ptRef. (All points here are relative to a display surface underneath the object being manipulated.) The other finger moves from the point ptOld to ptNew. As shown in **Figure 2**, you can use these three points solely to calculate horizontal and vertical scaling factors for the object. For example, horizontal scaling is the increase in the distance of ptOld.X and ptNew.X from ptRef.X, or:

```
scaleX = (ptNew.X - ptRef.X) / (ptOld.X - ptRef.X)
```

Vertical scaling is similar. For the example in **Figure 2**, the horizontal scaling factor is 2 and the vertical scaling factor is ½.

This is certainly the easier way to code it. Yet, the program seems to function more naturally if the two fingers rotate the object as well. This is shown in **Figure 3**.

First, the angles of the two vectors—from ptRef to ptOld, and from ptRef to ptNew—are calculated. (The Math.Atan2 method is ideal for this job.) Then ptOld is rotated relative to ptRef by the difference in these angles. This rotated ptOld is then used with ptRef and ptNew to calculate scaling factors. These scaling factors are much less because a rotation component has been removed.

The actual algorithm (implemented in the ComputeMoveMatrix method in the C# file) turned out to be fairly easy. However, the program also required a bunch of transform support code to compensate for the deficiencies of the Silverlight transform classes, which have no public Value property or matrix multiplication as in the WPF.

In the actual program, both fingers can be moving at the same time, and handling the interaction between the two fingers is simpler than it initially seems. Each moving finger is handled

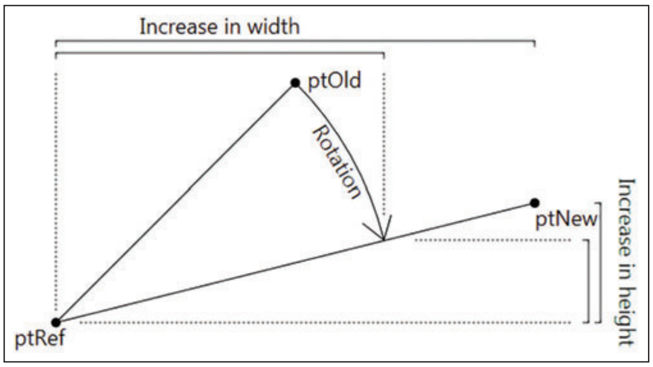


Figure 3 **Two-Finger Movement Converted to Rotation and Scaling**

Figure 4 **The XAML File for the SimpleTouchDialTemplate Project**

```
<UserControl x:Class="SimpleTouchDialTemplate.MainPage"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:multitouch="clr-namespace:Petzold.MultiTouch;assembly=Petzold.
MultiTouch">
  <UserControl.Resources>
    <Style x:Key="touchDialStyle"
      TargetType="multitouch:TouchDial">
      <Setter Property="Maximum" Value="180" />
      <Setter Property="Minimum" Value="-180" />
      <Setter Property="Width" Value="200" />
      <Setter Property="Height" Value="200" />
      <Setter Property="HorizontalAlignment" Value="Center" />
      <Setter Property="VerticalAlignment" Value="Center" />
      <Setter Property="Template">
        <Setter.Value>
          <ControlTemplate TargetType="multitouch:TouchDial">
            <Grid>
              <Ellipse Fill="{TemplateBinding Background}" />
              <Grid RenderTransform="{TemplateBinding RotateTransform}">
                <Rectangle Width="20" Margin="10"
                  Fill="{TemplateBinding Foreground}" />
              </Grid>
            </Grid>
          </ControlTemplate>
        </Setter.Value>
      </Setter>
    </Style>
  </UserControl.Resources>

  <Grid x:Name="LayoutRoot">
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="*" />
      <ColumnDefinition Width="*" />
    </Grid.ColumnDefinitions>

    <multitouch:TouchDial Grid.Column="0"
      Background="Blue" Foreground="Pink"
      Style="{StaticResource touchDialStyle}" />

    <multitouch:TouchDial Grid.Column="1"
      Background="Red" Foreground="Aqua"
      Style="{StaticResource touchDialStyle}" />
  </Grid>
</UserControl>
```

independently using the other finger as the reference point. Despite the increased complexity of the calculation, the result seems more natural and I think there's a simple explanation: In real life, it is very common to rotate objects with your fingers, but very unusual to scale them.

Rotation is so common in the real world that it might make sense to implement it when an object is manipulated by only one finger or the mouse. This is demonstrated in the alternative AltFinger-Manipulation program (runnable at charlespetzold.com/silverlight/AltFingerManipulation). For two fingers, the program works the same as TwoFingerManipulation. For one finger, it calculates a rotation relative to the center of the object, and then uses any excess movement away from the center for translation.

Wrapping the Event with More Events

Generally I like to work with classes that Microsoft thoughtfully provides in a framework rather than wrapping them in my own code. But I had in mind some multi-touch applications I thought would benefit from a more sophisticated event interface.

I wanted first a more modular system. I wanted to mix custom controls that would handle their own touch input with existing Silverlight controls that simply let touch input be converted to mouse input. I also wanted to implement capture. Although the Silverlight application itself captures the multi-touch device, I wanted individual controls to independently capture a particular touch point.

I also wanted Enter and Leave events. In a sense, these events are the opposite of a capture paradigm. To understand the difference, imagine an on-screen piano keyboard where each key is an instance of the PianoKey control. At first you might think of these keys like mouse-triggered buttons. On a mouse down event the piano key turns a note on, and on a mouse up event it turns the note off.

But that's not what you want for piano keys. You want the ability to run your finger up and down the keyboard to make glissando effects. The keys really shouldn't even bother with Down and Up events. They're really only concerned with Enter and Leave events.

Multi-touch input is captured on an application level.

WPF 4 and Microsoft Surface already have routed touch events, and they're likely coming to Silverlight in the future. But I met my current needs with a class I called TouchManager, implemented in the Petzold.MultiTouch library project in the TouchDialDemos solution. A large portion of TouchManager consists of static methods, fields, and a static handler for the Touch.FrameReported event that allows it to manage touch events throughout an application.

A class that wants to register with TouchManager creates an instance like so:

```
TouchManager touchManager = new TouchManager(element);
```

The constructor argument is of type UIElement, and usually it will be the element creating the object:

```
TouchManager touchManager = new TouchManager(this);
```

By registering with TouchManager, the class indicates that it is interested in all multi-touch events where the DirectlyOver property of TouchDevice is a child of the element passed to the TouchManager constructor, and that these multi-touch events should not be promoted to mouse events. There is no way to unregister an element.

After creating a new instance of TouchManager, a class can install handlers for events named TouchDown, TouchMove, TouchUp, TouchEnter, TouchLeave and LostTouchCapture:

```
touchManager.TouchEnter += OnTouchEnter;
```

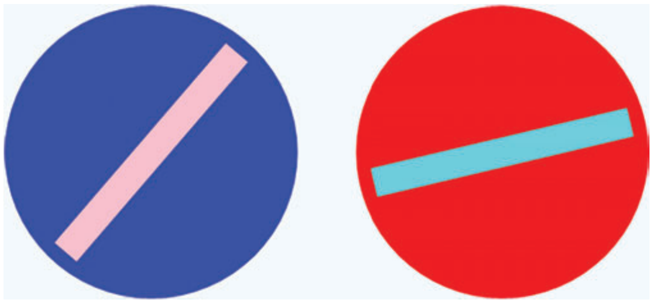


Figure 5 **The SimpleTouchDialTemplate Program**

Figure 6 **The OffCenterTouchDial XAML File**

```
<UserControl x:Class="OffCenterTouchDial.MainPage"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:multitouch="clr-namespace:Petzold.MultiTouch;assembly=Petzold.
MultiTouch">
  <Grid x:Name="LayoutRoot">
    <multitouch:TouchDial Width="300" Height="200"
      HorizontalAlignment="Center" VerticalAlignment="Center"
      Minimum="-20" Maximum="20"
      InputCenterX="35" InputCenterY="100"
      RenderCenterX="15" RenderCenterY="15">
      <multitouch:TouchDial.Template>
        <ControlTemplate TargetType="multitouch:TouchDial">
          <Grid Background="Pink">
            <Rectangle Height="30" Width="260"
              RadiusX="15" RadiusY="15" Fill="Lime"
              RenderTransform="{TemplateBinding RotateTransform}" />
            <Ellipse Width="10" Height="10"
              Fill="Black" HorizontalAlignment="Left"
              Margin="30" />
          </Grid>
        </ControlTemplate>
      </multitouch:TouchDial.Template>
    </multitouch:TouchDial>
  </Grid>
</UserControl>
```

All handlers are defined in accordance with the EventHandler<TouchEventArgs> delegate:

```
void OnTouchEnter(
  object sender, TouchEventArgs args) {
  ...
}
```

TouchEventArgs defines four properties:

- Source of type UIElement, which is the element originally passed to the TouchManager constructor.
- Position of type Point. This position is relative to Source.
- DirectlyOver of type UIElement, simply copied from the TouchDevice object.
- Id of type int, also just copied from the TouchDevice object.

Only while processing the TouchDown event is a class allowed to call the Capture method with the touch point ID associated with that event:

```
touchManager.Capture(id);
```

All further touch input for that ID goes to the element associated with this TouchManager instance until the TouchUp event or an explicit call to ReleaseTouchCapture. In either case, TouchManager fires the LostTouchCapture event.

The events are generally in the order: TouchEnter, TouchDown, TouchMove, TouchUp, TouchLeave and LostTouchCapture (if applicable). Of course there can be multiple TouchMove events between TouchDown and TouchUp. When a touch point is not captured, there can be multiple events in the order TouchLeave, TouchEnter and TouchMove as the touch point leaves one registered element and enters another.

The TouchDial Control

Changes in user-input paradigms often require you to question old assumptions about the proper design of controls and other input mechanisms. For example, few GUI controls are as solidly entrenched as the scrollbar and slider. You use these controls to navigate large documents or images, but also as tiny volume controls on media players.

As I considered making an on-screen volume control that would respond to touch, I wondered if the old approach was really the correct one. In the real world, sliders are sometimes used as volume controls, but generally restricted to professional mixing panels or graphic equalizers. Most volume controls in the real world are dials. Might a dial be a better solution for a touch-enabled volume control?

I won't pretend I have the definitive answer, but I'll show you how to build one.

The TouchDial control is included in the Petzold.MultiTouch library in the TouchDialDemos solution (see the code download for details). TouchDial derives from RangeBase so it can take advantage of the Minimum, Maximum and Value properties—including the coercion logic to keep Value within the Minimum and Maximum range—and the ValueChanged event. But in TouchDial, the Minimum, Maximum and Value properties are all angles in units of degrees.

TouchDial responds to both mouse and touch, and it uses the TouchManager class to capture a touch point. With either the mouse or touch input, TouchDial changes the Value property during a Move event based on the new location and previous location of the mouse or finger relative to a center point. The action is quite similar to **Figure 3** except that no scaling is involved. The Move event uses the Math.Atan2 method to convert Cartesian coordinates to angles, and then adds the difference in the two angles to Value.

In real life, it is very common to rotate objects with your fingers, but very unusual to scale them.

TouchDial does not include a default template, and hence has no default visual appearance. When using TouchDial, your job is to supply a template, but it can be as simple as a few elements. Obviously something on this template should probably rotate in accordance with changes in the Value property. For convenience, TouchDial supplies a get-only RotateTransform property where the Angle property is equal to the Value property of the RangeBase, and the CenterX and CenterY properties reflect the center of the control.

Figure 4 shows a XAML file with two TouchDial controls that reference a style and template defined as a resource.

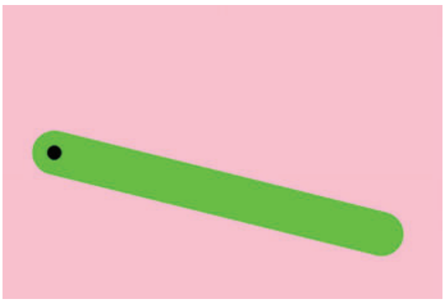


Figure 7 **The OffCenterTouchDial Program**

Notice that the style sets the Maximum property to 180 and the Minimum to -180 to allow the bar to be rotated 180 degrees to the left and right. (Oddly, the program did not function correctly when I switched the order of those two properties in the style definition.) The dial consists simply of a bar made from a Rectangle element within an Ellipse. The Bar is inside a single-cell Grid, which has its RenderTransform bound to the RotateTransform property calculated by TouchDial.

Figure 8 **The C# File for VolumeControl**

```
using System;
using System.Windows;
using System.Windows.Controls;

namespace Petzold.MultiTouch {
  public partial class VolumeControl : UserControl {
    public static readonly DependencyProperty VolumeProperty =
      DependencyProperty.Register("Volume",
      typeof(double),
      typeof(VolumeControl),
      new PropertyMetadata(0.0, OnVolumeChanged));

    public event DependencyPropertyChangedEventHandler VolumeChanged;

    public VolumeControl() {
      DataContext = this;
      InitializeComponent();
    }

    public double Volume {
      set { SetValue(VolumeProperty, value); }
      get { return (double)GetValue(VolumeProperty); }
    }

    void OnTouchDialValueChanged(object sender,
      RoutedPropertyChangedEventArgs<double> args) {

      Volume = 96 * (args.NewValue + 150) / 300;
    }

    static void OnVolumeChanged(DependencyObject obj,
      DependencyPropertyChangedEventArgs args) {

      (obj as VolumeControl).OnVolumeChanged(args);
    }

    protected virtual void OnVolumeChanged(
      DependencyPropertyChangedEventArgs args) {

      touchDial.Value = 300 * Volume / 96 - 150;

      if (VolumeChanged != null)
        VolumeChanged(this, args);
    }
  }
}
```

The SimpleTouchDialTemplate program is shown running in **Figure 5**.

You can try it out (along with the next two programs I'll be discussing here) at charlespetzold.com/silverlight/TouchDialDemos.

Turning the bar within the circle is a little awkward with the mouse and feels much more natural with a finger. Notice that you can turn the bar when you press the left mouse button (or put your finger on the screen) anywhere within the circle. While turning the bar, you can move the mouse or finger away because both are captured.

If you want to restrict the user from turning the bar unless the mouse or finger is pressed directly over the bar, you can set the IsHitTestVisible property of the Ellipse to False.

My first version of the TouchDial control didn't include the RotateTransform property. It made more sense to me that the template could include an explicit RotateTransform where the Angle property was the target of a TemplateBinding to the Value property of the control. However, in Silverlight 3, bindings don't work on properties of classes not derived from FrameworkElement, so the Angle property of RotateTransform can't be a binding target (this is fixed in Silverlight 4).

Rotation is always in reference to a center point, and that little fact complicates the TouchDial control. TouchDial uses a center point in two ways: to calculate the angles shown in **Figure 3**, and also to set the CenterX and CenterY properties of the RotateTransform it creates. By default, TouchDial calculates both centers as half the ActualWidth and ActualHeight properties, which is the center of the control, but there are very many cases where that's not quite what you want.

For example, in the template in **Figure 4**, suppose you want to bind the RenderTransform property of the Rectangle to the RotateTransform property of TouchDial. It won't work correctly because TouchDial is setting the CenterX and CenterY properties of RotateTransform to 100, but the center of the Rectangle relative to itself is actually the point (10, 90). To let you override these defaults that TouchDial calculates from the size of the control, the control defines RenderCenterX and RenderCenterY properties. In the SimpleTouchDialTemplate property you can set these properties in the style like so:

```
<Setter Property="RenderCenterX" Value="10" />
<Setter Property="RenderCenterY" Value="90" />
```

Changes in user-input paradigms often require you to question old assumptions.

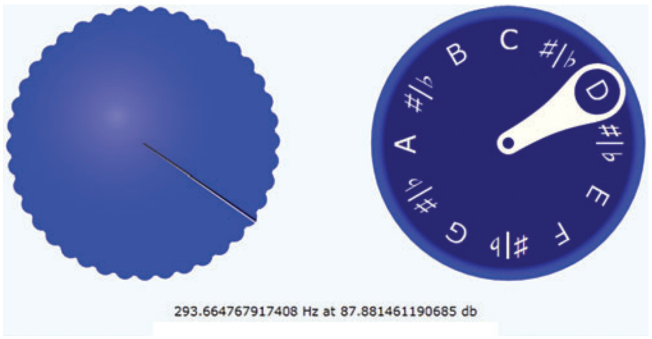
Or, you can set these properties to zero and set the Render-TransformOrigin of the Rectangle element to indicate the center relative to itself:

```
RenderTransformOrigin="0.5 0.5"
```

You might also want to use TouchDial in cases where the point used to reference the mouse or finger movement isn't in the center of the control. In that case, you can set the InputCenterX and InputCenterY properties to override the defaults.

Figure 6 shows the OffCenterTouchDial project XAML file. This file contains a single TouchDial control where properties are set on the control itself, and the Template property is set to a Control template containing a single-cell Grid with a Rectangle and Ellipse. The Ellipse is a tiny symbolic pivot point for the Rectangle, which you can swivel up and down by 20 degrees, as shown in **Figure 7**.

The InputCenterX and InputCenterY properties are always relative to the entire control, so they indicate the location of the center of



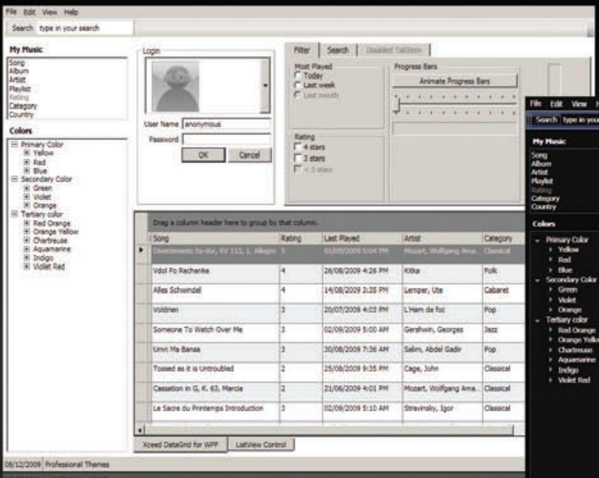
293.664767917408 Hz at 87.881461190685 db

Figure 9 **The PitchPipe Program**

XCEED
Professional Themes
for WPF

Beautifully themed WPF apps in under 2 minutes.

4:58 PM - Ewww!



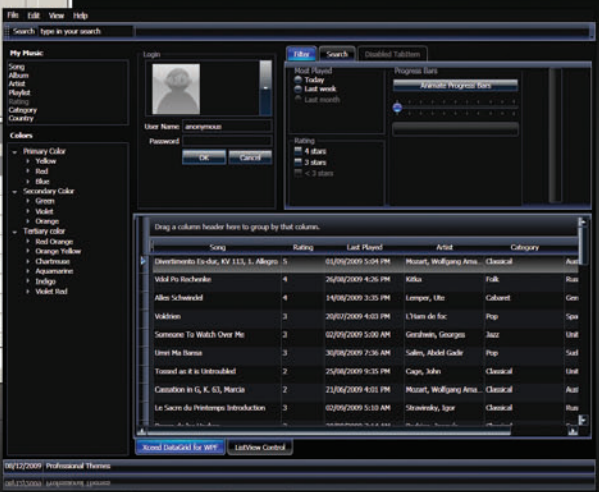
Search: type in your search

User Name: anonymous
Password:

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

08/12/2009 | Professional Themes

5:00 PM - Awesome!



Search: type in your search

User Name: anonymous
Password:

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

08/12/2009 | Professional Themes

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

Try it live on xceed.com

XCEED
MULTI-TALENTED COMPONENTS

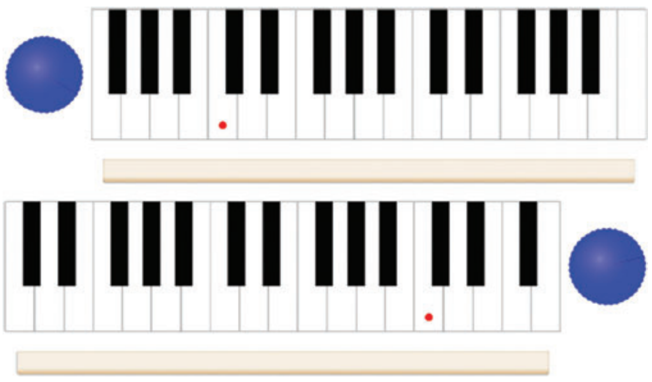


Figure 10 **The Piano Program**

the Ellipse element within the pink Grid. The RenderCenterX and RenderCenterY properties are always relative to the part of the control to which the RotateTransform property is applied.

Volume Controls and Pitch Pipes

The two previous examples demonstrate how you can give a visual appearance to TouchDial by either setting the Template property explicitly in markup or, if you need to share templates among multiple controls, by referencing a ControlTemplate defined as a resource.

You can also derive a new class from TouchDial and use the XAML file solely for setting a template. This is the case with the RidgedTouchDial in the Petzold.MultiTouch library. RidgedTouchDial is the same as TouchDial except it has a specific size and visual appearance (which you'll see shortly).

It is also possible to use TouchDial (or a derived class like RidgedTouchDial) within a class derived from UserControl. The advantage of this approach is that you can hide all the properties defined by RangeBase, including Minimum, Maximum and Value, and replace them with a new property.

This is the case with VolumeControl. VolumeControl derives from RidgedTouchDial for its visual appearance and defines a new property named Volume. The Volume property is backed by a dependency property and any changes to that property fire a VolumeChanged event.

The XAML file for VolumeControl simply references the RidgedTouchDial control and sets several properties, including Minimum, Maximum and Value:

```
<src:RidgedTouchDial
  Name="touchDial"
  Background="{Binding Background}"
  Maximum="150"
  Minimum="-150"
  Value="-150"
  ValueChanged="OnTouchDialValueChanged" />
```

Thus, the TouchDial can rotate through 300 degrees from the minimum position to the maximum position. **Figure 8** shows the VolumeControl.xaml.cs. The control translates the 300 degree range of the dial into the logarithmic decibel scale 0 through 96.

Why 96? Well, although the decibel scale is based on decimal numbers—whenever the amplitude of a signal increases by a multiplicative factor of 10, the loudness increases linearly by 20 decibels—it is also true that 10 to the 3rd power is approximately 2 to the 10th power. This means that when the amplitude doubles, the loudness increases by 6 decibels. Therefore, if you represent amplitude with a 16-bit value—which is the case with CD and PC sound—you get a range of 16 bits times 6 decibels per bit, or 96 decibels.

The PitchPipeControl class also derives from UserControl and defines a new property named Frequency. The XAML file includes a TouchDial control as well as a bunch of TextBlocks to show the 12 notes of the octave. PitchPipeControl also makes use of another property of TouchDial I haven't discussed yet: If you set SnapIncrement to a non-zero value in angles, the motion of the dial will not be smooth, but will jump between increments. Because PitchPipeControl can be set for the 12 notes of the octave, the SnapIncrement is set to 30 degrees.

Figure 9 shows the PitchPipe program that combines VolumeControl and PitchPipeControl. You can run PitchPipe at charlespetzold.com/silverlight/TouchDialDemos.

The Bonus Program

Earlier in this article I mentioned a control named PianoKey in the context of an example. PianoKey is an actual control, and it is one of several controls in the Piano program you can run at charlespetzold.com/silverlight/Piano. The program is intended to be displayed with your browser maximized. (Or press F11 to make Internet Explorer go into Full Screen mode and get even more room.) A very tiny rendition is shown in **Figure 10**. The keyboard is divided into overlapping treble and bass parts. The red dots indicate Middle C.

The Piano program demonstrates three different ways to use multi-touch. I suspect that there are many, many more.

It is for this program that I wrote TouchManager because the Piano program uses touch in three different ways. I've already discussed the blue VolumeControl, which captures the touch point on a TouchDown event and releases capture on TouchUp. The PianoKey controls that make up the keyboards also use TouchManager, but these controls only listen to the TouchEnter and TouchLeave events. You can indeed run your fingers across the keys for glissando effects. The brown rectangles that function as sustain pedals are ordinary Silverlight ToggleButton controls. These are not specifically touch-enabled; instead touch points are converted to mouse events.

The Piano program demonstrates three different ways to use multi-touch. I suspect that there are many, many more. ■

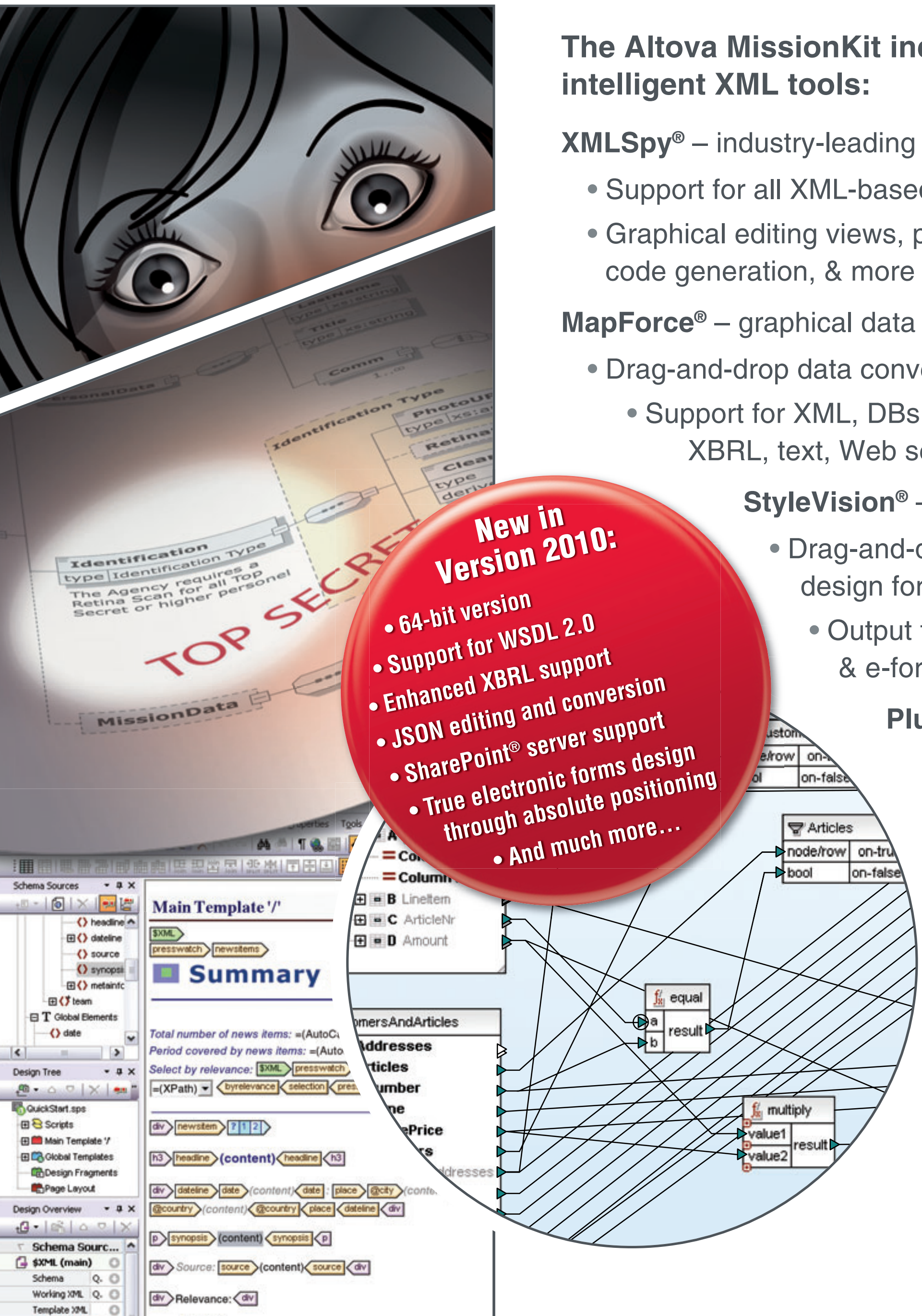
CHARLES PETZOLD *is a longtime contributing editor to MSDN Magazine. His most recent book is "The Annotated Turing: A Guided Tour Through Alan Turing's Historic Paper on Computability and the Turing Machine" (Wiley, 2008). Petzold blogs on his Web site charlespetzold.com.*

THANKS to the following technical experts for reviewing this article: *Robert Levy and Anson Tsao*

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

ALTOVA® missionkit®

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



New in Version 2010:

- 64-bit version
- Support for WSDL 2.0
- Enhanced XBRL support
- JSON editing and conversion
- SharePoint® server support
- True electronic forms design through absolute positioning
- And much more…

The Altova MissionKit includes multiple intelligent XML tools:

XMLSpy® – industry-leading XML editor

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

MapForce® – graphical data mapping tool

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

StyleVision® – visual stylesheet & e-Forms designer

- Drag-and-drop stylesheet and electronic forms design for XML, XBRL & databases
  - Output to HTML, PDF, RTF, Word 2007+, & e-forms

Plus up to five additional tools…

Download a 30 day free trial!

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

Performance Tuning with the Concurrency Visualizer in Visual Studio 2010

Hazim Shafi

Multicore processors have become widely available, and single-threaded performance in new processors is likely to remain relatively flat. That means added pressure on software developers to improve application performance by taking better advantage of parallelism.

Parallel programming is challenging for many reasons, but in this article I'd like to focus on the performance aspects of parallel applications. Multithreaded applications are not only prone to common sources of inefficiency in sequential implementations, such as inefficient algorithms, poor cache behavior, and excessive I/O, but they can also suffer from parallel performance bugs. Parallel performance and scalability may be limited by load imbalance, excessive synchronization overhead, inadvertent serialization, or thread migration.

Disclaimer: This article discusses a prerelease version of Visual Studio 2010. All information is subject to change.

This article discusses:

- CPU, Thread and Core views
- Blocking and inter-thread dependencies
- Creating reports
- Support for PPL, TPL and PLINQ

Technologies discussed:

Visual Studio 2010

Understanding such performance bottlenecks used to require significant instrumentation and analysis by expert developers. Even for those elite programmers, performance tuning was a tedious and time-consuming process.

This is about to change for the better. Visual Studio 2010 includes a new profiling tool—the Concurrency Visualizer—that should significantly reduce the burden of parallel performance analysis. Moreover, the Concurrency Visualizer can help developers analyze their sequential applications to discover opportunities for parallelism. In this article, I present an overview of the features of the Concurrency Visualizer in Visual Studio 2010, along with some practical usage guidance.

CPU Utilization

The Concurrency Visualizer comprises several visualization and reporting tools. There are three main views: CPU Utilization, Threads, and Cores.

The CPU Utilization view, shown in **Figure 1**, is intended to be the starting point in Concurrency Visualizer. The x axis shows the time elapsed from the start of the trace until the end of application activity (or the end of the trace, whichever is earlier). The y axis shows the number of logical processor cores in the system.

Before I describe the purpose of the view, it is important that you understand what a logical core is. A single CPU chip today can include multiple microprocessor circuits, referred to as physical cores. Each physical core may be capable of running multiple application threads simultaneously. This is often referred to as simultaneous multithreading (SMT); Intel calls it Hyper-Threading Technology. Each hardware-supported thread on an SMT-capable core presents itself as a logical core to the operating system.

If you collect a trace on a quad-core system that does not support SMT, the y axis would show four logical cores. If each core in your quad-core system is capable of running two SMT threads, then the y axis would show eight logical cores. The point here is that the number of logical cores is a reflection of the number of threads that can simultaneously execute in your system, not the number of physical cores.

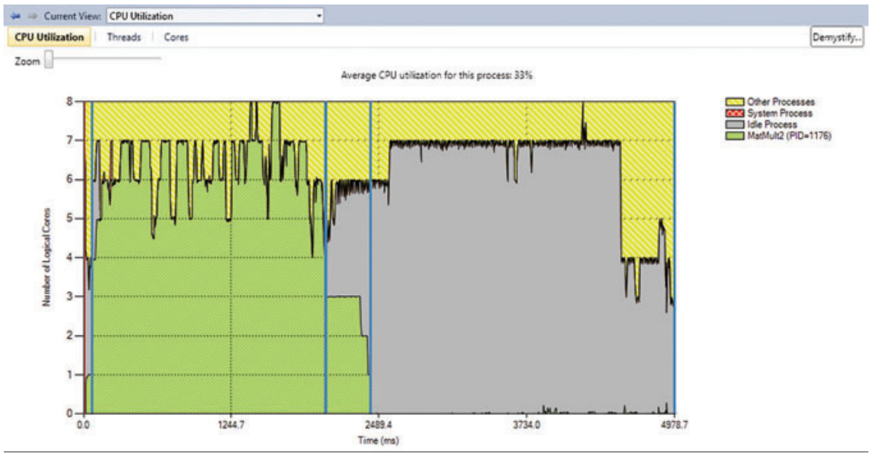


Figure 1 **CPU Utilization View**

Now, let's get back to the view. There are four areas shown in the graph, as described in the legend. The green area depicts the average number of logical cores that the application being analyzed is using at any given time during the profiling run. The rest of the logical cores are either idle (shown in gray), used by the System process (shown in red), or used by other processes running on the system (shown in yellow).

The blue vertical bars in this view correspond to an optional mechanism that allows users to instrument their code in order to correlate the visualizations in the tool with application constructs. I will explain how this can be done later in this article.

The Zoom slider control at the top left allows you to zoom in on the view to get more details, and the graph control supports a horizontal scrollbar when zoomed. You can also zoom by clicking the left mouse button and dragging in the area graph itself.

This view has three main purposes. First, if you are interested in parallelizing an application, you can look for areas of execution that either exhibit significant serial CPU-bound work, shown as lengthy green regions at the single-core level on the y axis, or regions where there isn't much CPU utilization, where the green doesn't show or is considerably less than 1 on average. Both of these circumstances might indicate an opportunity for parallelization. CPU-intensive work can be sped up by leveraging parallelism, and areas of unexpected low CPU utilization might imply blocking (perhaps due to I/O) where parallelism may be used by overlapping other useful work with such delays.

Second, if you are trying to tune your parallel application, this view allows you to confirm the degree of parallelism that exists when your application is actually running. Hints of many common parallel performance bugs are usually apparent just by examining this graph. For example, you can observe load imbalances as stair-step patterns in the graph, or contention for synchronization objects as serial execution when parallelism is expected.

Third, since your application lives in a system that may be executing many other applications that are competing for its resources, it is important to understand whether your application's performance is affected by other apps. When interference is unexpected, it is usually a good idea to reduce it by disabling applications or services to improve the fidelity of data, because performance is usually an iterative process. Sometimes, interference is caused by other processes with which your application collaborates to deliver an experience. Either way, you will be able to use this view to discover whether such interference exists, and then identify the actual processes involved by using the Threads view, which I will discuss later.

Another feature that can help reduce interference is using the profiler command-line tools to collect traces rather than doing so from within the Visual Studio IDE.

Focus your attention on some window of execution that piques your interest, zoom in on it, and then switch to the Threads view for further analysis. You can always come back to this view to find the next region of interest and repeat the process.

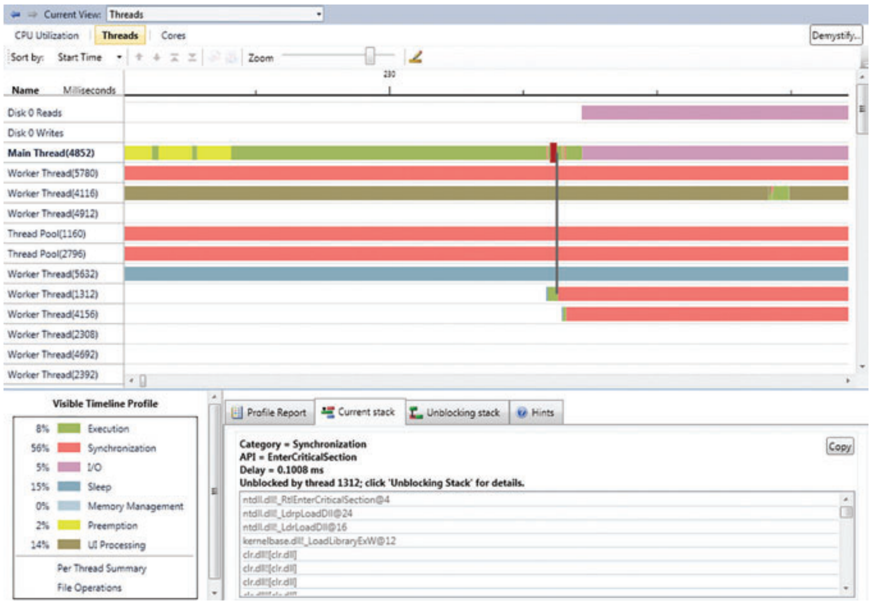
Threads

The Threads view, shown in **Figure 2**, contains the bulk of the detailed analysis features and reports in the Concurrency Visualizer. This is where you'll find information that explains behavior you identified in the CPU Utilization or Cores views. It is also where you can find data to link behavior to application source code when possible. There are three main components of this view: the timeline, the active legend and the reporting/details tab control.

Like the CPU Utilization view, the Threads view shows time on the x axis. (When switching between views in Concurrency Visualizer, the range of time shown on the x axis is preserved.) However, the Threads view y axis contains two types of horizontal channels.

The top channels are usually dedicated to physical disks on your system if they had activity in your application's profile. There are two channels per disk, one each for reads and writes. These channels show disk accesses that are made by your application threads or by the System process threads. (It shows the System accesses because they can sometimes reflect work being done on behalf of your process, such as paging.) Every read or write is drawn as a rectangle. The length of the rectangle depicts the latency of the access, including queuing delays; therefore, multiple rectangles may overlap.

To determine which files were accessed at a given point in time, select a rectangle by clicking the left mouse button. When you do that, the reports view below will switch to the Current Stack tab, which is the standard location for displaying data interactively with the timeline. Its contents will list the names of files that were



Visible Timeline Profile

8%	🟩	Execution
56%	🟥	Synchronization
5%	🟪	I/O
15%	🟦	Sleep
0%	▨	Memory Management
2%	🟨	Preemption
14%	🟫	UI Processing

Per Thread Summary
File Operations

```
Category = Synchronization
API = EnterCriticalSection
Delay = 0.1008 ms
Unblocked by thread 1312; click 'Unblocking Stack' for details.
ntdll.dll!_RtlEnterCriticalSection@4
ntdll.dll!_LdrpLoadDll@24
ntdll.dll!_LdrLoadDll@16
kernelbase.dll!_LoadLibraryExW@12
clr.dll![clr.dll]
clr.dll![clr.dll]
clr.dll![clr.dll]
```

Figure 2 **Threads View**

either read or written, depending on the disk channel selected. I will return to I/O analysis later.

One thing to be aware of is that not all file read and write operations performed by the application may be visible when they are expected to occur. This is because the operating system's file system uses buffering, allowing some disk I/O operations to complete without accessing the physical disk device.

The remaining channels in the timeline list all the threads that existed in your application during the profile collection period. For each thread, if the tool detected any activity during the profiler run, it will display the state of the thread throughout the trace until it is terminated.

If a thread is running, which is depicted by the green Execution category, the Concurrency Visualizer shows you what the thread was doing by leveraging sample profile information. There are two ways to get at this data. One is by clicking on a green segment, in which case you'll see the nearest (within +/- 1 ms) profile sample call stack in the Current Stack tab window.

You can also generate a sample profile report for the visible time range to understand where most of the work was spent. If you click on the Execution label in the active legend, the report will show up in the Profile Report tab. The profile report has two features that may be used to reduce complexity. One is a noise reduction feature that, by default, removes call stacks responsible for 2 percent or less of the profile samples. This threshold can be changed by the user. Another feature, called Just My Code, can be used to reduce the number of stack frames due to system DLLs in the report, if that's desirable. I'll cover the reports in more detail later.

Before going on, I'd like to point out a few more features for managing complexity in the reports and views. You will often encounter application scenarios consisting of many threads, some of which may not be doing anything useful in a given profiler run. Besides filtering reports based on the time range, the Concurrency Visualizer also allows you to filter by the threads that are active. If you're interested in threads that do work, you can use the Sort By option to sort the threads by the percentage of time that they are in the Execution state. You can then select the group of threads that are not doing much useful work and hide them from the display either by right-clicking and selecting the Hide option from the context menu or by clicking the Hide button in the toolbar at the top of the view. You can sort by all thread state categories and can hide/unhide as you see fit.

The effect of hiding threads is that their contributions to all the reports will be removed, in addition to hiding their channels from the timeline. All statistics and reports in the tool are kept up-to-date dynamically as filtering is performed on threads and time range.

Blocking Categories

Threads can block for many reasons. The Threads view attempts to identify the reason why a thread blocked by mapping each instance to a set of blocking categories. I say attempts because this categorization can sometimes be inaccurate, as I'll explain in a moment, so it should be viewed as a rough guide. That said, the Threads view shows all thread delays and accurately depicts execution periods. You should focus your attention on categories responsible for significant delays in the view based on your understanding of the application's behavior.

In addition, the Threads view provides the call stack at which the thread stopped execution in the Current Stack tab if you click on a blocking event. By clicking on a stack frame in the Current Stack window, the user will be taken to the source code file (when available) and line number where the next function is called. This is an important productivity feature of the tool.

Let's take a look at the various blocking categories:

Synchronization Almost all blocking operations can be attributed to an underlying synchronization mechanism in Windows. The Concurrency Visualizer attempts to map blocking events due to synchronization APIs such as EnterCriticalSection and WaitForSingleObject to this category, but sometimes other operations that result in synchronization internally may be mapped to this category—even though they might make more sense elsewhere. Therefore, this is often a very important blocking category to analyze during performance tuning, not just because synchronization overheads are important but also because it can reflect other important reasons for execution delays.

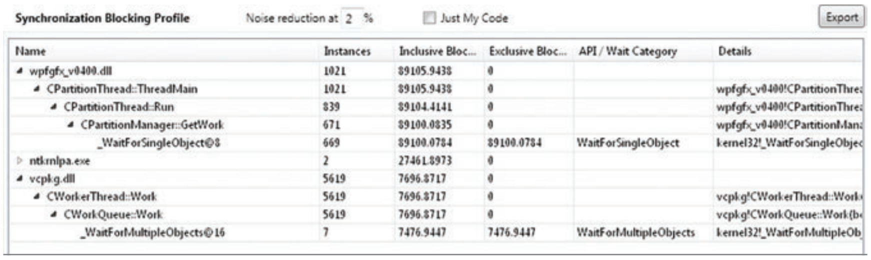
Preemption This includes preemption due to quantum expiration when a thread's share of time on its core expires. It also includes preemption due to OS scheduling rules, such as another process thread with a higher priority being ready to run. The Concurrency Visualizer also maps other sources of preemption here, such as interrupts and LPCs, which can result in interrupting a thread's execution. At each such event, the user can get the process ID/name and thread ID that took over by hovering over a preemption region and examining the tooltip (or clicking on a yellow region and observing the Current Stack tab contents). This can be a valuable feature for understanding the root causes of yellow interference in the CPU Utilization view.

Sleep This category is used to report thread blocking events as a result of an explicit request by the thread to sleep or yield its core voluntarily.

Paging/Memory Management This category covers blocking events due to memory management, which includes any blocking operations started by the system's memory manager as a response to an action by the application. Things like page faults, certain memory allocation contentions or blocking on certain resources would show up here. Page faults in particular are noteworthy because they can result in I/O. When you see a page fault blocking event, you should both examine the call stack and look for a corresponding I/O read event on the disk channel in case the page fault required I/O. A common source of such page faults is loading DLLs, memory-mapped I/O and normal virtual-memory paging by the kernel. You can identify whether this was a DLL load or paging by clicking on the corresponding I/O segment to get the filename involved.

I/O This category includes events such as blocking on file reads and writes, certain network socket operations and registry accesses. A number of operations considered by some to be network-related may not show up here, but rather in the synchronization category. This is because many I/O operations use synchronization mechanisms to block and the Concurrency Visualizer may not be looking for those API signatures in this category. Just as with the memory/paging category, when you see an I/O blocking event that seems to be related to accessing your disk drives, you should find out if there's a corresponding disk access in the disk channels. To make this easier, you can use the arrow buttons in the toolbar to move your threads closer to the disk channel. To do this, select a thread channel by clicking on its label on the left, then click on the appropriate toolbar button.

UI Processing This is the only form of blocking that is usually desirable. It is the state of a thread that is pumping messages. If your UI thread spends most of its time in this state, this implies that your application is responsive. On the other hand, if the UI thread does excessive work or blocking for other reasons, from the application user's perspective the UI will appear to hang. This category offers a great way to study the responsiveness of your application, and to tune it.



Synchronization Blocking Profile   Noise reduction at 2 %   ☐ Just My Code   Export

Name	Instances	Inclusive Bloc...	Exclusive Bloc...	API / Wait Category	Details
wpfgfx_v0400.dll	1021	89105.9438	0		
CPartitionThread::ThreadMain	1021	89105.9438	0		wpfgfx_v0400!CPartitionThre
CPartitionThread::Run	839	89104.4141	0		wpfgfx_v0400!CPartitionThre
CPartitionManager::GetWork	671	89100.0835	0		wpfgfx_v0400!CPartitionMan
_WaitForSingleObject@8	669	89100.0784	89100.0784	WaitForSingleObject	kernel32!_WaitForSingleObjec
ntkrnlpa.exe	2	27461.8973	0		
vcpkg.dll	5619	7696.8717	0		
CWorkerThread::Work	5619	7696.8717	0		vcpkg!CWorkerThread::Work
CWorkQueue::Work	5619	7696.8717	0		vcpkg!CWorkQueue::Work(b
_WaitForMultipleObjects@16	7	7476.9447	7476.9447	WaitForMultipleObjects	kernel32!_WaitForMultipleOb

Figure 3 **A Typical Profile Report**

Inter-Thread Dependencies

One of the most valuable features of the Threads view is the ability to determine inter-thread synchronization dependencies. In **Figure 2** I have selected a synchronization delay segment. The segment gets enlarged and its color is highlighted (in this case, it's red). The Current Stack tab shows the call stack of the thread at that moment. By examining the call stack, you can determine the API that resulted in blocking the thread's execution.

Another visualization feature is a line that connects the blocking segment to an execution segment on a different thread. When this visualization is visible, it illustrates the thread that ended up unblocking the blocked thread. In addition, you can click on the Unblocking stack tab in this case to see what the unblocking thread was doing when it released the blocked thread.

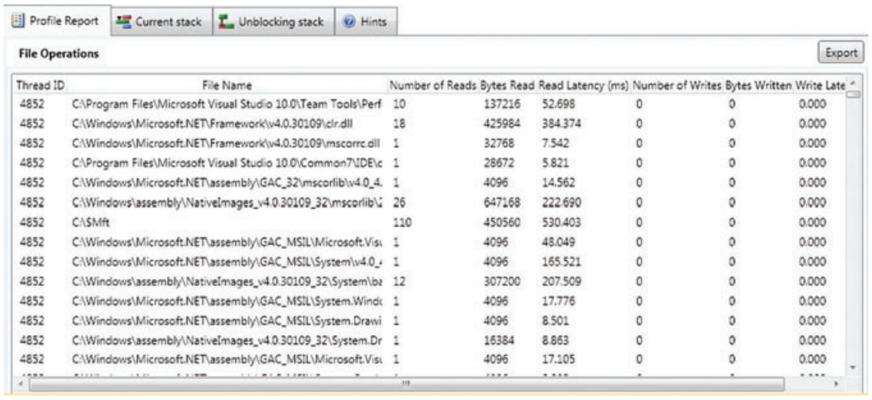
As an example, if the blocking thread was waiting on a Win32 critical section, you would see the signature of EnterCriticalSection on its blocking call stack. When it is unblocked, you should see the signature of LeaveCriticalSection in the call stack of the unblocking thread. This feature can be very valuable when analyzing complex application behavior.

Reports

The profile reports offer a simple way of identifying major contributors to the performance behavior of your application. Whether you are interested in execution overheads, blocking overheads or disk I/O, these reports allow you to focus on the most significant items that may be worth investigating.

There are four types of reports in the Threads view: execution sampling profiles, blocking profiles, file operations and per-thread summaries. All the reports are accessed using the legend. For example, to get the execution profile report, click the execution legend entry. This produces a report in the Profile Report tab. The reports look similar to what is shown in **Figure 3**.

For an execution profile report, the Concurrency Visualizer analyzes all the call stacks collected when sampling your application's execution (green segments) and collates them by identifying shared stack frames to assist the user in understanding the execution structure of the application. The tool also computes inclusive and exclusive costs for each frame. Inclusive samples account



File Operations   Export

Thread ID	File Name	Number of Reads	Bytes Read	Read Latency (ms)	Number of Writes	Bytes Written	Write Late
4852	C:\Program Files\Microsoft Visual Studio 10.0\Team Tools\Perf	10	137216	52.698	0	0	0.000
4852	C:\Windows\Microsoft.NET\Framework\v4.0.30109\clr.dll	18	425984	384.374	0	0	0.000
4852	C:\Windows\Microsoft.NET\Framework\v4.0.30109\mscorrc.dll	1	32768	7.542	0	0	0.000
4852	C:\Program Files\Microsoft Visual Studio 10.0\Common7\IDE\c	1	28672	5.821	0	0	0.000
4852	C:\Windows\Microsoft.NET\assembly\GAC_32\mscorlib\v4.0_4.	1	4096	14.562	0	0	0.000
4852	C:\Windows\assembly\NativeImages_v4.0.30109_32\mscorlib\	26	647168	222.690	0	0	0.000
4852	C:\$Mft	110	450560	530.403	0	0	0.000
4852	C:\Windows\Microsoft.NET\assembly\GAC_MSIL\Microsoft.Visu	1	4096	48.049	0	0	0.000
4852	C:\Windows\Microsoft.NET\assembly\GAC_MSIL\System\v4.0_	1	4096	165.521	0	0	0.000
4852	C:\Windows\assembly\NativeImages_v4.0.30109_32\System\ba	12	307200	207.509	0	0	0.000
4852	C:\Windows\Microsoft.NET\assembly\GAC_MSIL\System.Windo	1	4096	17.776	0	0	0.000
4852	C:\Windows\Microsoft.NET\assembly\GAC_MSIL\System.Drawi	1	4096	8.501	0	0	0.000
4852	C:\Windows\assembly\NativeImages_v4.0.30109_32\System.Dr	1	16384	8.863	0	0	0.000
4852	C:\Windows\Microsoft.NET\assembly\GAC_MSIL\Microsoft.Visu	1	4096	17.105	0	0	0.000

Figure 4 **File Operations Report**

for all samples in a given execution path, including all paths below it. Exclusive samples correspond to the number of samples of call-graph stack-frame leaves.

To get a blocking profile, you click on the blocking category of interest in the legend. The generated report is constructed like the execution profile report, but the inclusive and exclusive columns now correspond to blocking time attributed to the call stacks or frames in the report. Another column shows the number of instances of blocking attributed to that stack frame in the call tree.

Profile reports offer a simple way of identifying major contributors to the performance behavior of your application.

These reports offer a convenient way of prioritizing performance tuning efforts by identifying the parts of your application responsible for most delays. The preemption report is informational and usually does not offer any actionable data due to the nature of this category. All the reports allow you to jump to source code. You may do so by right-clicking on a stack frame of interest. The context menu that appears allows you to jump either to the function definition (the View Source option) or to the location in your application where that function was called (the View Call Sites option). If there were multiple callers, you will be presented with multiple options. This allows a seamless integration between the diagnostic data and the development process to tune your application's behavior. The reports may also be exported for cross-profile comparisons.

The File Operations report shown in **Figure 4** includes a summary of all file read and write operations visible in the current time range. For every file, the Concurrency Visualizer lists the application thread that accessed it, the number of read and write operations, the total bytes read or written, and the total read or write latency. Besides showing file operations directly attributed to the application, the Concurrency Visualizer also shows those performed by the System process. These are shown, as mentioned earlier, because they might include file operations performed by the system on behalf of your application. Exporting the report allows cross-profile comparisons during tuning efforts.

The Per Thread Summary report, shown in **Figure 5**, presents a bar graph for each thread. The bar is divided into the various thread state categories. This can be a useful tool to track your performance tuning progress. By exporting the graph data across various tuning iterations, you can document your progress and provide a means of comparing runs. The graph will not show all threads for applications that have too many threads to fit within the view.

Cores

Excessive context switches can have a detrimental effect on application performance, especially when threads migrate across cores or processor sockets when they resume execution. This is because a running thread loads instructions and data it needs (often referred to as the working set) into the cache hierarchy. When a thread resumes execution, especially on another core, it can suffer significant latency while its working set is reloaded from memory or other caches in the system.

There are two common ways to reduce this overhead. A developer can either reduce the frequency of context switches by resolving the underlying causes, or he can leverage processor or core affinity. The former is almost always more desirable because using thread affinity can be the source of other performance issues and should only be used in special circumstances. The Cores view is a tool that aids in identifying excessive context switches or performance bugs introduced by thread affinity.

As with the other views, the Cores view displays a timeline with time on the x axis. The logical cores in the system are shown on the y axis. Each thread in the application is allocated a color, and thread execution segments are drawn on the core channels. A legend and context switch statistics are shown in the bottom pane, as shown in **Figure 6**.

The statistics help the user identify threads that have excessive context switches and those that incur excessive core migrations. The user can then use this view to focus her attention on areas of execution where the threads in question are interrupted, or jump back and forth across cores by following the visual color hints. Once a region that depicts the problem is identified, the user can zoom in on it and switch back to the Threads view to understand what triggered the context switches and fix them if possible (for example, by reducing contention for a critical section). Thread affinity bugs can also manifest themselves in some cases when two or more threads contend for a single core while other cores appear to be idle.

Support for PPL, TPL and PLINQ

The Concurrency Visualizer supports the parallel programming models shipping in Visual Studio 2010 aside from existing Windows native and managed programming models. Some of the new parallel constructs—parallel_for in the Parallel Pattern Library (PPL), Parallel.For in the Task Parallel

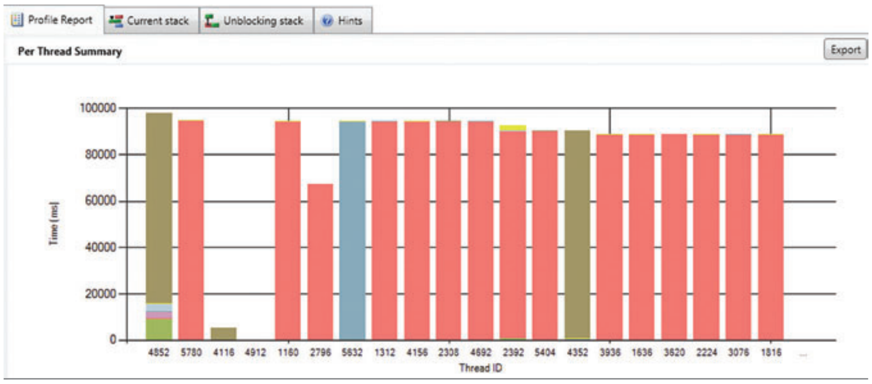
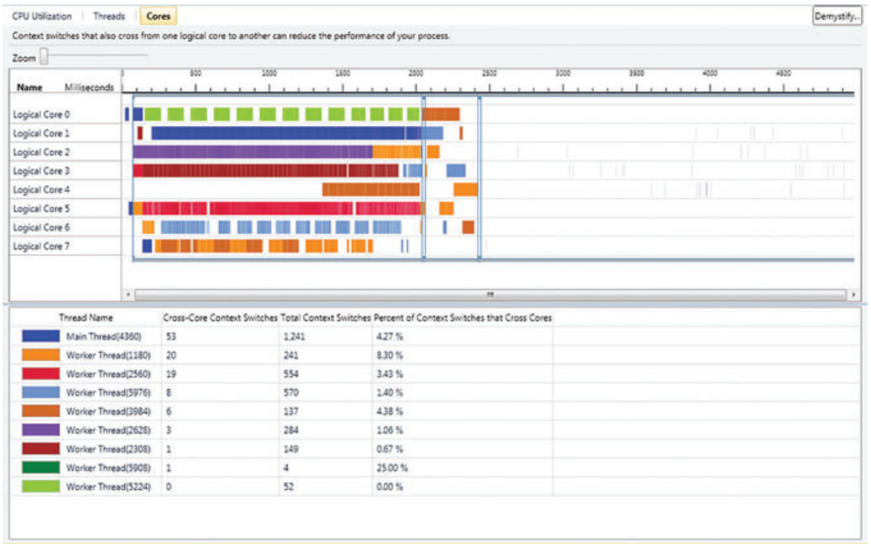


Figure 5 **Per Thread Summary Report**



Thread Name	Cross-Core Context Switches	Total Context Switches	Percent of Context Switches that Cross Cores
🟦 Main Thread(4360)	53	1,241	4.27 %
🟧 Worker Thread(1180)	20	241	8.30 %
🟥 Worker Thread(2560)	19	554	3.43 %
🟦 Worker Thread(5976)	8	570	1.40 %
🟧 Worker Thread(3984)	6	137	4.38 %
🟪 Worker Thread(2628)	3	284	1.06 %
🟥 Worker Thread(2308)	1	149	0.67 %
🟩 Worker Thread(5908)	1	4	25.00 %
🟩 Worker Thread(5224)	0	52	0.00 %

Figure 6 **Cores View**

Library (TPL) and PLINQ queries—include visualization aids in the performance tool that allow you to focus your attention on those regions of execution.

PPL requires turning on tracing for this functionality to be enabled, as shown in this example:

```
Concurrency::EnableTracing();
parallel_for (0, SIZE, 1, [&] (int i2) {
  for (int j2=0; j2<SIZE; j2++) {
    A[i2+j2*SIZE] = 1.0;
    B[i2+j2*SIZE] = 1.0;
    C[i2+j2*SIZE] = 0.0;
  }
});
Concurrency::DisableTracing();
```

When tracing is enabled, the Threads and Cores views will depict the parallel_for execution region by drawing vertical markers at the beginning and end of its execution. The vertical bars are connected via horizontal bars at the top and bottom of the view. By hovering with the mouse over the horizontal bars, a tooltip showing the name of the construct is drawn, as shown in **Figure 7**.

TPL and PLINQ do not require manual enabling of tracing for the equivalent functionality in the Concurrency Visualizer.

Collecting a Profile

The Concurrency Visualizer supports both the application launch and attach methods for collecting a profile. The behavior is exactly the same as users of the Visual Studio Profiler are accustomed to. A new profiling session may be initiated through the Analyze menu option either by launching the Performance Wizard, shown in **Figure 8**, or via the Profiler | New Performance Session option. In both cases, the Concurrency Visualizer is activated by choosing the Concurrency profiling method and then selecting the "Visualize the behavior of a multithreaded application" option.

The Visual Studio Profiler's command-line tools allow you to collect Concurrency Visualizer traces and then analyze them using the IDE. This lets users who are interested in server scenarios where installing the IDE is impossible collect a trace with the least intrusion possible.

You will notice that the Concurrency Visualizer does not have integrated support for profiling ASP.NET applications However, it may be possible to attach to the host process (usually w3wp.exe) while running your ASP.NET application in order to analyze its performance.

Since the Concurrency Visualizer uses Event Tracing for Windows (ETW), it requires administrative privileges to collect data. You can either launch the IDE as an administrator, or you will be prompted to do so when necessary. In the latter case, the IDE will be restarted with administrator rights.

Linking Visualizations to Application Phases

Another feature in the Concurrency Visualizer is an optional instrumentation library that allows developers to customize the views by drawing markers for application phases they care about. This can be extremely valuable to allow easier correlation between visualizations and application behavior. The instrumentation library is called the Scenario library and is available for download from the MSDN Code Gallery Web site at code.msdn.microsoft.com/scenario. Here's an example using a C application:

```
#include "Scenario.h"
int _tmain(int argc, _TCHAR* argv[]) {
  myScenario = new Scenario(0, L"Scenario Example", (LONG) 0);
  myScenario->Begin(0, TEXT("Initialization"));

  // Initialization code goes here

  myScenario->End(0, TEXT("Initialization"));
  myScenario->Begin(0, TEXT("Work Phase"));

  // Main work phase goes here

  myScenario->End(0, TEXT("Work Phase"));
  exit(0);
}
```

The usage is pretty simple; you include the Scenario header file and link the correct library. Then you create one or more Scenario objects and mark the beginning and end of each phase by invoking the Begin and End methods, respectively. You also specify the name of each phase to these methods. The visualization is identical to that shown in **Figure 7**, except that the tooltip will display the custom phase name you specify in your code. In addition, the

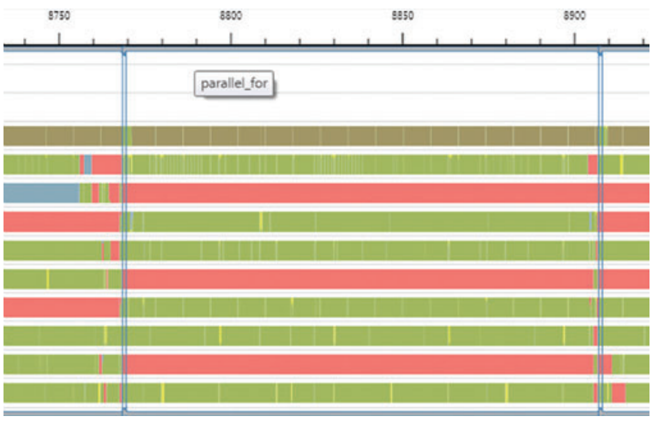


Figure 7 **An Example parallel_for Visual Marker in Threads View**

scenario markers are also visible in the CPU Utilization view, which is not the case for other markers. An equivalent managed implementation is also provided.

A word of caution is in order here. Scenario markers should be used sparingly; otherwise, the visualizations can be completely obscured by them. In fact, to avoid this problem, the tool will significantly reduce or eliminate the number of markers displayed if it detects excessive usage. In such cases, you can zoom in to expose markers that have been elided in most views. Further, when nesting of Scenario markers takes place, only the innermost marker will be displayed.

Resources and Errata

The Concurrency Visualizer includes many features to help you understand its views and reports. The most interesting such feature is the Demystify button shown in the top-right corner of all views. By clicking Demystify, you get a special mouse pointer allowing you to click on any feature in view that you'd like help on. This is our way of providing context-sensitive help in the tool.

In addition, there's a Tips tab with more help content, including a link to a gallery of visualization signatures for some common performance issues.

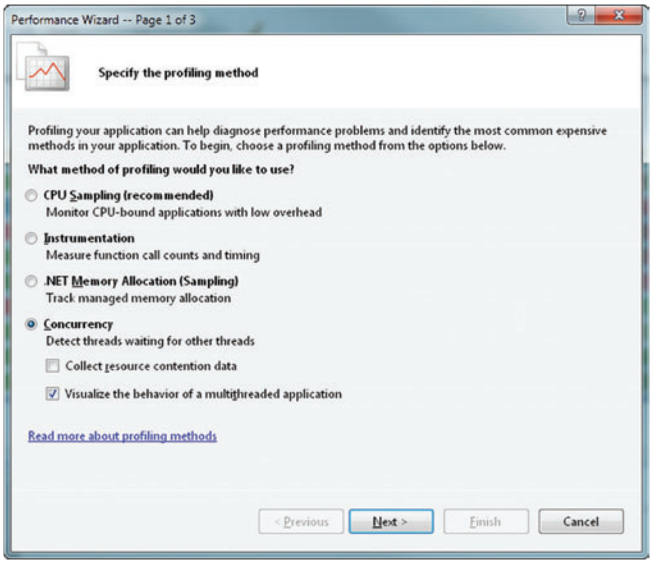
As mentioned earlier, the tool leverages ETW. Some of the events required by the Concurrency Analyzer do not exist on Windows XP or Windows Server 2003, so the tool only supports Windows Vista, Windows Server 2008, Windows 7 and Windows Server 2008 R2. Both 32-bit and 64-bit variants of these operating systems are supported.

In addition, the tool supports both native C/C++ and .NET applications (excluding .NET 1.1 and earlier). If you are not running on a supported platform, you should explore another valuable concurrency tool in Visual Studio 2010, which is enabled by selecting the "Collect resource contention data" option.

A single CPU chip today can include multiple microprocessor circuits.

In certain cases, when there's a significant amount of activity in a profiling scenario or when there is contention for I/O bandwidth from other applications, important trace events may be lost. This results in an error during trace analysis. There are two ways to handle this situation. First, you could try profiling again with a smaller number of active applications, which is a good methodology to follow in order to minimize interference while you are tuning your application. The command-line tools are an additional option in this case.

Second, you can increase the number or size of ETW memory buffers. We provide documentation through a link in the output window to instructions on how to accomplish this. If you choose option two, please set the minimum total buffer size necessary to collect a good trace since these buffers will consume important kernel resources when in use.


Performance Wizard -- Page 1 of 3

Specify the profiling method

Profiling your application can help diagnose performance problems and identify the most common expensive methods in your application. To begin, choose a profiling method from the options below.

What method of profiling would you like to use?

- CPU Sampling (recommended)
  Monitor CPU-bound applications with low overhead
- Instrumentation
  Measure function call counts and timing
- .NET Memory Allocation (Sampling)
  Track managed memory allocation
- [✓ Concurrency]
  Detect threads waiting for other threads
  - ☐ Collect resource contention data
  - ☑ Visualize the behavior of a multithreaded application

Read more about profiling methods

Figure 8 **The Performance Wizard Profiling Method Dialog**

Any diagnostic tool is only as good as the data it provides back to the user. The Concurrency Visualizer can help you pinpoint the root causes of performance issues with references to source code, but in order to do so, it needs access to symbol files. You can add symbol servers and paths in the IDE using the Tools | Options | Debugging | Symbols dialog. Symbols for your current solution will be implicitly included, but you should enable the Microsoft public symbol server as well as any other paths that are specific to the application under study where important symbol files may be found. It's also a good idea to enable a symbol cache because that will significantly reduce profile analysis time as the cache gets populated with symbol files that you need.

Although ETW provides a low-overhead tracing mechanism, the traces collected by the Concurrency Visualizer can be large. Analyzing large traces can be very time-consuming and may result in performance overheads in the visualizations provided by the tool. Generally, profiles should be collected for durations not exceeding one to two minutes to minimize the chances of these issues affecting your experience. For most analysis scenarios, that duration is sufficient to identify the problem. The ability to attach to a running process is also an important feature in order to avoid collecting data before your application reaches the point of interest.

There are multiple sources of information on the Concurrency Visualizer. Please visit the Visual Studio Profiler forum (social.msdn.microsoft.com/forums/en-us/vstsprofiler/threads) for community and development team answers. Further information is available from the team blog at blogs.msdn.com/visualizeparallel and my personal blog at blogs.msdn.com/hshafi. Please feel free to reach out to me or my team if you have any questions regarding our tool. We love hearing from people using the Concurrency Visualizer, and your input helps us improve the tool. ■

Dr. Hazim Shafi *is the parallel performance and correctness tools architect in the Parallel Computing Platform team at Microsoft. He has 15 years of experience in many aspects of parallel and distributed computing and performance analysis. He holds a B.S.E.E. from Santa Clara University, and M.S. and Ph.D. degrees from Rice University.*

Thanks to the following technical experts for reviewing this article:
Drake Campbell, Bill Colburn, Sasha Dadiomov and James Rapp

RadControls *for*

WinForms

Familiar technology. Stunning visual effects. Uncompromising performance.

UI COMPONENTS	DATA	PRODUCTIVITY TOOLS	AUTOMATED TESTING	TFS TOOLS	CMS
ASP.NET AJAX	OpenAccess ORM	JustCode	Web Testing Tools	Work Item Manager	Sitefinity
Silverlight	Reporting			Project Dashboard	
ASP.NET MVC					
WinForms					
WPF					

telerik

deliver more than expected

www.telerik.com/WinForms

MIDI Music in WPF Applications

Every PC contains a built-in 16-piece band ready to play some music. The members of this band probably feel much neglected, for they represent perhaps the most underutilized component of the array of sound and video features supported by Windows.

This 16-piece band is an electronic music synthesizer implemented in either hardware or software that conforms to the standard known as MIDI—the Musical Instrument Digital Interface. In the Win32 API, playing music through the MIDI synthesizer is supported through functions beginning with the words midiOut.

Every PC contains a built-in 16-piece band ready to play some music.

MIDI support is not part of the .NET Framework, however, so if you want to access this MIDI synthesizer in a Windows Forms or Windows Presentation Foundation (WPF) application, you'll need to use either P/Invoke or an external library.

I was very pleased to find MIDI support in the NAudio sound library available on CodePlex that I discussed in my last column. You can download that library with source code from codeplex.com/naudio. For this article I used NAudio version 1.3.8.

A Brief Example

You can think of MIDI as a high-level interface to waveform audio in which you're working with musical instruments and notes.

The MIDI standard was developed in the early 1980s. Manufacturers of electronic music synthesizers wanted a standard way to connect electronic music controllers (such as keyboards) with synthesizers, and they came up with a system to transmit small messages (mostly one, two or three bytes in length) through a cable with a 5-pin connector at the pokey rate of 3,125 bytes per second.

Two of the most important of these messages are called Note On and Note Off. When a musician presses a key on a MIDI keyboard, the keyboard generates a Note On message indicating the note that was pressed and the key's velocity. The synthesizer responds by playing that note, generally louder for higher key velocities. When the musician releases the key, the keyboard generates a Note Off message, and the synthesizer responds by turning the note off. No actual audio data goes through the MIDI cable.

Although MIDI is still used to connect electronic-music hardware, it can also be used entirely within a PC through software. Sound boards can include MIDI synthesizers, and Windows itself emulates a MIDI synthesizer entirely in software.

To access that synthesizer in your WinForms or WPF application using the NAudio library, add NAudio.dll as a reference and include this using directive in your source code:

```
using NAudio.Midi;
```

Suppose you want your application to play a single one-second note that sounds like the middle C of a piano. You can do that with the following code:

```
MidiOut midiOut = new MidiOut(0);
midiOut.Send(MidiMessage.StartNote(60, 127, 0).RawData);
Thread.Sleep(1000);
midiOut.Send(MidiMessage.StopNote(60, 0, 0).RawData);
Thread.Sleep(1000);
midiOut.Close();
midiOut.Dispose();
```

A PC might have access to multiple MIDI synthesizers; the argument to the MidiOut constructor is a numeric ID to select the one to open. The constructor will raise an exception if the MIDI output device is already in use.

A program can obtain information about the MIDI synthesizers by first using the static MidiOut.NumberOfDevices property to discover how many synthesizers are present. The numeric IDs range from 0 to one less than the number of devices. The static MidiOut.DeviceInfo method accepts a numeric ID and returns an object of type MidiOutCapabilities that describe the synthesizer. (I won't be using these features. For the remainder of this article I'll simply use the default MIDI synthesizer available with an ID of zero.)

There is a MIDI file format that combines MIDI messages with timing information.

The Send method of the MidiOut class sends a message to the MIDI synthesizer. A MIDI message comprises one, two or three bytes, but the Win32 API (and NAudio) wants them packed into

Code download available at code.msdn.microsoft.com/mag201003UIFrontiers.

a single 32-bit integer. The MidiMessage.StartNote and MidiMessage.StopNote methods do this packing for you. You can replace the two arguments to Send with 0x007F3C90 and 0x00003C80, respectively.

The first argument to StartNote and StopNote is a code ranging from 0 through 127 indicating the actual note, where the value 60 is middle C. An octave higher is 72. An octave lower is 48. The second argument is the velocity that the key is pressed or released. (Release velocities are usually ignored by synthesizers.) These can range from 0 to 127. Lower the second argument to MidiMessage.StartNote to make the note softer. (I'll discuss the third argument shortly.)

The MIDI standard was developed in the early 1980s.

The two calls to Thread.Sleep suspend the thread for 1,000 milliseconds. This is a very simple way of timing the messages, but should be avoided in a user-interface thread. The second Sleep call is necessary to let the note die off before it is abruptly truncated by the Close call.

What About Polyphony?

That's how you can play one note. What about multiple notes at the same time? That's possible as well. For example, if you wanted to play a C-major chord rather than just a single C note, you can replace the first Send message with the following:

```
midiOut.Send(MidiMessage.StartNote(60, 127, 0).RawData);
midiOut.Send(MidiMessage.StartNote(64, 127, 0).RawData);
midiOut.Send(MidiMessage.StartNote(67, 127, 0).RawData);
midiOut.Send(MidiMessage.StartNote(72, 127, 0).RawData);
```

Then replace the second Send message with:

```
midiOut.Send(MidiMessage.StopNote(60, 0, 0).RawData);
midiOut.Send(MidiMessage.StopNote(64, 0, 0).RawData);
midiOut.Send(MidiMessage.StopNote(67, 0, 0).RawData);
midiOut.Send(MidiMessage.StopNote(72, 0, 0).RawData);
```

If you want the various notes to start and stop at various times, you'll probably want to abandon the use of Thread.Sleep and get an actual timer involved, particularly if you're playing the music on a user-interface thread. More on this shortly.

There is a MIDI file format that combines MIDI messages with timing information, but these files require specialized software to create and I won't be discussing them here.

Instruments and Channels

So far I've been playing only piano sounds. You can switch the synthesizer to play sounds of other instruments using the MIDI Program Change message, implemented in NAudio with the ChangePatch method:

```
midiOut.Send(MidiMessage.ChangePatch(47, 0).RawData);
```

The first argument to ChangePatch is a numeric code ranging from 0 to 127 to indicate a particular instrument sound.

Back in the early days of MIDI, the actual sounds coming out of the synthesizers were entirely controlled by the performer through dials and patch cables. (That's why a particular synthesizer setup or instrument sound is often referred to as a "patch.") Later on, creators of MIDI files wanted a standard set of instruments so the files would sound pretty much the same regardless of the synthesizer they played on. This led to a standard called General MIDI.

A good reference for General MIDI is the Wikipedia entry en.wikipedia.org/wiki/General_midi. Under the heading "Melodic sounds" are 128 instrument sounds with codes ranging from 1 to 128. You use zero-based codes in the ChangePatch method, so code 47 in the previous example is instrument 48 in this list, which is a Timpani sound.

I mentioned at the outset that the MIDI synthesizer is equivalent to a 16-piece band. The MIDI synthesizer supports 16 *channels*. At any time, each channel is associated with a particular instrument based on the most recent Program Change message. The channel number ranges from 0 through 15 and is specified in the final argument of the StartNote, StopNote and ChangePatch methods.

Channel 9 is special. This is a percussion channel. (It's often referred to as Channel 10, but that's if the channels are numbered beginning at 1.) For channel 9, the codes passed to the StartNote and StopNote methods refer to particular non-tonal percussion sounds rather than pitches. In the Wikipedia entry on General MIDI, see the list under the heading "Percussion." For example, the following call plays a cowbell sound, which is indicated by a code of 56:

```
midiOut.Send(MidiMessage.StartNote(56, 127, 9).RawData);
```

There is much more to MIDI, but those are the essentials.

XAML-Based MIDI

In keeping with the spirit of WPF and XAML, I thought it would be fun to develop a string-based format for embedding short pieces of music directly in XAML files and playing them back. I call this format a MIDI string—a text string of notes and timing information. All tokens are separated by white space.

Notes are capital letters A through G, followed by any number of + signs or # signs (each raises the pitch one semitone) or – signs or the letter b (to lower the pitch one semitone) followed by an optional octave number, where the octave beginning at middle C is octave four. (This is a standard way of numbering octaves.) Thus, the C# below middle C is:

```
C#3
```

The letter R by itself is a rest. A note or a rest can be optionally followed by a duration, which indicates the period of time until the next note. For example, this is a quarter note, which is also the default if no duration is indicated:

```
1/4
```

Durations are sticky—that is, if a duration does not follow a note, the last duration will be used. If the duration begins with a slash, the numerator is assumed to be 1.

That duration indicates the time until the next note. This duration is also used for the length of the note—that is, the time until the note is turned off. For a more staccato sound, you may want the note's length to be less than its duration. Or you might want successive notes to overlap somewhat. You indicate a note's length the same way as the duration, but with a minus sign:

```
-3/16
```

Durations and lengths always appear after the note to which they apply, but the order doesn't matter. Lengths are not sticky. If a note length does not appear, the duration is used for the length.

Figure 1 **WpfMusicDemo.xaml Encodes Several Simple MIDI Strings**

```
<Window x:Class="WpfMusicDemo.Window1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:midi="clr-namespace:Petzold.Midi;assembly=Petzold.Midi"
  Title="WPF Music Demo"
  Height="300" Width="300">
  <Grid>
    <midi:MidiStringPlayer Name="player"
      PlayOnLoad="True"
      MidiString="{Binding ElementName=chargeButton, Path=Tag}" />

    <UniformGrid Rows="2"
      ButtonBase.Click="OnButtonClick">
      <UniformGrid.Resources>
        <Style TargetType="Button">
          <Setter Property="HorizontalAlignment" Value="Center" />
          <Setter Property="VerticalAlignment" Value="Center" />
          <Style.Triggers>
            <DataTrigger
              Binding="{Binding ElementName=player, Path=IsPlaying}"
              Value="True">
              <Setter Property="IsEnabled" Value="False" />
            </DataTrigger>
          </Style.Triggers>
        </Style>
      </UniformGrid.Resources>

      <Button Name="chargeButton"
        Content="Charge!"
        Tag="T100 I56 G4 /12 C5 E5 G5 3/16 -3/32 E5 /16 G5 /2" />

      <Button Content="Bach D-Minor Toccata"
        Tag="T24 I19 A5 /64 G5 A5 5/32 R /32 G5 /64 F5 E5 D5 C#5 /32 D5
/16 R 4/16 A4 /64 G4 A4 5/32 R /32 E4 F4 C#4 D4 /16 R 4/16 | T24 I19 A4
/64 G4 A4 5/32 R /32 G4 /64 F4 E4 D4 C#4 /32 D4 /16 R 4/16 A3 /64 G3 A3
5/32 R /32 E3 F3 C#3 D3 /16 R 4/16"/>

      <Button Content="Shave &amp; a Haircut"
        Tag="T130 I58 C5 G4 /8 G4 Ab4 /4 G4 R I75 B4 C5" />

      <Button Content="Beethoven Fifth"
        Tag="T200 I71 R /8 G4 G4 G4 Eb4 7/8 R /8 F4 F4 F4 D4 5/4 | T200
I40 R /8 G4 G4 G4 Eb4 7/8 R /8 F4 F4 F4 D4 5/4 | T200 I40 R /8 G4 G4 G4
Eb4 7/8 R /8 F4 F4 F4 D4 5/4 | T200 I41 R /8 G3 G3 G3 Eb3 7/8 R /8 F3 F3
F3 D3 5/4 | T200 I43 R /8 G2 G2 G2 Eb2 7/8 R /8 F2 F2 F2 D2 5/4 | T200
I43 R /8 G2 G2 G2 Eb2 7/8 R /8 F2 F2 F2 D2 5/4"/>

    </UniformGrid>
  </Grid>
</Window>
```

Notes can also be preceded by tokens. To set an instrument voice, follow the letter I by the zero-based patch number. For example, this indicates a violin for the successive notes:

```
I40
```

The piano is the default patch.

To set a new volume (that is, a velocity) for successive notes use V, such as:

```
V64
```

For both I and V, the number that follows must range from zero through 127.

By default, the tempo is 60 quarter notes per minute. To set a new tempo for the following notes, use T followed by the number of quarter notes per minute, for example:

```
T120
```

If you'd like a group of notes to be played with all the same parameters, you can put them in parentheses. Here's a C-major chord:

```
(C4 E4 G4 C5)
```

Only notes may appear in parentheses.

The vertical bar | separates channels. The channels are played simultaneously, and they are entirely independent, including the tempos.

If a particular channel contains a capital P anywhere within, that channel becomes the percussion channel. That channel can contain notes or rests in the normal notation, but also allows percussion voices to be indicated numerically. For example, this is the cowbell:

```
P56
```

If you go to en.wikipedia.org/wiki/Charge_(fanfare), you'll see the "Charge!" tune often played at sporting events. That can be expressed in the MIDI string format as:

```
"T100 I56 G4 /12 C5 E5 G5 3/16 -3/32 E5 /16 G5 /2"
```

The MidiStringPlayer

The MidiStringPlayer is the only public class in the Petzold.Midi library project included with the downloadable source code. It derives from FrameworkElement so you can embed it in the visual tree in a XAML file, but it has no visual appearance. Set the MidiString property to a string in the format shown in the previous example and call Play (and, optionally, Stop to stop the sequence before it's completed).

The Win32 API includes a high-resolution timer specifically for playing MIDI sequences.

MidiStringPlayer also has a PlayOnLoad property to play a sequence when the element loads, and a get-only IsPlaying property. The element generates an Ended event when it's completed playing a sequence, and a Failed event that's fired if there's an error in the syntax of the MIDI string. The event includes an offset in the text string indicating the problematic token and a text explanation of the error.

Two WPF programs are also included in the downloadable code. The MusicComposer program lets you interactively put together a MIDI string. The WpfMusicDemo program encodes some simple sequences in a MIDI file, as shown in **Figure 1**.

A crucial part of any piece of music-playing software is the timer, but for MidiStringPlayer I used the very simple DispatcherTimer, which runs on the UI thread. This is certainly not optimum. If another program is hogging the CPU, the music playback will become irregular. DispatcherTimer also cannot generate Tick events faster than about 60 per second, which is satisfactory for simple pieces, but doesn't provide the necessary precision of for more rhythmically complex music.

The Win32 API includes a high-resolution timer specifically for playing MIDI sequences, but this has not yet made it to the NAudio library. Perhaps at some later time I'll replace the DispatcherTimer with something a little more precise and regular, but for now I'm happy that it works as well as it does with this simple solution. ■

CHARLES PETZOLD *is a long-time Contributing Editor to* MSDN Magazine. *His most recent book is "The Annotated Turing: A Guided Tour through Alan Turing's Historic Paper on Computability and the Turing Machine" (Wiley, 2008). Petzold blogs on his Web site charlespetzold.com.*

THANKS to the following technical expert for reviewing this article: *Mark Heath*

Does your Team do more than just track bugs?

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

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

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

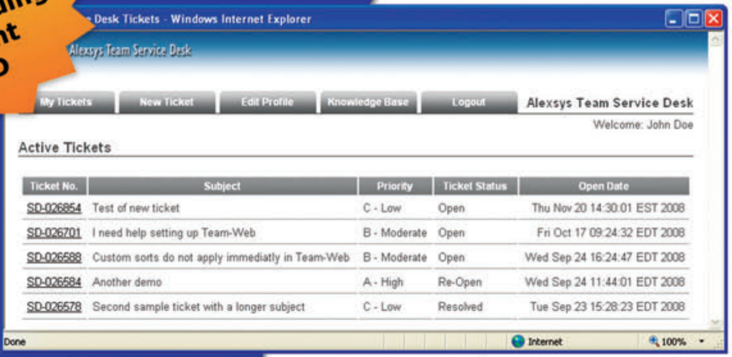
- Quality Control / Compliance Tracking
- Project Management
- End User Accessible Service Desk Portal
- Bugs and Features
- Action Items
- Sales and Marketing
- Help Desk

Native Smart Card Login Support including Government and DOD

Alexsys Team

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

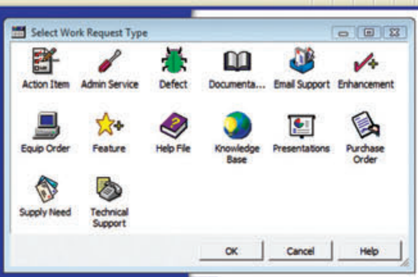


Alexsys Team Service Desk

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



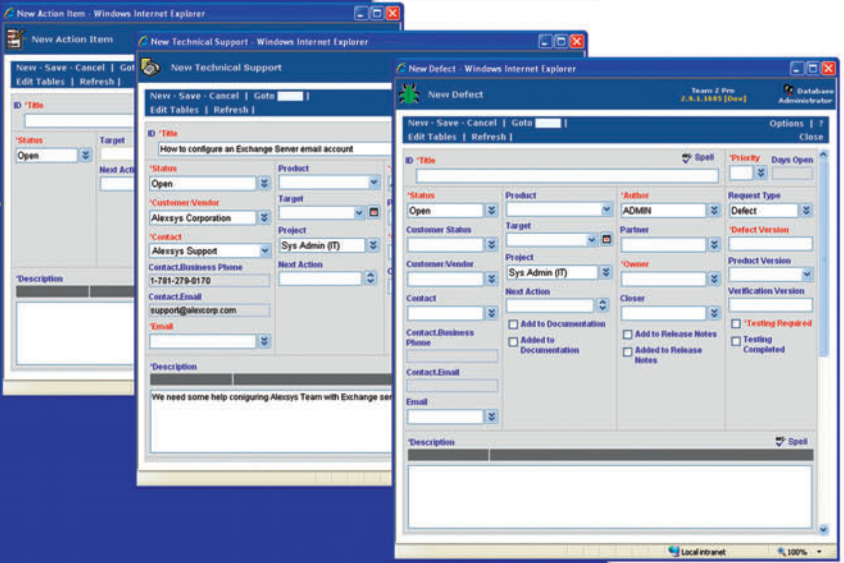
Service Desk Features

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

Team 2 Features

- Windows and Web Clients
- Multiple Work Request Forms
- Customizable Database
- Point and Click Workflows
- Role Based Security
- Clear Text Database
- Project Trees
- Time Recording
- Notifications and Escalations
- Outlook Integration

ZDNet Editors' Pick



New Action Item: ID *Title; *Status: Open; Target; Next Action; *Description

New Technical Support: ID *Title: How to configure an Exchange Server email account; *Status: Open; *Customer Vendor: Alexsys Corporation; *Contact: Alexsys Support; Contact.Business Phone: 1-781-279-0170; Contact.Email: support@alexcorp.com; *Email; Product; Target; Project: Sys Admin (IT); Next Action; *Description: We need some help coniguring Alexsys Team with Exchange ser

New Defect: ID *Title; *Priority; Days Open; *Status: Open; Customer Status; Customer Vendor; Contact; Contact.Business Phone; Contact.Email; Email; Product; Target; Project: Sys Admin (IT); Next Action; *Author: ADMIN; Partner; *Owner; Closer; Request Type: Defect; *Defect Version; Product Version; Verification Version; *Description

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.

Generic Co- and Contravariance in Visual Basic 2010

Visual Studio 2010 has a new feature called generic co- and contravariance that is available when working with generic interfaces and delegates. In versions that precede Visual Studio 2010 and the Microsoft .NET Framework 4, generics behave invariantly with respect to subtyping, so conversions between generic types with different type arguments aren't allowed.

For example, if you try passing a List(Of Derived) to a method that accepts an IEnumerable(Of Base), you'll get an error. But Visual Studio 2010 can handle type-safe co- and contravariance that supports declaration of covariant and contravariant type parameters on generic interfaces and delegate type. In this article I'll discuss what this feature really is and how you can take advantage of it in your applications.

Because a button is a control, you'd expect this code to work because of basic object-oriented inheritance principles:

```
Dim btnCollection As IEnumerable(Of Button) = New List(Of Button) From
{New Button}
Dim ctrlCollection As IEnumerable(Of Control) = btnCollection
```

It's not allowed, though, in Visual Studio 2008, which will give the error "IEnumerable(Of Button) cannot be converted to IEnumerable(Of Control)." But as object-oriented programmers, we know that a value of type Button can be converted to a control, so as noted earlier, according to basic inheritance principles, the code should be allowed.

Consider the following example:

```
Dim btnCollection As IList(Of Button) = New List(Of Button) From {New
Button}
Dim ctrlCollection As IList(Of Control) = btnCollection
ctrlCollection(0) = New Label
Dim firstButton As Button = btnCollection(0)
```

This would fail with an InvalidCastException because the programmer converted the IList(Of Button) into an IList(Of Control), then inserted a control into it that wasn't even a button.

Visual Studio 2010 recognizes and allows code like that in the first case, but can still disallow the type of code shown in the second case when targeting the .NET Framework 4. For most users, and the majority of the time, programs will just work in the expected way and there'll be no need to dig deeper. In this article, though, I will dig deeper to explain how and why the code works.

Covariance

In the first listing, which viewed an IEnumerable(Of Button) as an IEnumerable(Of Control), why was the code safe in Visual Studio 2010, while the second code sample, which viewed an IList(Of Button) as an IList(Of Control), unsafe?

The first is OK because IEnumerable(Of T) is an "out" interface, which means that in IEnumerable(Of Control), users of the interface can only take controls *out* of the list.

The second is unsafe because IList(Of T) is an "in-and-out" interface, so in IList(Of Control), users of the interface can *put in* controls as well as take them out.

The new language feature in Visual Studio 2010 that allows this is called generic covariance. In the .NET Framework 4, Microsoft has rewritten the framework along these lines:

```
Interface IEnumerable(Of Out T)
...
End Interface
Interface IList(Of T)
...
End Interface
```

The Out annotation in IEnumerable(Of Out T) indicates that if a method in IEnumerable mentions T, it will do so only in an out position, such as that for the return of a function or the type of a read-only property. This allows users to cast any IEnumerable(Of Derived) into an IEnumerable(Of Base) without running into an InvalidCastException.

IList lacks an annotation because IList(Of T) is an in-and-out interface. As a result, users can't cast IList(Of Derived) into IList(Of Base) or vice versa; doing so could lead to an InvalidCastException, as you saw above.

Contravariance

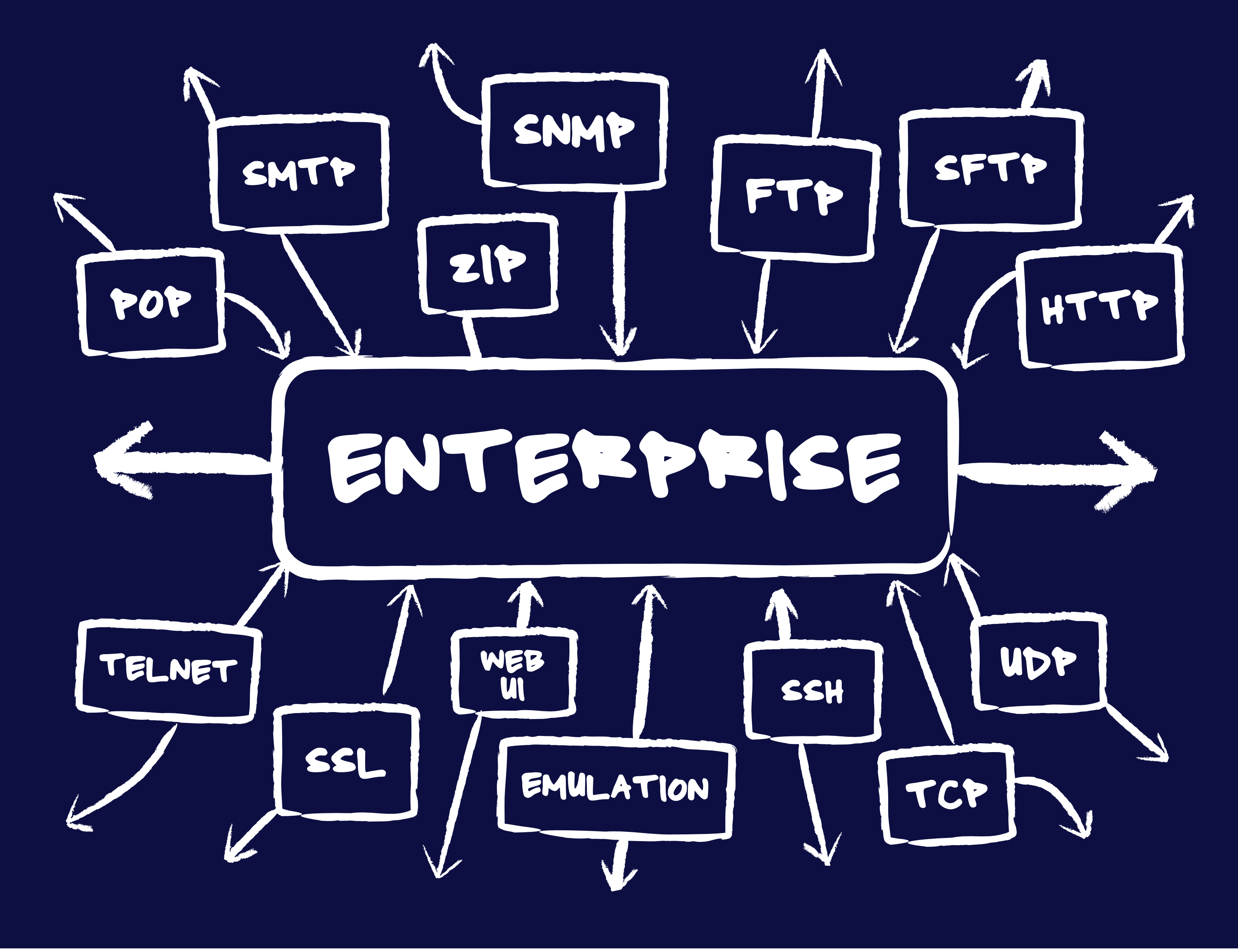
There's a mirror of the Out annotation. It's a bit more subtle, so I'll start with an example:

```
Dim _compOne As IComparer(Of Control) = New MyComparerByControlName()
Dim _compTwo As IComparer(Of Button) = _compOne
Dim btnOne = new Button with {.Name = "btnOne", OnClick = AddressOf
btnOneClick, Left=20}
Dim btnTwo = new Button with {.Name = "btnTwo", OnClick = AddressOf
btnTwoClick, Left=100}
Dim areSame = _compTwo.Compare(btnOne, btnTwo)
```

Here I've created a comparer that can determine whether any two controls are the same, which it does by looking at their names only.

Because the comparer can compare *any controls at all*, it certainly has the ability to evaluate two controls that happen to be buttons. That's why you can safely cast it to an IComparer(Of Button). In general, you can safely cast an IComparer(Of Base) into any IComparer(Of

This article is based on a prerelease version of Visual Studio 2010.
All information is subject to change.



Internet Connectivity for the Enterprise

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

PowerSNMP for ActiveX and .NET

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

PowerWEB for ASP.NET

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

PowerTCP for ActiveX and .NET

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

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

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

Download a fully functional product trial today!

DART.com

Figure 1 **An Ambiguous Conversion**

```
Option Strict On
Imports System.Windows.Forms
Interface IComparer(Of Out Tout)
End Interface
Class Comparer
    Implements IComparer(Of Button)
    Implements IComparer(Of CheckBox)
End Class

Module VarianceExample
    Sub Main()
        Dim iComp As IComparer(Of Control) = New Comparer()
    End Sub
End Module
```

Derived). This is called contravariance. It's done with In annotations and is the exact mirror image of the Out annotations.

The .NET Framework 4 has also been modified to incorporate In generic type parameters:

```
Interface IComparer(Of In T)
 ...
End Interface
```

Because of the IComparer(Of T) In annotation, every method in IComparer that mentions T will do so only in an in position such as that of a ByVal argument or the type of a write-only property. Thus users can cast an IComparer(Of Base) into any IComparer(Of Derived) without running into an InvalidCastException.

Let's consider an example of the Action delegate in .NET 4 where the delegate becomes contravariant in T:

```
Dim actionControl As Action(Of Control)
Dim actionButton As Action(Of Button) = actionControl
```

The example works in .NET 4 because the user of the delegate actionButton will always invoke it with Button arguments, which are controls.

You can add In and Out annotations to your own generic interfaces and delegates as well. But because of common language runtime (CLR) limitations, you can't use these annotations on classes, structures or anything else. In short, only interfaces and delegates can be co- or contravariant.

Declarations/Syntax

Visual Basic uses two new contextual keywords: *Out*, which introduces covariance, and *In*, which does the same for contravariance, as illustrated in this example:

```
Public Delegate Function Func(Of In TArg, Out TResult)(ByVal arg As
TArg) As TResult

Public Interface IEnumerable(Of Out Tout)

  Inherits IEnumerable
  Function GetEnumerator() As IEnumerator(Of Tout)
End Interface

Public Interface IEnumerator(Of Out Tout)
  Inherits IEnumerator
  Function Current() As Tout
End Interface

Public Interface IComparer(Of In Tin)
  Function Compare(ByVal left As Tin, ByVal right As Tin) As Integer
End Interface
```

Why do we need these two contextual keywords or the syntax at all, though? Why don't we infer variance In/Out automatically? First, it's useful for programmers to declare their intent. Second, there are places where the compiler can't infer the best variance annotation automatically.

Let's consider two interfaces, IReadWriteBase and IReadWrite:

```
Interface IReadWriteBase(Of U)
  Function ReadWrite() As IReadWrite(Of U)
End Interface
Interface IReadWrite(Of T) : Inherits IReadWriteBase(Of T)
End Interface
```

If the compiler infers them both to be Out, as below, the code works fine:

```
Interface IReadWriteBase(Of Out U)
  Function ReadWrite() As IReadWrite(Of U)
End Interface
Interface IReadWrite(Of Out T)
  Inherits IReadWriteBase(Of T)
End Interface
```

And if the compiler infers both to be In, as shown here, again the code works fine:

```
Interface IReadWrite(Of In T)
  Inherits IReadWriteBase(Of T)
End Interface
Interface IReadWriteBase(Of In U)
  Function ReadWrite() As IReadWrite(Of U)
End Interface
```

The compiler can't know which one to pick—In or Out—so it provides a syntax.

Only interfaces and delegates can be co- and contravariant, and only when the type arguments are reference types.

Out/In contextual keywords appear in interface and delegate declarations only. Using the keywords in any other generic parameter declaration will cause a compile-time error. The Visual Basic compiler doesn't allow a variant interface to contain nested enumerations, classes and structures because the CLR doesn't support variant classes. You can, however, nest variant interfaces inside a class.

Dealing with Ambiguity

Co- and contravariance introduce ambiguity in member lookup, so you should know what triggers ambiguity and how the Visual Basic compiler handles it.

Figure 2 **A Conversion that Succeeds at Runtime**

```
Option Strict On
Imports System.Windows.Forms
Interface IEnumerable(Of Out Tout)
End Interface
Class ControlList
   Implements IEnumerable(Of Button)
   Implements IEnumerable(Of CheckBox)
End Class

Module VarianceExample
    Sub Main()
     Dim _ctrlList As IEnumerable(Of Control) = CType(New ControlList,
IEnumerable(Of Button))
     Dim _ctrlList2 As IEnumerable(Of Control) = CType(New ControlList,
IEnumerable(Of CheckBox))
    End Sub
End Module
```

Figure 3 **Different Results from the Same Method**

```
Option Strict Off
Module VarianceExample
    Sub Main()
        Dim _comp As IComparer(Of Account) = New Comparer1()
        Dim _comp2 As IComparer(Of Account) = New Comparer2()

        Dim _account = New Account With {.AccountType = "Checking",
.IsActive = True}
        Dim _account2 = New Account With {.AccountType = "Saving",
.IsActive = False}

        '// Even though _comp and _comp2 are *IDENTICAL*, they give
different results!
        Console.WriteLine(_comp.Compare(_account, _account2)) '; //
prints 0
        Console.WriteLine(_comp2.Compare(_account, _account2)) '; //
prints -1

    End Sub
    Interface IAccountRoot
        Property AccountType As String
    End Interface
    Interface IAccount
        Inherits IAccountRoot
        Property IsActive As Boolean
    End Interface
    Class Account
        Implements IAccountRoot, IAccount
        Public Property AccountType As String Implements IAccountRoot.
AccountType
        Public Property IsActive As Boolean Implements IAccount.IsActive
    End Class

    Class Comparer1
        Implements IComparer(Of IAccountRoot), IComparer(Of IAccount)

        Public Function Compare(ByVal x As IAccount, ByVal y As IAccount)
As Integer Implements System.Collections.Generic.IComparer(Of IAccount).
Compare
            Dim c As Integer = String.Compare(x.AccountType,
y.AccountType)
            If (c <> 0) Then
                Return c
            Else
                Return (If(x.IsActive, 0, 1)) - (If(y.IsActive, 0, 1))
            End If

        End Function

        Public Function Compare(ByVal x As IAccountRoot, ByVal y
As IAccountRoot) As Integer Implements System.Collections.Generic.
IComparer(Of IAccountRoot).Compare
            Return String.Compare(x.AccountType, y.AccountType)
        End Function
    End Class

    Class Comparer2
        Implements IComparer(Of IAccount), IComparer(Of IAccountRoot)

        Public Function Compare(ByVal x As IAccount, ByVal y As IAccount)
As Integer Implements System.Collections.Generic.IComparer(Of IAccount).
Compare
            Dim c As Integer = String.Compare(x.AccountType,
y.AccountType)
            If (c <> 0) Then
                Return c
            Else
                Return (If(x.IsActive, 0, 1)) - (If(y.IsActive, 0, 1))
            End If
        End Function

        Public Function Compare(ByVal x As IAccountRoot, ByVal y
As IAccountRoot) As Integer Implements System.Collections.Generic.
IComparer(Of IAccountRoot).Compare
            Return String.Compare(x.AccountType, y.AccountType)
        End Function
    End Class
End Module
```

Let's consider the example in **Figure 1**, in which we try to convert Comparer to IComparer(Of Control) where Comparer implements IComparer(Of Button) and IComparer(Of CheckBox).

Because both IComparer(Of Button) and IComparer(Of CheckBox) are variant-convertible to IComparer(Of Control), the conversion will be ambiguous. As a result, the Visual Basic compiler looks for ambiguities according to the CLR rules, and if Option Strict is On, disallows such ambiguous conversions at compile time; if Option Strict is Off, the compiler generates a warning.

The conversion in **Figure 2** succeeds at runtime and doesn't generate a compile-time error.

Figure 3 illustrates the danger of implementing both IComparer(of IBase) and IComparer(of IDerived) generic interfaces. Here, Comparer1 and Comparer2 classes implement the same variant generic interface with different generic type parameters in different order. Even though Comparer1 and Comparer2 are identical, apart from ordering when they implement the interface, the call to the Compare method in those classes gives different results.

Here's why different results emerge from the code in **Figure 3**, even though _comp and _comp2 are identical. The compiler just emits Microsoft Intermediate Language that performs the cast. Thus the choice of whether to get the IComparer(Of IAccountRoot) or IComparer(Of IAccount) interface, given an implementation of the Compare() method that's different, falls to the CLR, which always picks the first assignment-compatible interface in the list of interfaces. So with the code in **Figure 3**, the Compare() method gives different results because the CLR chooses the IComparer(Of IAccountRoot) interface for Comparer1 class and the IComparer(Of IAccount) interface for Comparer2 class.

Figure 4 **How a Generic Constraint Encompasses Variance-Convertibility**

```
Option Strict On
Imports System.Windows.Forms
Module VarianceExample
    Interface IEnumerable(Of Out Tout)
    End Interface
    Class List(Of T)
        Implements IEnumerable(Of T)
    End Class
    Class Program
        Shared Function Foo(Of T As U, U)(ByVal arg As T) As U
            Return arg
        End Function

        Shared Sub Main()
            'This is allowed because it satisfies the constraint Button
AS Control
            Dim _ctrl As Control = Foo(Of Button, Control)(New Button)
            Dim _btnList As IEnumerable(Of Button) = New List(Of Button)
()
            'This is allowed because it satisfies the constraint
IEnumerable(Of Button) AS IEnumerable(Of Control)
            Dim _ctrlCol As IEnumerable(Of Control) = Foo(Of
IEnumerable(Of Button), IEnumerable(Of Control))(_btnList)

        End Sub
    End Class
End Module
```

Figure 5 **Where a Constraint Is Useful in a Collection of Objects**

```
Option Strict On
Imports System.Windows.Forms

Interface IPipe(Of Out Tout, In Tin As Tout)
    Sub Push(ByVal x As Tin)
    Function Pop() As Tout
End Interface

Class Pipe(Of T)
    Implements IPipe(Of T, T)

    Private m_data As Queue(Of T)

    Public Function Pop() As T Implements IPipe(Of T, T).Pop
        Return m_data.Dequeue()
    End Function

    Public Sub Push(ByVal x As T) Implements IPipe(Of T, T).Push
        m_data.Enqueue(x)
    End Sub
End Class

Module VarianceDemo
    Sub Main()
        Dim _pipe As New Pipe(Of ButtonBase)
        Dim _IPipe As IPipe(Of Control, Button) = _pipe
    End Sub
End Module
```

Constraints on a Generic Interface

When you write a generic constraint—(Of T As U, U)—As now encompasses variance-convertibility in addition to inheritance. **Figure 4** demonstrates that (of T As U, U) encompasses variance-convertibility. A variant generic parameter can be constrained to a different variant parameter. Consider this:

```
Interface IEnumerable(Of In Tin, Out Tout As Tin)
End Interface
Interface IEnumerable(Of Out Tout, In Tin As Tout)
End Interface
```

In this example, an IEnumerator(Of ButtonBase, ButtonBase) can be variance-converted to an IEnumerator(Of Control, Button), and IEnumerable(Of Control, Button) can be converted to IEnumerable(Of ButtonBase, ButtonBase) with the constraints still satisfied. Notionally, it could be further variance-converted to IEnumerable(Of ButtonBase, Control), but this no longer satisfies the constraints, so it isn't a valid type. **Figure 5** represents a first-in, first-out collection of objects where a constraint could be useful.

In **Figure 5**, if I give _IPipe to you, you can only push Button into the pipe, and you can only read Control from it. Note that you can constrain a variant interface to a value type, given that the interface will never allow a variance conversion. Here's an example with value type constraint in a generic parameter:

```
Interface IEnumerable(Of Out Tout As Structure)
End Interface
```

Constraint to value type might be useless, given that variance conversion is not allowed on a variant interface instantiated with value types. But note that with Tout being a structure, it might be useful to infer the type indirectly through constraints.

Constraints on a Function's Generic Parameters

Constraints in methods/functions must have In types. Here are two basic ways of thinking about constraints on a function's generic parameters:

- In most cases, a generic parameter to a function is basically an input to the function, and all inputs must have In types.
- A client can variance-convert any Out type into System.Object. If a generic parameter is constrained to some Out type, then effectively, a client can remove that constraint, which isn't what constraints are about.

Let's consider **Figure 6**, which makes clear what would happen without this variance-validity rule on constraints.

In **Figure 6**, we've stored a Label inside m_data as Button, which is illegal. Therefore, constraints in methods/functions must have In types.

Overload Resolution

Overloading refers to creating multiple functions with the same name that take different argument types. Overload resolution is a compile-time mechanism for selecting the best function from a set of candidates.

Let's take a look at the following example:

```
Private Overloads Sub Foo(ByVal arg As Integer)
End Sub
Private Overloads Sub Foo(ByVal arg As String)
End Sub

Foo(2)
```

What actually happens behind the scenes here? When the compiler sees the call to Foo(2), it has to figure out which Foo you want to invoke. To do so, it uses the following simple algorithm:

1. Generate a set of all applicable candidates by looking up everything with the name Foo. In our example, there are two candidates to consider.
2. For each candidate, look at the arguments and remove non-applicable functions. Note that the compiler also performs a little verification and type inference for generics.

With the introduction of variance, a set of predefined conversions is expanded, and as the result, Step 2 will accept more candidate functions than there were before. Also, in cases where there used to be two

Figure 6 **What Happens Without a Variance-Validity Rule on Constraints**

```
Option Strict On
Imports System.Windows.Forms
Interface IEnumerable(Of Out Tout)
    Sub Foo(Of U As Tout)(ByVal arg As U)
End Interface

Class List(Of T)
    Implements IEnumerable(Of T)

    Private m_data As T
    Public Sub Foo(Of U As T)(ByVal arg As U) Implements IEnumerable(Of
T).Foo
        m_data = arg
    End Sub
End Class

Module VarianceExample
    Sub Main()
        'Inheritance/Implements
        Dim _btnCollection As IEnumerable(Of Button) = New List(Of
Button)
        'Covariance
        Dim _ctrlCollection As IEnumerable(Of Control) = _btnCollection
        'Ok, Constraint-satisfaction, because Label is a Control
        _ctrlCollection.Foo(Of Label)(New Label)
    End Sub
End Module
```

Figure 7 **Code that Might Break with the Addition of Variance into Visual Basic**

```
Option Strict On
Imports System.Windows.Forms
Imports System.Collections.Generic
Module VarianceExample
    Sub Main()
        Dim _ctrlList = New ControlList(Of Button)
        'Picks Add(ByVal f As IEnumerable(Of Control)), Because of
variance-convertibility
        _ctrlList.Add(New ControlList(Of Button))
    End Sub
End Module
Interface IEnumerable(Of Tout)
End Interface
Class ControlList(Of T)
    Implements IEnumerable(Of T)

    Sub Add(ByVal arg As Object)
    End Sub

    Sub Add(ByVal arg As IEnumerable(Of Control))
    End Sub
End Class
```

equally specific candidates, the compiler would have picked the unshadowed one, but now the shadowed one may be wider, so the compiler may pick it instead. **Figure 7** demonstrates code that could potentially break with the addition of variance into Visual Basic.

In Visual Studio 2008, the call to Add would bind to Object, but with Visual Studio 2010's variance-convertibility, we use IEnumerable(Of Control) instead.

The compiler picks a narrowing candidate only if there's no other, but with variance-convertibility, if there's a new widening candidate, the compiler picks it instead. If variance-convertibility makes another new narrowing candidate, the compiler emits an error.

Extension Methods

Extension methods enable you to add methods to existing types without creating a new derived type, recompiling or otherwise modifying the original type. In Visual Studio 2008, extension methods support array covariance, as in the following example:

```
Option Strict On
Imports System.Windows.Forms
Imports System.Runtime.CompilerServices
Module VarianceExample
  Sub Main()
    Dim _extAdd(3) As Button 'Derived from Control
    _extAdd.Add()
  End Sub

  <Extension()>
  Public Sub Add(ByVal arg() As Control)
     System.Console.WriteLine(arg.Length)
  End Sub
```

But in Visual Studio 2010, extension methods also dispatch on generic variance. This may be a breaking change, as shown in **Figure 8**, because you may have more extension candidates than before.

User-Defined Conversions

Visual Basic allows you to declare conversions on classes and structures so that they can be converted to or from other classes and structures as well as basic types. In Visual Studio 2010, variance-convertibility is already added into user-defined conversion algorithms. Therefore, the scope of every user-defined conversion will increase automatically, which might introduce breaks.

Because Visual Basic and C# don't allow user-defined conversions on interfaces, we need worry only about delegate types. Consider the conversion in **Figure 9**, which works in Visual Studio 2008 but causes an error in Visual Studio 2010.

Figure 10 gives another example of a conversion that would cause a compile-time error in Visual Studio 2008 with Option Strict On, but will succeed in Visual Studio 2010 with variance-convertibility.

Effects of Option Strict Off

Option Strict Off normally allows narrowing conversions to be done implicitly. But whether Option Strict is On or Off, variance-convertibility requires its generic arguments to be related through the CLR's assignment-compatible widening; it's not enough for them to be related through narrowing (see **Figure 11**). Note: We do count T->U as narrowing if there's a variance conversion U->T, and we count T->U as narrowing if T->U is ambiguous.

Restrictions on Co-and Contravariance

Here's a list of restrictions with co- and contravariance:

1. In/Out contextual keywords may appear in interface declarations or in delegate declarations. Using the keywords in any other generic parameter declaration results in a compile-time error. A variant interface can't nest a class or structure inside it but can contain nested interfaces and nested delegates, which will then take on variance from the containing type.

Enumerations, classes, events and structures are not allowed to go inside a variant interface.

2. Only interfaces and delegates can be co- or contravariant, and only when the type arguments are reference types.
3. Variance conversion can't be performed on variant interfaces instantiated with value types.
4. Enumerations, classes, events and structures are not allowed to go to inside a variant interface. This is because we emit these

Figure 8 **A Breaking Change**

```
Option Strict On
Imports System.Runtime.CompilerServices
Imports System.Windows.Forms
Module VarianceExample
    Sub Main()
        Dim _func As Func(Of Button) = Function() New Button
        'This was a compile-time error in VB9, But in VB10 because of
variance convertibility, the compiler uses the extension method.
        _func.Add()
    End Sub

    <Extension()> _
    Public Sub Add(ByVal this As Func(Of Control))
        Console.WriteLine("A call to func of Control")
    End Sub
End Module
```

Figure 9 **A Conversion that Works in Visual Studio 2008 but Produces an Error in Visual Studio 2010**

```
Option Strict On
Imports System.Windows.Forms
Module VarianceExample
    Class ControlList
        Overloads Shared Widening Operator CType(ByVal arg As
ControlList) As Func(Of Control)
            Console.WriteLine("T1->Func(Of Control)")
            Return Function() New Control
        End Operator
    End Class
    Class ButtonList
        Inherits ControlList
        Overloads Shared Widening Operator CType(ByVal arg As ButtonList) As
Func(Of Button)
            Console.WriteLine("T2->Func(Of Button)")
            Return Function() New Button
        End Operator
    End Class
    Sub Main()
       'The conversion works in VB9 using ButtonList->ControlList->Func(Of
Control)
'Variance ambiguity error in VB10, because there will be another widening path
(ButtonList-->Func(Of Button)--[Covariance]-->Func(Of Control)
        Dim _func As Func(Of Control) = New ButtonList
    End Sub
End Module
```

classes/structures/enumerations as generic, inheriting their containers' generic parameters, so they end up inheriting their containers' variance. Variant classes/structures/enumerations are disallowed by the CLI spec.

More-Flexible, Cleaner Code

When working with generics, in some cases you may have known you could have written simpler or cleaner code if co- and contravariance had been supported. Now that these features are implemented in Visual Studio 2010 and the .NET Framework 4, you can make your code much cleaner and more flexible by declaring variance properties on type parameters in generic interfaces and delegates.

To facilitate this, in the .NET Framework 4, IEnumerable is now declared covariant using the Out modifier in its type parameter, and IComparer is declared contravariant using the In modifier. So for

Figure 10 **Visual Studio 2010 Allows a Formerly Illegal Conversion by Using Variance Convertibility**

```
Option Strict On
Imports System.Windows.Forms
Module VarianceExample
    Class ControlList
        Overloads Shared Narrowing Operator CType(ByVal arg As
ControlList) As Func(Of Control)
            Return Function() New Control
        End Operator

        Overloads Shared Widening Operator CType(ByVal arg As
ControlList) As Func(Of Button)
            Return Function() New Button
        End Operator
    End Class

    Sub Main()
'This was an error in VB9 with Option Strict On, but the conversion will
succeed in VB10 using Variance->Func(Of Button)-[Covariance]-Func(Of
Control)
        Dim _func As Func(Of Control) = New ControlList
    End Sub
End Module
```

Figure 11 **With Option Strict Off**

```
Option Strict Off
Imports System.Windows.Forms
Module VarianceExample
    Interface IEnumerable(Of Out Tout)
    End Interface
    Class ControlList(Of T)
       Implements IEnumerable(Of T)
    End Class
    Sub Main()
       'No compile time error, but will throw Invalid Cast Exception at
run time
        Dim _ctrlList As IEnumerable(Of Button) = New ControlList(Of
Control)
    End Sub
End Module
```

IEnumerable(Of T) to be variance-convertible to IEnumerable(Of U), you have to have one of the following conditions:

- Either T inherits from U or
- T is variance-convertible to U or
- T has any other kind of predefined CLR reference conversion

In the Basic Class Library, these interfaces are declared as follows:

```
Interface IEnumerable(Of Out T)
  Function GetEnumerator() As IEnumerator(Of T)
End Interface
Interface IEnumerator(Of Out T)
  Function Current() As T
End Interface
Interface IComparer(Of In T)
  Function Compare(ByVal arg As T, ByVal arg2 As T) As Integer
End Interface
Interface IComparable(Of In T)
  Function CompareTo(ByVal other As T) As Integer
End Interface
```

Variance conversion can't be performed on variant interfaces instantiated with value types.

To be type-safe, covariant type parameters can appear only as return types or read-only properties (for example, they can be result types, as in the GetEnumerator method and Current property above); contravariant type parameters can appear only as parameter or write-only properties (argument types, for instance, as in the Compare and CompareTo methods above).

Co- and contravariance are interesting features that eliminate certain inflexibilities when working with generic interfaces and delegates. Having some basic knowledge about these features can be very helpful when writing code that works with generics in Visual Studio 2010. ■

BINYAM KELILE *is a software design engineer in Test with the Microsoft Managed Language Team. During the VS 2008 release, he worked on many of the language features, including LINQ Queries and Lexical Closure. For the upcoming Visual Studio release, he worked on co- and contravariance features. You can reach him at binyamk@microsoft.com.*

THANKS to the following technical experts for reviewing this article: *Beth Massi, Lucian Wischik*

ESRI® Developer Network

Integrate Mapping and GIS into Your Applications

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

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

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

ESRI®

Model Validation & Metadata in ASP.NET MVC 2

One of the new features added to the ASP.NET MVC 2 release is the ability to validate user input on both the server and client. All you need to do is give the framework some information about the data you need validated, and the framework will take care of the hard work and details.

This feature is a tremendous boon for those of us who wrote custom validation code and custom model binders to perform simple model validation with ASP.NET MVC 1.0. In this article, I'll look at the built-in validation support in ASP.NET MVC 2.

Before I discuss the new capabilities, however, I'm going to revisit the old methodology. The validation features in ASP.NET WebForms have served me well for many years. I think it's useful to review them to understand what an ideal validation framework provides.

Controlling Validation

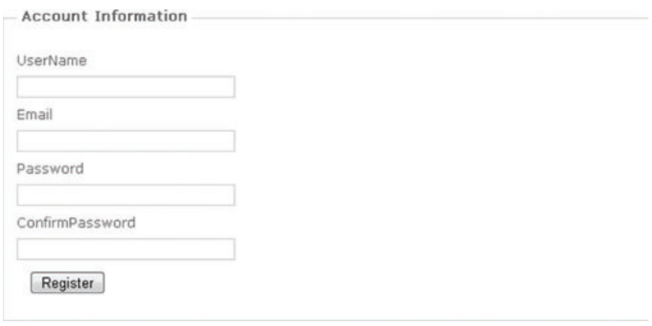
If you've ever used ASP.NET WebForms, you know it's relatively easy to add validation logic to a WebForm. You express the validation rules using controls. For example, if you wanted to make sure the user enters some text into a TextBox control, you just add a RequiredFieldValidator control pointing to the TextBox, like this:

```
<form id="form1" runat="server">
  <asp:TextBox runat="server" ID="_userName" />
  <asp:RequiredFieldValidator runat="server" ControlToValidate="_
userName"
                              ErrorMessage="Please enter a username" />
  <asp:Button runat="server" ID="_submit" Text="Submit" />
</form>
```

If you've ever used ASP.NET WebForms, you know it's relatively easy to add validation logic to a WebForm.

The RequiredFieldValidator encapsulates both client-side and server-side logic to ensure that the user provides a user name. To provide client-side validation, the control emits JavaScript into the client's browser, and this script ensures that the user satisfies all the validation rules before posting the form back to the server.

Think about what these WebForm validation controls offer—they're incredibly powerful!


Account Information

UserName

Email

Password

ConfirmPassword

Register

Figure 1 **Register Information**

- You can declaratively express validation rules for a page in a single location.
- Client validation prevents a round trip to the server if the user doesn't fulfill the validation rules.
- Server validation prevents a malicious user from circumventing the client script.
- The server and client validation logic stay in sync without becoming a maintenance problem.

But in ASP.NET MVC, you can't use these validation controls and remain faithful to the spirit of the MVC design pattern. Fortunately, with version 2 of the framework, there is something even better.

Controls vs. Models

You can think of a WebForm control, like the TextBox, as a simple container for user data. You can populate the control with an initial value and display that value to the user, and you can retrieve any value the user enters or edits by inspecting the control after a postback. When using the MVC design pattern, the M (model) plays this same role as a data container. You populate a model with information you need to deliver to a user, and it will carry back updated values into your application. Thus, the model is an ideal place to express validation rules and constraints.

Here's an example that comes out-of-the-box. If you create a new ASP.NET MVC 2 application, one of the controllers you'll find in the new project is the AccountController. It's responsible for

This column is based on a prerelease version of ASP.NET MVC 2.
All information is subject to change.

Code download available at code.msdn.microsoft.com/mag201003ASPNET.

Figure 2 **The RegisterModel Class with Validation Attributes**

```
[PropertiesMustMatch("Password", "ConfirmPassword",
  ErrorMessage = "The password and confirmation password do not match.")]
public class RegisterModel
{
  [Required]
  public string UserName { get; set; }

  [Required]
  public string Email { get; set; }

  [Required]
  [ValidatePasswordLength]
  public string Password { get; set; }

  [Required]
  public string ConfirmPassword { get; set; }
}
```

handling new user registration requests, as well as log-in and change-password requests. Each of these actions uses a dedicated model object. You can find these models in the AccountModels.cs file in the Models folder. For example, the RegisterModel class, without validation rules, looks like this:

```
public class RegisterModel
{
  public string UserName { get; set; }
  public string Email { get; set; }
  public string Password { get; set; }
  public string ConfirmPassword { get; set; }
}
```

Server validation prevents a malicious user from circumventing the client script.

The Register action of the AccountController takes an instance of this RegisterModel class as a parameter:

```
[HttpPost]
public ActionResult Register(RegisterModel model)
{
    // ...
}
```

If the model is valid, the Register action forwards the model information to a service that can create a new user.

The RegisterModel model is a great example of a *view-specific model*, or view model. It's not a model designed to work with a specific database table, Web service call or business object. Instead, it's designed to work with a specific view (the Register.aspx view, part of which is shown in **Figure 1**). Each property on the model maps to an input control in the view. I encourage you to use view models, as they simplify many scenarios in MVC development, including validation.

Of Models and Metadata

When the user enters account information in the Register view, the MVC framework ensures that the user provides a UserName and Email. The framework also ensures that the Password and ConfirmPassword strings match, and that the password is at least six characters long. How does it do all this? By inspecting and acting on metadata attached to the RegisterModel class. **Figure 2** shows the RegisterModel class with its validation attributes showing.

When the user submits the Register view, the default model binder in ASP.NET MVC will try to build a new instance of the RegisterModel class to pass as the parameter to the AccountController's Register action. The model binder retrieves information in the current request to populate the RegisterModel object. For example, it can automatically find the POST value of an HTML input control named UserName, and use that value to populate the UserName property of RegisterModel. This behavior has been in ASP.NET MVC since version 1.0, so it won't be new if you've already used the framework.

What is new in version 2 is how the default model binder will also ask a *metadata provider* if there is any metadata available for the RegisterModel object. This process ultimately produces a ModelMetaData derived object whose purpose is to describe not only the validation rules associated with the model, but also information related to the display of the model in a view. ASP.NET team member Brad Wilson has an in-depth series of posts on how this model metadata can influence the display of a model through templates. The first post in the series is at bradwilson.typepad.com/blog/2009/10/aspnet-mvc-2-templates-part-1-introduction.html.

Once the model binder has a ModelMetaData object associated with the model, it can use the validation metadata inside to validate the model object. By default, ASP.NET MVC uses metadata from data annotation attributes like [Required]. Of course, ASP.NET MVC is pluggable and extensible, so if you want to devise a different source for model metadata, you can implement your own metadata provider. Ben Scheirman has some great information on this topic in an article titled "Customizing ASP.NET MVC 2—Metadata and Validation," available at dotnetslackers.com/articles/aspnet/customizing-asp-net-mvc-2-metadata-and-validation.aspx.

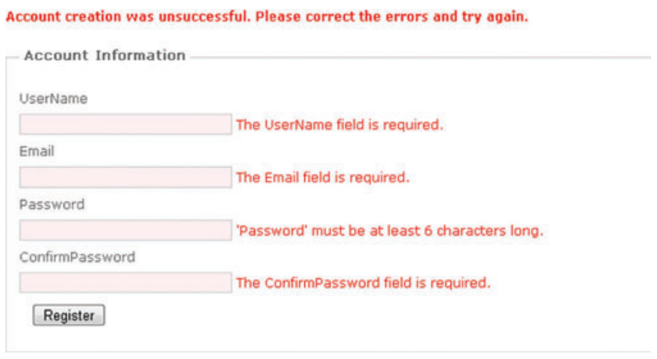
Data Annotations

As a brief aside, you can build your own validation attributes, as we'll see later, but [Required] is one of a number of standard validation attributes that live inside the System.ComponentModel.DataAnnotations assembly. **Figure 3** shows a complete list of validation attributes from the annotations assembly.

These data annotation attributes are quickly becoming pervasive across the Microsoft .NET Framework. Not only can you use these attributes in an ASP.NET MVC application, but ASP.NET Dynamic Data, Silverlight and Silverlight RIA services understand them as well.

Figure 3 **Validation Attributes from the Annotations Assembly**

Attribute	Description
StringLength	Specifies the maximum length of the string allowed in the data field.
Required	Specifies that a data field value is required.
RegularExpression	Specifies that a data field value must match the specified regular expression.
Range	Specifies the numeric range constraints for the value of a data field.
DataType	Specifies the name of an additional type to associate with a data field (one of the DataType enumerated values, like EmailAddress, Url or Password).


Account creation was unsuccessful. Please correct the errors and try again.

Account Information

UserName

The UserName field is required.

Email

The Email field is required.

Password

'Password' must be at least 6 characters long.

ConfirmPassword

The ConfirmPassword field is required.

Figure 4 **Validation Fail**

Viewing Validations

With the validation metadata in place, errors will automatically appear in a view when the user enters incorrect data. **Figure 4** shows what the Register view looks like when a user hits Register without supplying any information.

The display in **Figure 4** was built using some of the new HTML helpers in ASP.NET MVC 2, including the ValidationMessageFor helper. ValidationMessageFor controls the placement of a validation message when validation fails for a particular data field. **Figure 5** shows an excerpt from Register.aspx demonstrating how to use the ValidationMessageFor and ValidationSummary helpers.

Custom Validations

Not all of the validation attributes on the RegisterModel class are attributes from Microsoft's data annotations assembly. The [PropertiesMustMatch] and [ValidatePasswordLength] are custom attributes you'll find defined in the same AccountModel.cs file that holds the RegisterModel class. There is no need to worry about custom metadata providers or metadata classes if you just want to provide a custom validation rule. All you need to do is derive from the abstract class ValidationAttribute and provide an implementation for the IsValid method. The implementation of the ValidatePasswordLength attribute is shown in **Figure 6**.

The other attribute, PropertiesMustMatch, is a great example of a validation attribute you can apply at the class level to perform cross-property validations.

Client Validation

The RegisterModel validation we've looked at so far all takes place on the server. Fortunately, it's easy to enable validation on the client, too. I try to use client validation whenever possible because it can give a user quick feedback while offloading some work from my server. The server-side logic needs to stay in place, however, in case someone doesn't have scripting enabled in a browser (or is intentionally trying to send bad data to the server).

Enabling client validation is a two-step process. Step 1 is making sure the view includes the proper validation scripts. All the scripts you need reside in the Scripts folder of a new MVC application. The MicrosoftAjax.js script is the core of the Microsoft AJAX libraries and is the first script you'll need to include. The second script is MicrosoftMvcValidation.js. I generally add a ContentPlaceHolder to my MVC application's master page to hold scripts, as shown here:

```
<head runat="server">
    <title><asp:ContentPlaceHolder ID="TitleContent" runat=
"server" /></title>
    <link href="../../Content/Site.css" rel="stylesheet" type=
"text/css" />

    <asp:ContentPlaceHolder ID="Scripts" runat="server">

    </asp:ContentPlaceHolder>

</head>
```

A view can then include the scripts it needs using a Content control. The code below would ensure the validation scripts are present:

```
<asp:Content ContentPlaceHolderID="Scripts" runat="server">
    <script src="../../Scripts/MicrosoftAjax.js"
            type="text/javascript"></script>
    <script src="../../Scripts/MicrosoftMvcValidation.js"
            type="text/javascript"></script>
</asp:Content>
```

When using the MVC design pattern, the M (model) plays this same role as a data container.

The second step in using client-side validation is to invoke the EnableClientValidation HTML helper method inside the view where you need validation support. Make sure to invoke this method before using the BeginForm HTML helper, as shown below:

```
<%
   Html.EnableClientValidation();
   using (Html.BeginForm())
    {
 %>

 <!-- the rest of the form ... -->

 <% } %>
```

Note that the client-side validation logic only works with the built-in validation attributes. For the Register view, this means the client-side validation will ensure the required fields are present, but will not know how to validate the password length or confirm that the two password fields match. Fortunately, it's easy to add custom JavaScript validation logic that plugs in to the ASP.NET MVC JavaScript validation framework. Phil Haack has details on his blog entry, "ASP.NET MVC 2 Custom Validation," located at haacked.com/archive/2009/11/19/aspnetmvc2-custom-validation.aspx.

Figure 5 **How to Use New HTML Helpers**

```
<% using (Html.BeginForm()) { %>
    <%= Html.ValidationSummary(true, "Account creation was unsuccessful. " +
    "Please correct the errors and try again.") %>
    <div>
        <fieldset>
            <legend>Account Information</legend>

            <div class="editor-label">
                <%= Html.LabelFor(m => m.UserName) %>
            </div>
            <div class="editor-field">
                <%= Html.TextBoxFor(m => m.UserName) %>
                <%= Html.ValidationMessageFor(m => m.UserName) %>
            </div>
```

Figure 6 **Implementation of the ValidatePasswordLength Attribute**

```
[AttributeUsage(AttributeTargets.Field |
                AttributeTargets.Property,
                AllowMultiple = false,
                Inherited = true)]
public sealed class ValidatePasswordLengthAttribute
    : ValidationAttribute
{
    private const string _defaultErrorMessage =
        "'{0}' must be at least {1} characters long.";

    private readonly int _minCharacters =
        Membership.Provider.MinRequiredPasswordLength;

    public ValidatePasswordLengthAttribute()
        : base(_defaultErrorMessage)
    {
    }

    public override string FormatErrorMessage(string name)
    {
        return String.Format(CultureInfo.CurrentUICulture,
            ErrorMessageString,
            name, _minCharacters);
    }

    public override bool IsValid(object value)
    {
        string valueAsString = value as string;
        return (valueAsString != null &&
            valueAsString.Length >= _minCharacters);
    }
}
```

Wrapping up, you can see that built-in support for common validation scenarios is a huge new addition for ASP.NET MVC 2. Not only are the validation rules easy to add via attributes on a model object, but the validation features themselves are flexible and easy to extend. Start taking advantage of these features to save time and lines of code with your next ASP.NET MVC application. ■

There is no need to worry about custom metadata providers or metadata classes if you just want to provide a custom validation rule.

K. SCOTT ALLEN *is a member of the Pluralsight technical staff and the founder of OdeToCode. You can reach Allen at scott@OdeToCode.com, read his blog at odetocode.com/blogs/scott or follow him on Twitter at twitter.com/OdeToCode.*

THANKS to the following technical expert for reviewing this article: *Brad Wilson*

InishTech Software Licensing and Protection Services

LICENSE MANAGE PROTECT

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

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

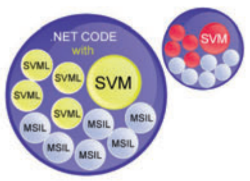
License your Products

Configure products, features and business models independent of code.

Manage your Customers

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

Protect your Hard Work



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

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

Add a Security Bug Bar to Microsoft Team Foundation Server 2010

One of the most contentious tasks a software development team faces during the course of its products' lifecycles is triaging bugs. Deciding the relative level of importance of any given bug—and consequently determining the chance that that bug might not be fixed at all in time for release—is a serious matter to everyone involved in the product's development.

Programmers, testers, architects and program managers all have different viewpoints and base their individual triage decisions on disparate factors such as:

- How much code would have to be regression-tested once the fix is made.
- How close to release the project is.
- How many users would be affected by the change.
- Whether the bug is blocking other issues from being tested or fixed.

I will admit that these are all important factors to consider when triaging functional bugs in product features. However, none of these factors should play any role in determining whether to fix security bugs—that is, bugs that could potentially lead to security vulnerabilities in the product. Classification of security bugs must be objective and consistent. It doesn't make any difference to an attacker that you found a vulnerability only a week before your code-complete milestone; he'll exploit it just the same.

This column describes the objective security bug classification system—the "bug bar"—used by Microsoft internal product and online services teams, which is required by the Security Development Lifecycle (SDL). It also shows how you can incorporate this classification system into your own development environment using Microsoft Team Foundation Server 2010.

DREAD

Before I discuss the bug bar as it exists inside Microsoft today, it's worth describing an earlier Microsoft initiative to classify security bugs: DREAD. DREAD is a mnemonic that stands for:

- Damage Potential
- Reproducibility
- Exploitability
- Affected Users
- Discoverability

Anyone filing a new security bug would assign each of the DREAD parameters a value from 1 to 10, with 10 being the most severe and 1 the least. The values were then averaged to form an overall DREAD rating. For example, say a developer called Doug discovers a blind SQL injection vulnerability in the administration portal page for his team's new Web application. Doug might classify the vulnerability as shown in **Figure 1**.

Figure 1 **A Developer's Security Vulnerability Classification**

DREAD Parameter	Rating	Rationale
Damage Potential	5	An attacker could read and alter data in the product database.
Reproducibility	10	Can reproduce every time.
Exploitability	2	Requires expert knowledge and large time investment.
Affected Users	1	Only affects small subset of user base.
Discoverability	1	Affected page not linked from any user pages.
Overall Rating	3.8	

The classification shown in **Figure 1** seems fairly straightforward and effective. But consider that Doug's tester colleague, Tina, might see the exact same vulnerability in a completely different way, as shown in **Figure 2**.

Figure 2 **The Same Bug as Classified by a Tester**

DREAD Parameter	Rating	Rationale
Damage Potential	10	An attacker could read and alter data in the product database.
Reproducibility	10	Can reproduce every time.
Exploitability	10	Easily exploitable by automated tools found on the Internet.
Affected Users	10	Affects critical administrative users.
Discoverability	10	Affected page "admin.aspx" easily guessed by an attacker.
Overall Rating	10.0	

The Microsoft security bug bar classifies vulnerabilities based on their effects.

Because Tina is aware that there are tools to automate blind SQL injection attacks, she rated Exploitability as 10, whereas Doug saw it as a difficult manual attack and rated the Exploitability as 2. Doug gave Affected Users a 1,because it would only affect a very small portion of the system's users, but Tina rated it as 10 because the users affected would have administrative rights. Perhaps the parameter of most concern is Damage Potential: both Doug and Tina gave the exact same rationale but scored it with different values!

The question we need to ask at this point is not, "Which team member's DREAD rating is better?" but rather, "How can we rely on a system that produces such subjective, variable results?" If Tina had been the first person to find the bug, it certainly would have been fixed before release; but if Doug found it first, there's a good chance it would have been deferred and the application released with the vulnerability. The SDL team concluded that we can't rely on such a system, and

Figure 3 **Sample Security Bug Bar**

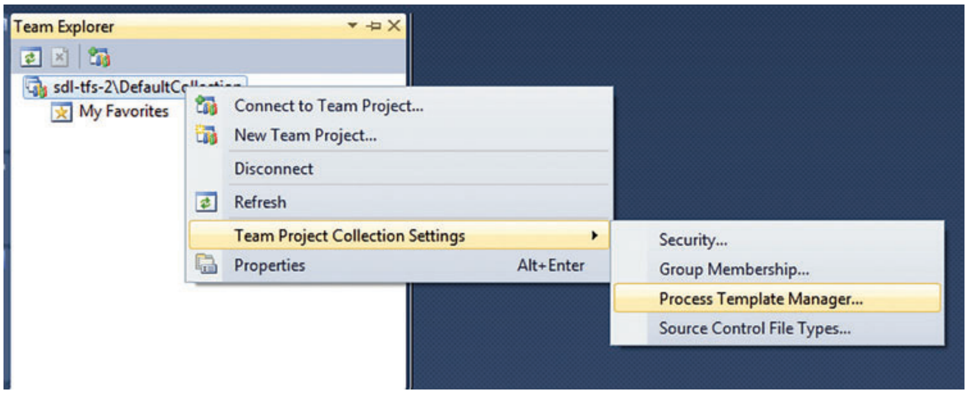
STRIDE Value	Client/ Server	Scope	Severity
Spoofing	Client	Ability for attacker to present a UI that is different from but visually identical to the UI that users *must rely on to make valid trust decisions in a default/common scenario*. A trust decision is defined as any time the user takes an action believing some information is being presented by a particular entity—either the system or some specific local or remote source.	Important
		Ability for attacker to present a UI that is different from but visually identical to the UI that users *are accustomed to trust in a specific scenario*. "Accustomed to trust" is defined as anything a user is commonly familiar with based on normal interaction with the OS or application but does not typically think of as a "trust decision."	Moderate
		Ability for attacker to present a UI that is different from but visually identical to the UI *that is a single part of a bigger attack scenario*.	Low
	Server	Computer connecting to server is able to masquerade as a different user or computer of his or her choice *using a protocol* that is designed and marketed to provide strong authentication.	Important
		Client user or computer is able to masquerade as a different, random user or computer using a protocol that is designed and marketed to provide strong authentication.	Moderate
Tampering/ Repudiation	Client	Permanent modification of any user data or data used to make trust decisions in a common or default scenario that persists after restarting the OS/application.	Important
		Temporary modification of any data that does not persist after restarting the OS/application.	Low
	Server	Permanent modification of any user data or data used to make trust decisions *in a common or default scenario* that persists after restarting the OS/application.	Important
		Permanent modification of any user data or data used to make trust decisions *in a specific scenario* that persists after restarting the OS/application.	Moderate
		Temporary modification of data *in a common or default scenario* that does not persist after restarting the OS/application.	Moderate
		Temporary modification of data *in a specific scenario* that does not persist after restarting the OS/application.	Low
Information Disclosure	Client	Cases where the attacker can locate and read information on the system, including system information that was not intended or designed to be exposed.	Important
		Cases where the attacker can read information on the system *from known locations*, including system information that was not intended or designed to be exposed.	Moderate
		Any untargeted information disclosure (that is, disclosure of random data).	Low
	Server	Cases where the attacker can locate and read information *from anywhere* on the system, including system information that was not intended or designed to be exposed.	Important
		Cases where the attacker can easily read information on the system *from known locations*, including system information that was not intended or designed to be exposed.	Moderate
		Any untargeted information disclosure (for example, disclosure of random data) including runtime data.	Low
Denial of Service	Client	"System corruption DoS": Requires reinstallation of system and/or components.	Important
		"Permanent DoS": Requires cold reboot or causes Blue Screen/Bug Check.	Moderate
		"Temporary DoS": Requires restart of application.	Low
	Server	Anonymous, must be "easy to exploit" by sending a small amount of data or be otherwise quickly induced.	Important
		Anonymous, temporary DoS without amplification in a default/common install.	Moderate
		Authenticated, permanent DoS.	Moderate
		Authenticated, temporary DoS with amplification in a default/common install.	Moderate
Elevation of Privilege	Client	Remote user, the ability to either execute arbitrary code *or* to obtain more privilege than intended.	Critical
		Remote user, execution of arbitrary code *with* extensive user action.	Important
		Local, low-privilege user can elevate himself to another user, administrator or local system.	Important
	Server	Remote anonymous user, the ability to either execute arbitrary code *or* to obtain more privilege than intended.	Critical
		Remote authenticated user, the ability to either execute arbitrary code *or* to obtain more privilege than intended.	Important
		Local authenticated user, the ability to either execute arbitrary code *or* to obtain more privilege than intended.	Important

consequently Microsoft developed a more consistent approach: the security bug bar.

The Microsoft Security Bug Bar

The Microsoft security bug bar classifies vulnerabilities based on their effects. The person filing the bug starts by assigning the bug a security effect value from a list of STRIDE values. STRIDE is another mnemonic, in this case used to categorize threats. Unlike DREAD, STRIDE is still very much used by the SDL and is a core component of several SDL tools, including the SDL Threat Modeling Tool. The STRIDE values are:

- Spoofing
- Tampering
- Repudiation
- Information Disclosure
- Denial of Service (DoS)
- Elevation of Privilege (EoP)



Team Explorer
sdl-tfs-2\DefaultCollection
My Favorites
Connect to Team Project...
New Team Project...
Disconnect
Refresh
Team Project Collection Settings
Properties Alt+Enter
Security...
Group Membership...
Process Template Manager...
Source Control File Types...

Figure 4 **The Process Template Manager for TFS 2010 Server**

Unlike DREAD, STRIDE is still very much used by the SDL and is a core component of several SDL tools.

However, the broad STRIDE threat category alone isn't sufficient to classify and triage a bug. In most cases, we need to know whether the bug affects client-side code or server-side code. For example, a DoS attack that takes out a single targeted user would not be considered as severe as one that takes out an entire server. We also need to know some specific scope information for the bug, depending on the STRIDE and client/server classification.

Continuing the DoS vulnerability example, we need to know who can execute the attack (for example, what privilege level) and how long the effects will last. A vulnerability exploitable by an anonymous user is worse than one exploitable only by an authenticated user, and a vulnerability that locks up the affected server until someone physically reboots it is worse than one that just makes it unavailable for a few seconds.

Armed with the primary STRIDE classification plus the additional scope characteristics, the person triaging the bug can now use this information to look up the bug's severity on the bug bar. **Figure 3** shows a sample security bug bar, as published in the Security Development Lifecycle Process Guidance document version 4.1a. The bug bar defines four levels of severity: Critical, Important, Moderate and Low.

Note that in order for this system to work, the bug-tracking database you're using must have a field for STRIDE security effect. This makes it easy to distinguish security bugs from functional bugs and also to determine the correct bug bar classification. Tracking your bugs' security effects is so important that there is actually a separate SDL requirement to do this work. Luckily, this is fairly easy in most cases. If you're using Team Foundation Server (TFS), the next section will show how to add security effect fields and bug bar ratings to Bug work items in your Team Projects.

Adding Bug Bar Functionality to Team Foundation Server

In order to add a bug bar to a TFS Team Project, you need to make a change to the underlying process template that the project was created from. For the purposes of this article, we will assume the project was created from the MSF-Agile for Software Development version 5.0 (beta) template that ships with the TFS 2010 beta 2. However, if you primarily use a different process template, such as the MSF for CMMI Process Improvement template, or any custom third-party template, the techniques used to edit these templates would be the same.

Figure 5 **Adding Fields for Security Effect, Security Effect Scope and Bug Bar Severity**

```
<FIELD
reportable="dimension"
type="String"
name="Effect"
refname="MSDN.SDL.Security.Effect">
<HELPTEXT>The effect of the security bug</HELPTEXT>
</FIELD>
<FIELD
reportable="dimension"
type="String"
name="EffectScope"
refname="MSDN.SDL.Security.Effect.Scope">
<HELPTEXT>The scope of the effect of the security bug. This value is used
to determine the default bug bar severity</HELPTEXT>
</FIELD>
<FIELD
reportable="dimension"
type="String"
name="BugBarSeverity"
refname="MSDN.SDL.Security.Severity.BugBar">
<HELPTEXT>The suggested severity of the bug as determined by the security
bug bar</HELPTEXT>
</FIELD>
```

Figure 6 **Adding Allowed Values for Effect**

```
<FIELD
reportable="dimension"
type="String"
name="Effect"
refname="MSDN.SDL.Security.Effect">
<HELPTEXT>The effect of the security bug</HELPTEXT>
<ALLOWEDVALUES>
<LISTITEM value="Not a Security Bug" />
<LISTITEM value="Spoofing" />
<LISTITEM value="Tampering" />
<LISTITEM value="Repudiation" />
<LISTITEM value="Information Disclosure" />
<LISTITEM value="Denial of Service" />
<LISTITEM value="Elevation of Privilege" />
<LISTITEM value="Attack Surface Reduction" />
</ALLOWEDVALUES>
<DEFAULT from="value" value="Not a Security Bug" />
</FIELD>
```

Start by opening the Process Template Manager for the TFS 2010 server you want to work on. You can do this by bringing up the context menu for the server in the Team Explorer window, then navigating to Team Project Collection Settings | Process Template Manager, as shown in **Figure 4**.

In the Process Template Manager, choose the MSF for Agile Software Development v5.0 and click the Download button to bring the template source files down locally. Save them to a folder of your choice.

Once the download is complete, close the Process Template Manager. We want to add the bug bar to the Bug work item type, so open the folder to which you just downloaded the MSF-Agile template, then open the file \WorkItem Tracking\TypeDefinitions\Bug.xml in the XML editor of your choice.

The first task is to add fields for Security Effect, Security Effect Scope and Bug Bar Severity. (Bug Bar Severity should remain distinct from the inherent Severity field for reasons I'll explain later.) Under the witd:WITD/WORKITEMTYPE/FIELDS element, add the block of XML shown in **Figure 5**.

This code defines the three new fields, including their names, types and help text; but there's still no logic implemented. Start adding logic by adding allowed-value constraints that lock the possible values for the field.

One of the most contentious tasks a software development team faces during the course of its products' lifecycles is triaging bugs.

For Effect, the allowed values are each of the STRIDE values: Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service and Elevation of Privilege. It's also a good idea to add Attack Surface Reduction as an allowed value for cases that may not be vulnerabilities in and of themselves, but are good opportunities to reduce potential future vulnerabilities. Finally, add Not a Security Bug as an allowed value for cases where the bug is just a normal functional bug with no security implications (see **Figure 6**).

For EffectScope, summarize each of the possible scope items in the bug bar and add these summaries as allowed values for the EffectScope field (see **Figure 7**).

We also want to further restrict the allowed value of EffectScope depending on the current value of Effect. If Effect is currently set to Spoofing, only the spoofing-related EffectScope values should be valid. If Effect is set to Tampering, only the tampering-related EffectScope values should be valid, and so on. We can accomplish this by adding WHEN clause elements to the FIELD definition (see **Figure 8**).

Now it's time to implement the allowed values logic for the BugBarSeverity field. The logic for BugBarSeverity is slightly different from the Effect logic in that we don't want the user to be able to directly set the value of the BugBarSeverity field. The whole point of implementing a bug bar like this is that the severity should reflect the characteristics of the vulnerability. If a user could just set the severity to any value he or she wanted, it would completely defeat the purpose.

Tracking your bugs' security effects is so important that there is actually a separate SDL requirement to do this work.

Instead of creating a list of allowed values for BugBarSeverity, we will use WHEN fields to copy the appropriate value into the BugBarSeverity field, as determined by the bug bar we defined earlier, based on the current value of Effect. For example, the bug bar specifies that a spoofed trusted UI in a common or default

Figure 7 **Adding Allowed Values for EffectScope**

```
<FIELD
reportable="dimension"
type="String"
name="EffectScope"
refname="MSDN.SDL.Security.Effect.Scope">
<HELPTEXT>The scope of the effect of the security bug. This value is used to
determine the default bug bar severity</HELPTEXT>
<ALLOWEDVALUES>
<LISTITEM value="Not Applicable" />
<LISTITEM value="Client - Spoofed trusted UI in common/default scenario" />
<LISTITEM value="Client - Spoofed trusted UI in specific other scenario" />
<LISTITEM value="Client - Spoofed UI as part of a larger attack scenario" />
<LISTITEM value="Server - Spoofed specific user or computer over secure
protocol" />
<LISTITEM value="Server - Spoofed random user or computer over secure
protocol" />
<LISTITEM value="Client - Tampered trusted data that persists after restart"
/>
<LISTITEM value="Client - Tampered data that does not persist after restart"
/>
<!-- additional allowed values omitted for brevity -->
</ALLOWEDVALUES>
<DEFAULT from="value" value="Not Applicable" />
</FIELD>
```

Figure 8 **Adding a WHEN Clause to Restrict Allowed EffectScope Values**

```
<FIELD
reportable="dimension"
type="String"
name="EffectScope"
refname="MSDN.SDL.Security.Effect.Scope">
<HELPTEXT>The scope of the effect of the security bug. This value is used to
determine the default bug bar severity</HELPTEXT>
<ALLOWEDVALUES>
<!-- omitted for brevity -->
</ALLOWEDVALUES>
<DEFAULT from="value" value="Not Applicable" />
<WHEN field="MSDN.SDL.Security.Effect" value="Not a Security Bug">
<ALLOWEDVALUES>
<LISTITEM value="Not Applicable" />
</ALLOWEDVALUES>
</WHEN>
<WHEN field="MSDN.SDL.Security.Effect" value="Attack Surface Reduction">
<ALLOWEDVALUES>
<LISTITEM value="Not Applicable" />
</ALLOWEDVALUES>
</WHEN>
<WHEN field="MSDN.SDL.Security.Effect" value="Spoofing">
<ALLOWEDVALUES>
<LISTITEM value="Client - Spoofed trusted UI in common/default scenario" />
<LISTITEM value="Client - Spoofed trusted UI in specific other scenario" />
<LISTITEM value="Client - Spoofed UI as part of a larger attack scenario" />
<LISTITEM value="Server - Spoofed specific user or computer over secure
protocol" />
<LISTITEM value="Server - Spoofed random user or computer over secure
protocol" />
<LISTITEM value="Not Applicable" />
</ALLOWEDVALUES>
</WHEN>

<!-- Additional WHEN elements for the other STRIDE values omitted for
brevity -->

</FIELD>
```

scenario should be treated as an Important bug, so when Effect is "Client – Spoofed trusted UI in common/default scenario," we copy "2 – Important" into BugBarSeverity (see **Figure 9**).

You might be wondering why I've added number prefixes to the values, like "1 – Critical" and "2 – Important," instead of just defining them as "Critical" and "Important." The answer is that TFS automatically alphabetizes lists of allowed values, and without the number prefixes the choices would be displayed out of order and could confuse the user.

Also, the inherent MSF-Agile Severity field (Microsoft.VSTS.Common.Severity) values have the same number prefixes applied to them, so adding the prefixes to the BugBarSeverity fields reinforces the fact that there is a relationship between these two fields.

Speaking of the relationship between Severity and BugBarSeverity, it's time to enforce that relationship in the template. The next step in the process is to constrain the value of the Severity field so that it's at least as severe as the BugBarSeverity field. If the team has a specific business reason to make the actual severity higher than the value determined by the bug bar, that's OK—we just don't want it to work the other way.

To make this work, we use the same ALLOWEDVALUES technique we used to constrain the EffectScope field based on the value of the Effect field (see **Figure 10**).

There is one final change you need to make to the Bug work item: you need to add UI controls for the new fields we've added, as shown in the snippet below. Work item controls are defined in the witd:WITD/WORKITEMTYPE/FORM/Layout section of the document. It's up to you where you place the new fields, but I suggest adding a new "Security" tab to the main TabGroup and adding the fields there:

```
<Tab Label="Security">
<Control Field Name="MSDN.SDL.Security.Effect" Type="FieldControl"
Label="Effect "LabelPosition="Top" />
<Control Field Name="MSDN.SDL.Security.Effect.Scope" Type="FieldControl"
Label="Scope" LabelPosition="Top" />
<Control Field Name="MSDN.SDL.Security.Severity.BugBar"
Type="FieldControl" Label="Bug Bar Rating" ReadOnly="True"
LabelPosition="Top" />
</Tab>
```

You are now finished editing the Bug work item definition. However, before you can use the new bug bar functionality, you have to import the modified process template definition back into Team Foundation Server. You have two choices for how to do this. You can either replace the existing MSF-Agile for Software Development template with the new template, or you can add the new template side-by-side with the previous version.

If you choose to replace the existing template, open the Process Template Manager, select the MSF-Agile for Software Development template and click the Delete button. Note that this is a non-reversible action. You won't be able to get the original template back without reinstalling Team Foundation Server. Once you've deleted the existing MSF-Agile template, click the Upload button and select the folder containing the edited process template. The template will appear in the list as "MSF-Agile for Software Development" just as before, but now any future projects created from this template will have the bug bar functionality.

If you choose to add the new template side-by-side with the existing MSF-Agile template instead of replacing it, you'll need to change the name of your new template so that it doesn't conflict with the existing one. To do this, you need to make one more file edit. In your XML editor of choice, edit the file Process-

A vulnerability exploitable by an anonymous user is worse than one exploitable only by an authenticated user, and a vulnerability that locks up the affected server until someone physically reboots it is worse than one that just makes it unavailable for a few seconds.

Template.xml, which can be found in the folder to which you originally downloaded the template. Change the value of the ProcessTemplate/metadata/name element to something like "MSF for Agile Software

Development plus Bug Bar," save the file and exit. Open the Process Template Manager, click the Upload button and select the folder containing the edited template. The new template will appear in the list with the name you've given it, and any future projects created from this template will include the bug bar.

Figure 9 **Implementing Value Logic for the BugBarSeverity Field**

```
<FIELD
reportable="dimension"
type="String"
name="BugBarSeverity"
refname="MSDN.SDL.Security.Severity.BugBar">
<HELPTEXT>The suggested severity of the bug as determined by the security
bug bar</HELPTEXT>
<WHEN field="MSDN.SDL.Security.Effect.Scope" value="Not Applicable">
<COPY from="value" value="4 - Low"/>
</WHEN>
<WHEN field="MSDN.SDL.Security.Effect.Scope" value="Client - Spoofed
trusted UI in common/default scenario">
<COPY from="value" value="2 - Important"/>
</WHEN>
<WHEN field="MSDN.SDL.Security.Effect.Scope" value="Client - Spoofed
trusted UI in specific other scenario">
<COPY from="value" value="3 - Moderate"/>
</WHEN>
<WHEN field="MSDN.SDL.Security.Effect.Scope" value="Client - Spoofed UI
as part of a larger attack scenario">
<COPY from="value" value="4 - Low"/>
</WHEN>
<WHEN field="MSDN.SDL.Security.Effect.Scope" value="Server - Spoofed
specific user or computer over secure protocol">
<COPY from="value" value="2 - Important"/>
</WHEN>
<WHEN field="MSDN.SDL.Security.Effect.Scope" value="Server - Spoofed
random user or computer over secure protocol">
<COPY from="value" value="3 - Moderate"/>
</WHEN>

<!-- additional WHEN clauses omitted for brevity -->

</FIELD>
```

Figure 10 **Constraining Allowed Values for the Severity Field**

```
<FIELD
name="Severity"
refname="Microsoft.VSTS.Common.Severity"
type="String"
reportable="dimension">
<HELPTEXT>Assessment of the effect of the bug on the project</HELPTEXT>
<ALLOWEDVALUES expanditems="true">
<LISTITEM value="1 - Critical"/>
<LISTITEM value="2 - High"/>
<LISTITEM value="3 - Medium"/>
<LISTITEM value="4 - Low"/>
</ALLOWEDVALUES>
<DEFAULT from="value "value="3 - Medium" />

<WHEN field="MSDN.SDL.Security.Severity.BugBar"value="1 - Critical">
<ALLOWEDVALUES expanditems="true">
<LISTITEM value="1 - Critical"/>
</ALLOWEDVALUES>
</WHEN>
<WHEN field="MSDN.SDL.Security.Severity.BugBar" value="2 - Important">
<ALLOWEDVALUES expanditems="true">
<LISTITEM value="1 - Critical"/>
<LISTITEM value="2 - High"/>
</ALLOWEDVALUES>
</WHEN>
<WHEN field="MSDN.SDL.Security.Severity.BugBar" value="3 - Moderate">
<ALLOWEDVALUES expanditems="true">
<LISTITEM value="1 - Critical"/>
<LISTITEM value="2 - High"/>
<LISTITEM value="3 - Medium"/>
</ALLOWEDVALUES>
</WHEN>
</FIELD>
```

Using the Bug Bar to Determine Exit Criteria

Of course, all the process template functionality in the world still won't help you unless you have an organizational policy to back it up. The standard for internal development teams at Microsoft is that any security bug that falls above the "Low" category of the bug bar must be fixed before release. This standard is never relaxed, no matter how close the product is to release. This approach assures the objectivity of security bug triage, based solely on the possible effect and scope of the bug.

If the team has a specific business reason to make the actual severity higher than the value determined by the bug bar, that's OK—we just don't want it to work the other way.

This is the reason we define a separate BugBarSeverity field instead of just constraining the value of the common Severity field based on EffectScope. From a strict security standpoint, we don't care if the product ships with any severity Critical bugs as long as those bugs have no security effects. All we really care about is whether the bug has a BugBarSeverity higher than "4 – Low." If so, that bug must be fixed before release.

Call to Action

Without a consistent, objective process for triaging security bugs, you will not be able to create secure applications. Using the Microsoft security bug bar is an excellent way to accomplish this, and it's a key component of the Security Development Lifecycle, which has been proven to reduce vulnerabilities in software.

Additionally, if you are using Microsoft Team Foundation Server, you can easily add bug bar functionality to your team projects. This makes it even easier for your team to appropriately classify security bugs, even if the developers are not necessarily security experts.

As a final thought, I'd like to encourage you to download the MSF-Agile + SDL process template for Visual Studio 2010. This process template will be available for free at microsoft.com/sdl shortly after the release of Visual Studio 2010. It incorporates the bug bar described in this article, as well as many other features designed to help you create more secure software. ■

BRYAN SULLIVAN *is a security program manager for the Microsoft Security Development Lifecycle team, where he specializes in Web application and .NET security issues. He is the author of "Ajax Security" (Addison-Wesley, 2007).*

THANKS to the following technical expert for reviewing this article: *Brian Harry*

Testing Silverlight Apps Using Messages

I am a big fan of Silverlight and in this month's column I describe a technique you can use to test Silverlight applications.

Silverlight is a complete Web application framework that was initially released in 2007. The current version, Silverlight 3, was released in July 2009. Visual Studio 2010 provides enhanced support for Silverlight, in particular a fully integrated visual designer that makes designing Silverlight user interfaces a snap.

The best way for you to see where I'm headed in this article is to take a look at the apps themselves. **Figure 1** shows a simple but representative Silverlight application named MicroCalc. You can see that MicroCalc is hosted inside Internet Explorer, though Silverlight applications can also be hosted by other browsers including Firefox, Opera and Safari.

Figure 2 shows a lightweight test harness, which is also a Silverlight application.

In this example, the first test case has been selected. When the button control labeled Run Selected Test was clicked, the Silverlight test harness sent a message containing the selected test case input data to the Silverlight MicroCalc application under test. This test case data consists of instructions to simulate a user typing 2.5 and 3.0 into the input areas of the application, selecting the Multiply operation, and then clicking the Compute button.

The application accepted the test case data and programmatically exercised itself using test code that is instrumented into the application. After a short delay, the test harness sends a second message to the application under test, requesting that the application send a message containing information about the application's state—namely, the value in the result field. The test harness received the resulting message from the application and determined that

Visual Studio 2010 provides enhanced support for Silverlight.

the actual value in the application, 7.5000, matched the expected value in the test case data, and displayed a Pass test case result in the harness comments area.

This article assumes you have basic familiarity with the C# language, but does not assume you have any experience with Silverlight. In the sections of this column that follow, I first describe the Silverlight application under test. I walk you through the details of creating the lightweight Silverlight-based test harness shown in

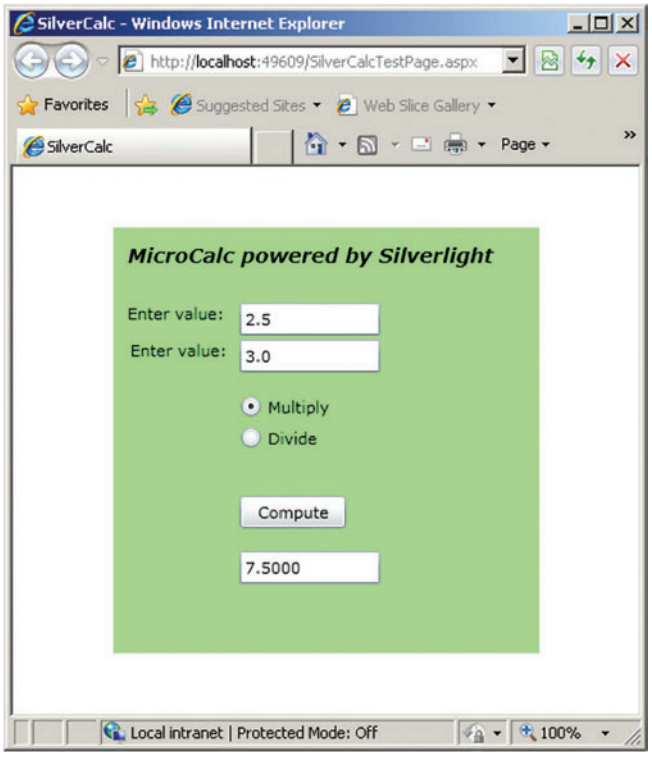

SilverCalc - Windows Internet Explorer
http://localhost:49609/SilverCalcTestPage.aspx
Favorites
Suggested Sites
Web Slice Gallery
SilverCalc
Page
MicroCalc powered by Silverlight
Enter value:
2.5
Enter value:
3.0
Multiply
Divide
Compute
7.5000
Local intranet | Protected Mode: Off
100%

Figure 1 **MicroCalc Silverlight App**

Figure 1, then I explain how to instrument the application. I wrap up by describing alternative testing approaches.

The Application Under Test

Let's take a look at the code for the Silverlight MicroCalc application that is the target of my test automation example. I created MicroCalc using Visual Studio 2010 beta 2. Silverlight 3 is fully integrated into Visual Studio 2010, but the code I present here also works with Visual Studio 2008 with the Silverlight 3 SDK installed separately.

After launching Visual Studio, I clicked on File | New | Project. Note that a Silverlight application is a .NET component that may be hosted in a Web application, rather than a Web application itself. In the New Project dialog, I selected the C# language templates option.

This article discusses a prerelease version of Visual Studio 2010. All information is subject to change.

Code download available at code.msdn.microsoft.com/mag201003TestRun.

Do Wonders

With the same premium tools used by more than 50% of Fortune 100 companies

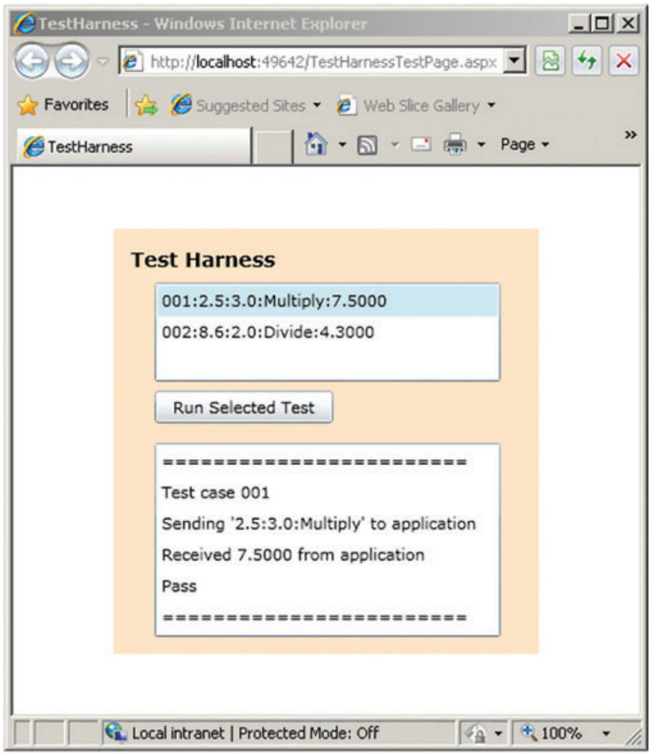
Aspose provides extensive file format processing capabilities for some of the most popular file formats including:

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

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

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

ASPOSE®
The .NET & Java Component Publisher™


TestHarness - Windows Internet Explorer
http://localhost:49642/TestHarnessTestPage.aspx
Favorites
Suggested Sites
Web Slice Gallery
TestHarness
Page
Test Harness
Run Selected Test
Local intranet | Protected Mode: Off
100%

```
001:2.5:3.0:Multiply:7.5000
002:8.6:2.0:Divide:4.3000
```

```
========================
Test case 001
Sending '2.5:3.0:Multiply' to application
Received 7.5000 from application
Pass
========================
```

Figure 2 **Test Harness for MicroCalc**

Silverlight applications can also be created using Visual Basic, and you can even create Silverlight libraries using the new F# language.

I selected the default Microsoft .NET Framework 4 library option and the Silverlight Application template. Silverlight contains a subset of the .NET Framework, so not all parts of the .NET Framework 4 are available to Silverlight applications. After filling in the Name (SilverCalc) and Location (C:\SilverlightTesting) fields, I clicked the OK button. (Note that SilverCalc is the Visual Studio project name, and MicroCalc is the application name.)

By default, a Silverlight application occupies the entire client area in its hosting page.

Visual Studio then prompted me with a New Silverlight Application dialog box that can be confusing to Silverlight beginners. Let's take a closer look at the options in **Figure 3**.

The first entry, "Host the Silverlight application in a new Web site," is checked by default. This instructs Visual Studio to create two different Web pages to host your Silverlight application.

The next entry is the name of the Visual Studio project that contains the two host pages. The project will be added to your Visual Studio solution.

The third entry in the dialog box is a dropdown control with three options: ASP.NET Web Application Project, ASP.NET Web Site, and ASP.NET MVC Web Project. A full discussion of these options is outside the scope of this article, but the bottom line is that the best general purpose option is ASP.NET Web Application Project.

The fourth entry in the dialog box is a dropdown to select the Silverlight version, in this case 3.0. After clicking OK, Visual Studio creates a blank Silverlight application.

I double-clicked on the MainPage.xaml file to load the XAML-based UI definitions into the Visual Studio editor. I modified the default attributes for the top-level Grid control by adding Width and Height attributes and changing the Background color attribute:

```
<Grid x:Name="LayoutRoot"
  Background="PaleGreen"
  Width="300" Height="300">
```

By default, a Silverlight application occupies the entire client area in its hosting page. Here I set the width and height to 300 pixels to make my Silverlight application resemble the default size of a WinForm application. I adjusted the color to make the area occupied by my Silverlight application clearly visible.

Next I used Visual Studio to add the labels, three TextBox controls, two RadioButton controls and a Button control onto my application as shown in **Figure 1**. Visual Studio 2010 has a fully integrated design view so that when I drag a control, such as a TextBox, onto the design surface, the underlying XAML code is automatically generated:

```
<TextBox Width="99" Height="23" Name="textBox1" ... />
```

After placing the labels and controls onto my MicroCalc application, I double-clicked on the Button control to add its event handler to the MainPage.xaml.cs file. In the code editor I typed the following C# code to give MicroCalc its functionality:

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

  double x = double.Parse(textBox1.Text);
  double y = double.Parse(textBox2.Text);
  double result = 0;
  if (radioButton1.IsChecked == true)
    result = x * y;
  else if (radioButton2.IsChecked == true)
    result = x / y;
  textBox3.Text = result.ToString("0.0000");
}
```

I begin by grabbing the values entered as text into the textBox1 and textBox2 controls and converting them to type double. Notice that, to keep my example short, I have omitted the normal error-checking you'd perform in a real application.

Next I determine which RadioButton control has been selected by the user. I must use the fully qualified Boolean expression:

```
if radioButton1.IsChecked == true
```

You might have expected that I'd use the shortcut form:

```
if radioButton1.IsChecked
```

I use the fully qualified form because the IsChecked property is the nullable type bool? rather than plain bool.

After computing the indicated result, I place the result formatted to four decimal places into the textBox3 control.

MicroCalc is now ready to go and I can hit the F5 key to instruct Visual Studio to run the application. By default, Visual Studio will launch Internet Explorer and load the associated .aspx host page that was automatically generated. Visual Studio runs a Silverlight test host page through the built-in Web development server rather than through IIS. In addition to an .aspx test host page, Visual Studio also generates an HTML test page that you can manually load by typing its address into Internet Explorer.

The Test Harness

Now that you've seen the Silverlight application under test, let me describe the test harness.

I decided to use Local Messaging to send messages between the harness and the application. I began by launching a new instance of Visual Studio 2010. Using the same process as described in the previous section, I created a new Silverlight application named TestHarness. As with the MicroCalc application, I edited the top-level Grid control to change its default size to 300x300 pixels, and its background color to Bisque in order to make the Silverlight control stand out clearly. Next I added a Label control, two ListBox controls and a Button control to the harness design surface.

I decided to use Local Messaging to send messages between the harness and the application.

After changing the Content property of the Button control to Run Selected Test, I double-clicked the button to generate its event handler. Before adding the logic code to the handler, I declare a class-scope LocalMessageSender object and test case data in the MainPage.xaml.cs file of the harness so that the harness can send messages to the application under test:

```
public partial class MainPage : UserControl {
  LocalMessageSender lms = null;
  private string[] testCases = new string[] {
    "001:2.5:3.0:Multiply:7.5000",
    "002:8.6:2.0:Divide:4.3000"
  };
...
```

The LocalMessageSender class is contained in the System.Windows.Messaging namespace so I added a reference to it with a using statement at the top of the .cs file so that I don't have to fully qualify the class name. I employ a simple approach for my test case data and use a colon-delimited string with fields for test case ID, first input value, second input value, operation and expected result. Next I add class scope string variables for each test case field:

```
private string caseID;
private string input1;
private string input2;
private string operation;
private string expected;
...
```

These variables aren't technically necessary, but make the test code easier to read and modify.

Now I instantiate a LocalMessageReceiver object into the MainPage constructor so that my test harness can accept messages from the application under test:

```
public MainPage() {
  InitializeComponent();

  try {
    LocalMessageReceiver lmr =
      new LocalMessageReceiver("HarnessReceiver",
        ReceiverNameScope.Global,
        LocalMessageReceiver.AnyDomain);
...
```

The LocalMessageReceiver object constructor accepts three arguments. The first argument is a name to identify the receiver—this will be used by a LocalMessageSender object to specify which receiver to target. The second argument is an Enumeration type that specifies whether the receiver name is scoped to the global domain or to a more restricted domain. The third argument specifies where the receiver will accept messages from, in this case any domain.

Next I wire up an event handler for the receiver, and then fire up the receiver object:

```
lmr.MessageReceived += HarnessMessageReceivedHandler;
lmr.Listen();
...
```

Here I indicate that when the test harness receives a message, control should be transferred to a program-defined method named HarnessMessageReceivedHandler. The Listen method, as you might expect, continuously monitors for incoming messages sent from a LocalMessageSender in the application under test.

Now I instantiate the sender object I declared earlier:

```
lms = new LocalMessageSender(
  "AppReceiver", LocalMessageSender.Global);
lms.SendCompleted += HarnessSendCompletedHandler;
...
```

Notice that the first argument to the sender object is the name of a target receiver object, not an identifying name of the sender. Here my test harness sender will be sending messages only to a receiver named AppReceiver located in the application under

TeeChart
TeeChart Charts
TeeChart Maps
TeeChart Gauges
come and visit us at steema software www.steema.com
Charting for Developers
Charts, Maps and Gauges for Microsoft.NET (WinForms, Pocket, WebForms and Ajax), for Delphi/C++Builder, as native Java, ActiveX or PHP.

test. In other words, receiver objects have names and will accept messages from any sender objects, but sender objects do not have names and will send messages only to a specific receiver.

After instantiating the sender object, I wire up an event handler for the SendCompleted event. Now I can load my test cases and handle any exceptions:

```
      ...
      foreach (string testCase in testCases) {
        listBox1.Items.Add(testCase);
      }
    } // try
    catch (Exception ex) {
      listBox2.Items.Add(ex.Message);
    }
  } // MainPage()
```

I simply iterate through the test case array, adding each test case string to the listBox1 control. If any exception is caught, I just display its text in the listBox2 control used for comments.

At this point I have a sender object in the harness that can send test case input to the application, and a receiver object in the harness that can accept state information from the application. Now I go back to the button1_Click handler method I added earlier. In the handler, I begin by parsing the selected test case:

```
string testCaseData = (string)listBox1.SelectedItem;
string[] tokens = testCaseData.Split(':');
caseID = tokens[0];
input1 = tokens[1];
input2 = tokens[2];
operation = tokens[3];
expected = tokens[4];
...
```

Now I'm ready to send test case input to the Silverlight application under test:

```
string testCaseInput =
  input1 + ":" + input2 + ":" + operation;
listBox2.Items.Add("========================");
listBox2.Items.Add("Test case " + caseID);
listBox2.Items.Add(
  "Sending '" + testCaseInput + "' to application");
lms.SendAsync("data" + ":" + testCaseInput);
...
```

I do not send the test case ID or the expected value to the application because only the harness deals with those values.

I stitch back together just test case input. I do not send the test case ID or the expected value to the application because only the harness deals with those values. After displaying some comments to the listBox2 control, I use the SendAsync method of the LocalMessageSender object to send the test case data. I prepend the string "data" so that the application has a way to identify what type of message is being received.

My button event handler finishes up by pausing for one second in order to give the application time to execute, and then I send a message asking the application for its state information:

```
  System.Threading.Thread.Sleep(1000);
  lms.SendAsync("response");
} // button1_Click
```

Recall that I wired up an event handler for send completion, but in this design I do not need to perform any explicit post-send processing.

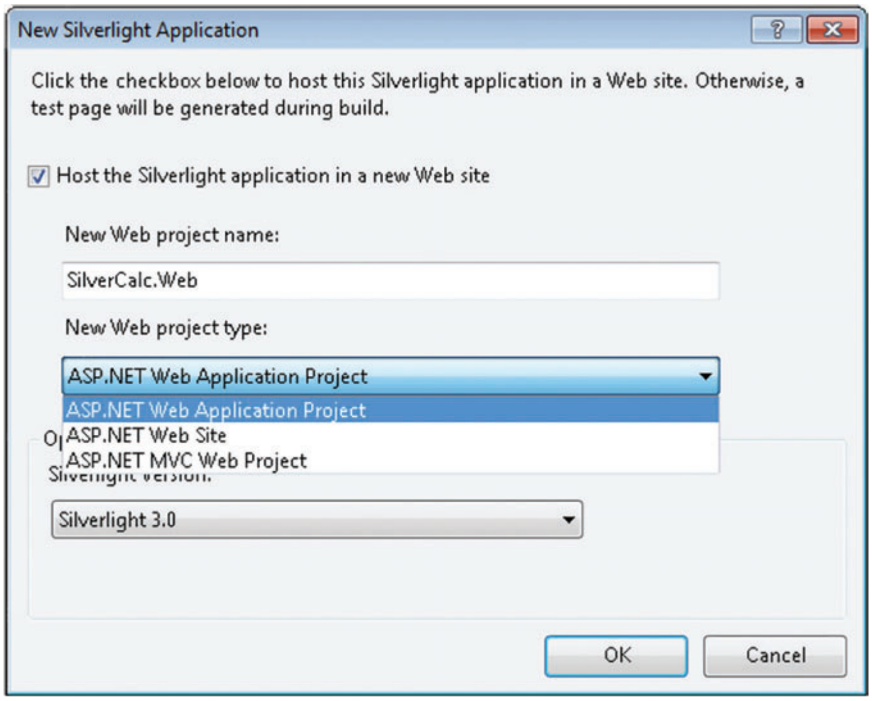
The final part of the harness code deals with the message sent from the Silverlight application to the harness:

```
private void HarnessMessageReceivedHandler(object sender,
  MessageReceivedEventArgs e) {

  string actual = e.Message;
  listBox2.Items.Add(
    "Received " + actual + " from application");
  if (actual == expected)
    listBox2.Items.Add("Pass");
  else
    listBox2.Items.Add("**FAIL**");

  listBox2.Items.Add("========================");
}
```

Here I fetch the message from the application, which is the value in the textBox3 result control, and store that value into a variable named actual. After displaying a comment, I compare the actual value that was sent by the application with the expected value parsed from the test case data to determine and display a test case pass/fail result.

Instrumenting the Silverlight Application

Now let's examine the instrumented code inside the Silverlight application under test. I begin by declaring a class-scope LocalMessageSender object.


New Silverlight Application

Click the checkbox below to host this Silverlight application in a Web site. Otherwise, a test page will be generated during build.

☑ Host the Silverlight application in a new Web site

New Web project name:

SilverCalc.Web

New Web project type:

ASP.NET Web Application Project

ASP.NET Web Application Project
ASP.NET Web Site
ASP.NET MVC Web Project

Silverlight 3.0

OK | Cancel

Figure 3 **New Silverlight Application Dialog Box Options**

This sender will send messages to the test harness:

```
public partial class MainPage : UserControl {
  LocalMessageSender lms = null;
  public MainPage() {
    InitializeComponent();
...
```

Next I instantiate a receiver in the MainPage constructor to accept messages from the test harness, wire up an event handler, and start listening for messages from the harness:

```
try {
  LocalMessageReceiver lmr =
    new LocalMessageReceiver("AppReceiver",
      ReceiverNameScope.Global,
      LocalMessageReceiver.AnyDomain);
  lmr.MessageReceived += AppMessageReceivedHandler;
  lmr.Listen();
...
```

The BeginInvoke method allows you to asynchronously call a method on the Silverlight user interface thread.

As before, note that you assign a name to the receiver object, and that this name corresponds to the first argument to the sender object in the harness. Then I deal with any exceptions:

```
...
  }
  catch (Exception ex) {
    textBox3.Text = ex.Message;
  }
} // MainPage()
```

I display exception messages in the textBox3 control, which is the MicroCalc application result field. This approach is completely ad hoc, but sending the exception message back to the test harness may not be feasible if the messaging code throws the exception. Now I handle messages sent by the test harness:

```
private void AppMessageReceivedHandler(object sender,
  MessageReceivedEventArgs e) {
  string message = e.Message;
  if (message.StartsWith("data")) {
    string[] tokens = message.Split(':');
    string input1 = tokens[1];
    string input2 = tokens[2];
    string operation = tokens[3];
...
```

The test harness sends two types of messages. Test case input data starts with "data" while a request for application state is just "response." I use the StartsWith method to determine if the message received by the application is test case input. If so, I use the Split method to parse the input into variables with descriptive names.

Now the instrumentation uses the test case input to simulate user actions:

```
textBox1.Text = input1;
textBox2.Text = input2;
if (operation == "Multiply")
  radioButton1.IsChecked = true;
else if (operation == "Divide")
  radioButton2.IsChecked = true;
...
```

In general, modifying properties of controls, such as the Text and IsChecked properties in this example, to simulate user input is straightforward. However, simulating events such as button clicks requires a different approach:

```
button1.Dispatcher.BeginInvoke(
  delegate { button1_Click(null,null); });
```

The Dispatcher class is part of the Windows.Threading namespace so I added a using statement referencing that class to the application. The BeginInvoke method allows you to asynchronously call a method on the Silverlight user interface thread. BeginInvoke accepts a delegate, which is a wrapper around a method. Here I use the anonymous delegate feature to simplify my call. BeginInvoke returns a Dispatcher-Operation object, but in this case I can safely ignore that value.

The Dispatcher class also has a CheckAccess method you can use to determine whether BeginIvoke is required (when Check-Access returns false) or whether you can just modify a property (CheckAccess returns true).

I finish my instrumentation by dealing with the message from the test harness that requests application state:

```
...
  }
  else if (message == "response") {
    string actual = textBox3.Text;
    lms.SendAsync(actual);
  }
} // AppMessageReceivedHandler()
```

If the message received is just the string "response," I grab the value in the textBox3 control and send it back to the test harness.

The test system I describe in this article is just one of many approaches you can use and is best suited for 4-4-4 ultra-light test automation. By this I mean a harness that has an expected life of 4 weeks or less, consists of 4 pages or less of code, and requires 4 hours or less to create.

The main advantage of testing using Messages compared to other approaches is that the technique is very simple. The main disadvantage is that the application under test must be heavily instrumented, which may not always be feasible.

The Messages-based approach is an efficient and effective technique.

Two important alternatives to the technique I have presented here are using the HTML Bridge with JavaScript and using the Microsoft UI Automation library. As usual, I'll remind you that no one particular testing approach is best for all situations, but the Messages-based approach I've presented here can be an efficient and effective technique in many software development scenarios. ■

DR. JAMES MCCAFFREY *works for Volt Information Sciences Inc., where he manages technical training for software engineers working at the Microsoft Redmond, Wash., campus. He has worked on several Microsoft products including Internet Explorer, and MSN Search. Dr. McCaffrey is the author of ".NET Test Automation Recipes" (Apress, 2006). He can be reached at jammc@microsoft.com.*

THANKS to the following technical experts for reviewing this article: *Karl Erickson and Nitya Ravi*

REDMOND, WA / MICROSOFT CAMPUS

AUGUST 2–6, 2010

DIRECT FROM

REGISTER NOW

“Great conference overall and nice mix of best practices, tips and new technologies.”

– John Merritt, Syrasoft, LLC

“The conference was a great way to see new and exciting features coming out for the .Net Framework and new technologies out for developers to use with current projects and future work.”

– Mario Rosal, Jr., Tybrin Corp.

THE
MICROSOFT CAMPUS...

VSLive!

Empowering Developers Since 1993

"*I have over 10 years of software development with many different technologies. The sessions were very insightful...*"

– Mickey Manganelli, Veritext

DETAILS AND REGISTRATION AT

VSLIVE.COM

USE PRIORITY CODE VSLMS4

VSLive!

Empowering Developers Since 1993

MARK YOUR CALENDARS!
COMING AUGUST 2–6, 2010

Microsoft's Redmond campus is rolling out the red carpet for VSLive! Reserve your place today on campus and take advantage of this incredible on-site opportunity to get exclusive insights into the latest product updates and upcoming technologies direct from the source! Earn your bragging rights and be the envy of your team when you tell them you're heading to Microsoft for VSLive!

REGISTER TODAY AND GET IN-DEPTH COVERAGE OF:

- **SILVERLIGHT/WPF**
  Silverlight, WPF, XAML, VS10 Xaml designer, Blend, WCF RIA Services
- **WEB**
  Web Forms, ASP.NET MVC, AJAX
- **CLOUD COMPUTING**
  Includes cloud, server and messaging technologies
  Azure, Amazon, AppFabric, REST services, "Dallas", WCF, Windows Workflow
- **SHAREPOINT**
  SharePoint, Office
- **DATA MANAGEMENT**
  SQL Server, BI, reporting, analysis, ADO.NET EF, ANDS, oData, Sync services
- **VISUAL STUDIO 2010/.NET 4**
  Visual Studio features, TFS, languages, parallel extensions

Supported by:

Microsoft msdn Visual Studio Microsoft Visual Studio

1105 MEDIA INC

ATTEND VSLIVE! REDMOND ON THE MICROSOFT CAMPUS

AND GET AN INSIDER'S LOOK AT MICROSOFT'S CORPORATE HEADQUARTERS!

DETAILS AND REGISTRATION AT

VSLIVE.COM

USE PRIORITY CODE VSLMS4

JOIN VSLIVE! AT THE MICROSOFT CONFERENCE CENTER IN REDMOND WA!

Edge Cases

In my last column, I praised Word's auto-correct feature, which automatically converts the "hte" that I typed into the "the" that I really meant. I got a number of responses, saying, "No, Plattski, that's a really bad feature. Sometimes you actually mean "hte," like when you wrote that column, and auto-correct gets in the way. Word shouldn't do that, because sometimes it's wrong."

This is the typical geek worldview. We are trained mathematically, logically. We get hammered into us from middle-school algebra onward that a theorem that's true in 99 cases but false in the 100th case is a false theorem. Bad geek. Throw it away; go find a true one.

That's right for mathematical theorems, but it's way wrong for human users. Word's auto-correct feature doesn't always correct our documents correctly. But making its best guess, which improves as we use it, and having us correct any resulting errors, is a large net profit for the user. (If you don't believe me, try writing with WordPad for a whole week. I guarantee you'll go crazy by Wednesday.) That net result is what most users judge our products on, most of the time.

Unlike a theorem, if your program makes 99 out of 100 users happy, you're probably having a pretty good day. And it's probably more important to make those 99 users happy again tomorrow than it is to figure out how to please that 100th user—especially if what he needs to make him happy would annoy the other 99.

Consider the classic model of editing a document—when you close the program, it asks, "Do you want to save your changes?" How often do you want to discard all your changes? Once in a while, but not often. Not once a day, probably not once a week, I doubt even once a month; especially since the Undo capability in most applications lets you revert to any intermediate state in your editing session.

Yet this uncommon edge case gets equal treatment in the UI with the almost-universal case of keeping the work you just spent all that time on. Not only does it waste the time of almost all users, not only is it difficult for new users to learn, but it places all users in danger of wrongly discarding the work they almost always want to keep—one sneeze while clicking the mouse, or my cat pouncing on the keyboard ("Knock it off, Simba—no, NO, I meant stop that!") and it's gone.

Suppose the dialog box read "Throw away everything you just did?" What dimwit would ask such a silly question after every editing session? But it's exactly the same question, rephrased from the program's viewpoint to the user's.

We can break this anti-pattern, and occasionally we do. Microsoft One Note saves its documents automatically. We can roll back changes with the undo key if we need to, but the program doesn't bother us about it every time. Quicken, the personal finance manager from Intuit, works the same way. It never asks if we want to save a check; the act of entering the check means that we want it. If we change our minds, we'll delete the check. Saving changes is one less thing we have to think about to use these apps successfully. Studies show that users pick up this change from convention very quickly, and they like it.

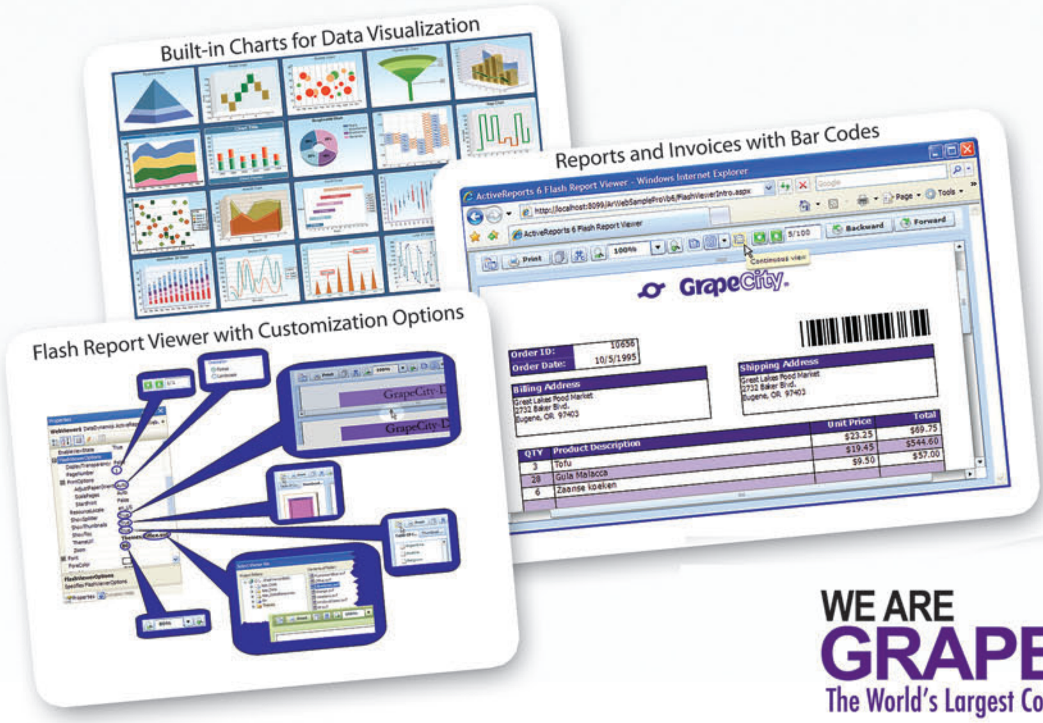
Clearly, some situations exist where this design approach can't apply, where our programs have to handle every case correctly the first time or we fail—air traffic control springs to mind, or cancer chemotherapy. But these life-critical applications have their own, different user interface problems that require specialist attention.

The case in which edge cases need equal treatment in the UI is itself an uncommon edge case. If your program has to deal with one of them, then attack it with my blessing. But for most business and consumer programs, the world is a better place when you handle the main case seamlessly and fix edge cases only as they arise, rather than annoying every user with every edge case that might ever exist. ■

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, 2003). Microsoft named him a Software Legend in 2002. He wonders whether he should tape down two of his daughter's fingers so she learns how to count in octal. You can contact him at rollthunder.com.*

GrapeCity.

WE ARE
REPORTING


Built-in Charts for Data Visualization
Reports and Invoices with Bar Codes
Flash Report Viewer with Customization Options
ActiveReports 6 Flash Report Viewer - Windows Internet Explorer
Print
Backward
Forward
Continuous view
GrapeCity.

Order ID:	10656
Order Date:	10/5/1995

Billing Address
Great Lakes Food Market
2732 Baker Blvd.
Eugene, OR 97403

Shipping Address
Great Lakes Food Market
2732 Baker Blvd.
Eugene, OR 97403

QTY	Product Description	Unit Price	Total
3	Tofu	$23.25	$69.75
28	Gula Malacca	$19.45	$544.60
6	Zaanse koeken	$9.50	$57.00

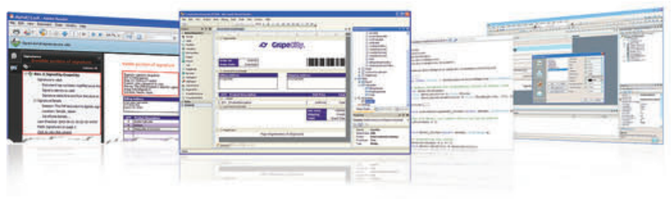
WE ARE
GRAPECITY
The World's Largest Component Vendor

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
- ✔ Expert Technical Support and Extensive Samples and Documentation

PowerTools	SPREAD 5	ACTIVEREPORTS 6	ANALYSIS 1
Deep Functionality	100%	100%	100%
Award Winning	✓	✓	New
Category Leader	✓	✓	✓
Industry's Best Support	✓	✓	✓
Proven Track Record	✓	✓	✓

ACTIVEREPORTS 6



www.FarPointSpread.com & www.DataDynamics.com

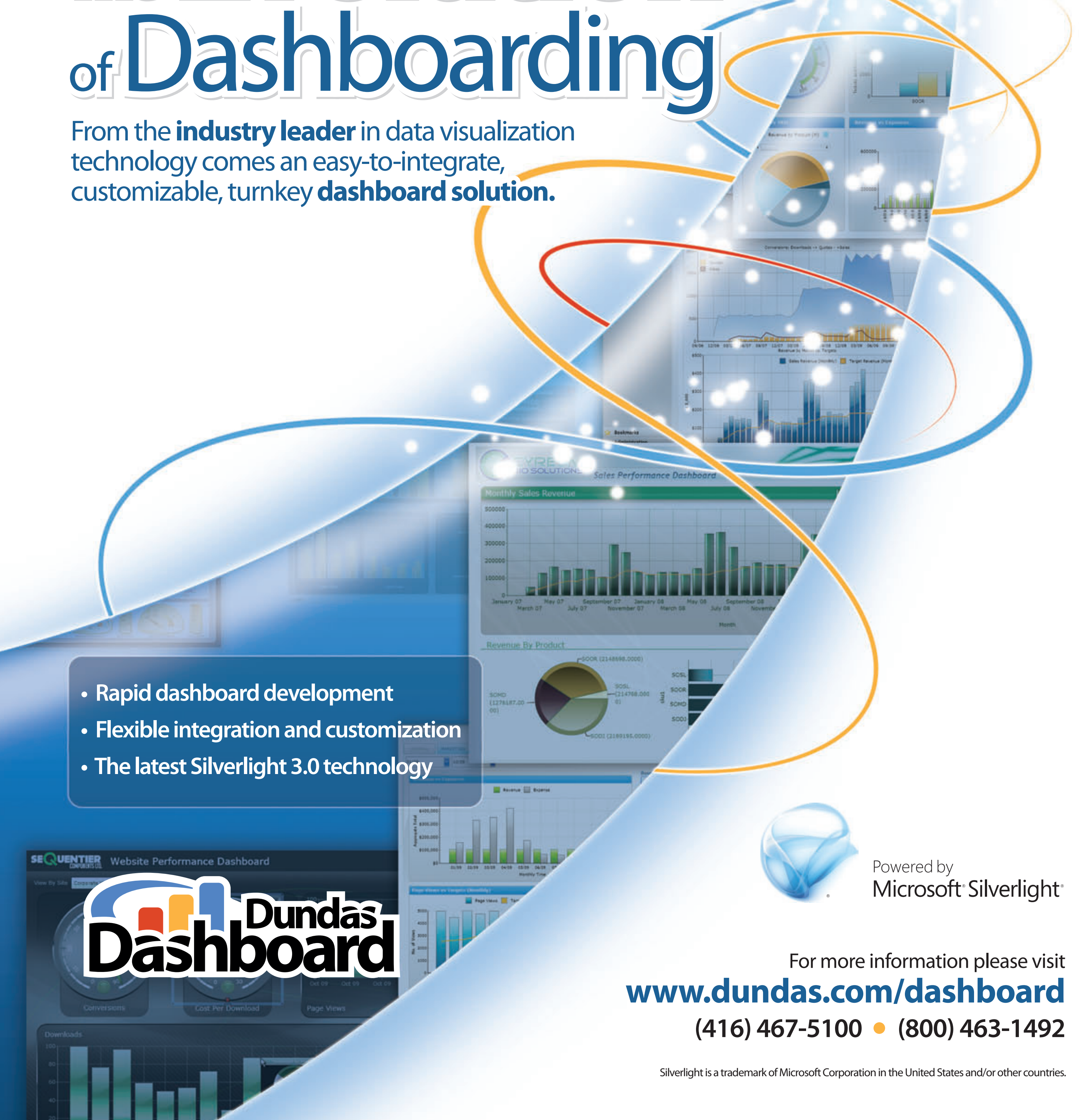
Download your free evaluation today at www.DataDynamics.com

Spreadsheets Reporting Analysis

Dundas
Data Visualization

The Evolution of Dashboarding

From the **industry leader** in data visualization technology comes an easy-to-integrate, customizable, turnkey **dashboard solution.**



- Rapid dashboard development
- Flexible integration and customization
- The latest Silverlight 3.0 technology

Dundas Dashboard

Powered by Microsoft® Silverlight®

For more information please visit
www.dundas.com/dashboard
(416) 467-5100 • (800) 463-1492