THE MICROSOFT JOURNAL FOR DEVELOPERS

JUNE 2011 VOL 26 NO 6

msdn magazine

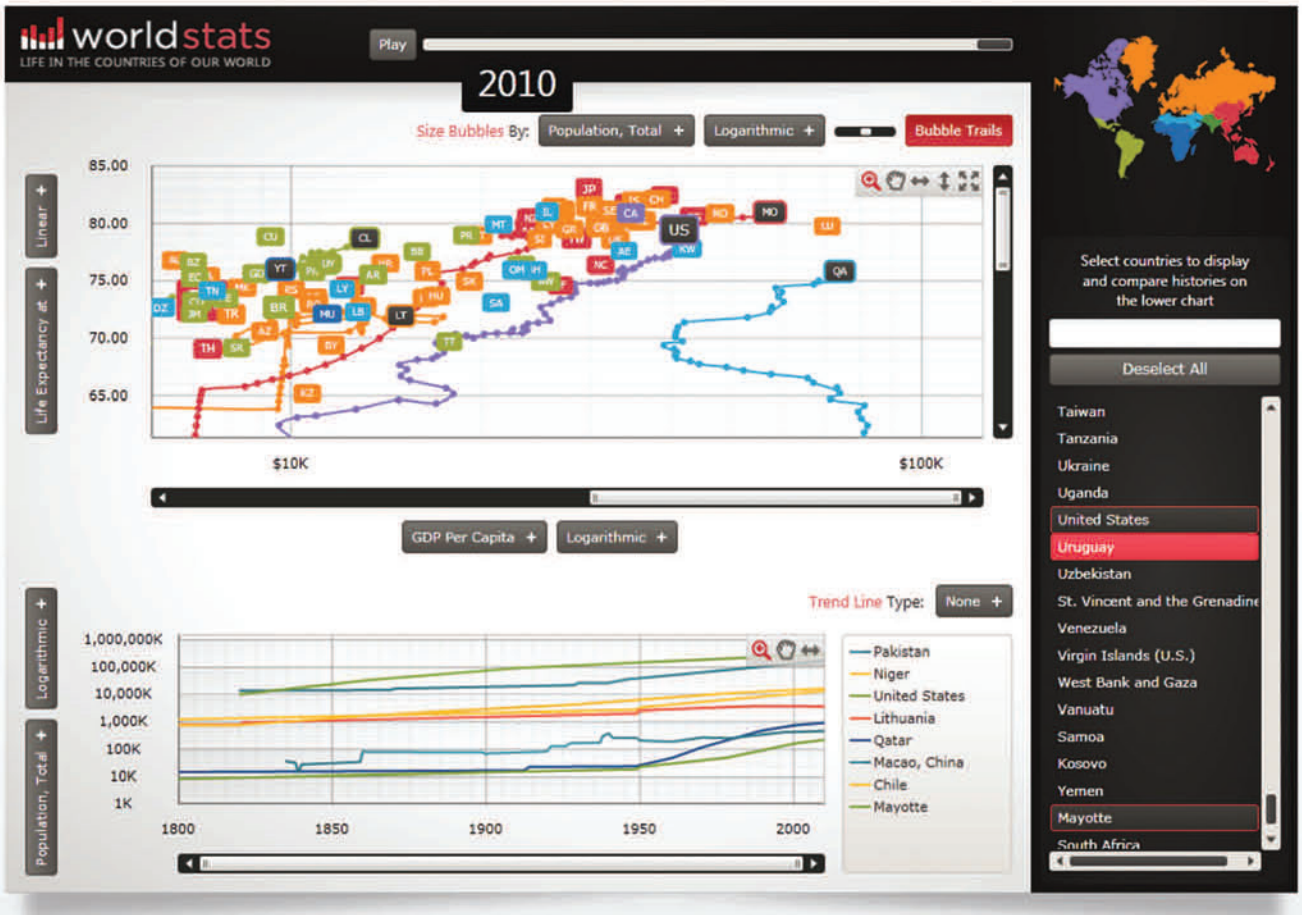
AGILE DEVELOPMENT

COLUMNS

Microsoft

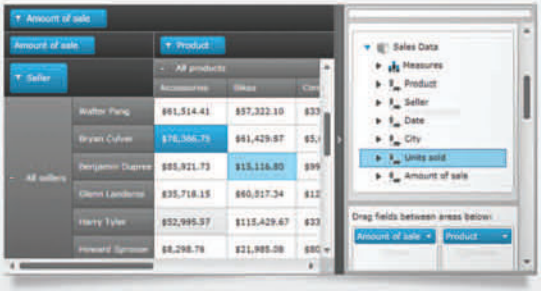
NetAdvantage®
ULTIMATE

REPORTING, DATA VISUALIZATION AND LOB UI CONTROLS FOR ASP.NET, WINDOWS FORMS, JQUERY/HTML5, WPF, SILVERLIGHT AND WINDOWS PHONE 7



INFRAGISTICS MOTION FRAMEWORK™

Delivering a great user experience in Windows Presentation Foundation (WPF) and Microsoft Silverlight business intelligence applications requires more than styling, it requires giving your application's end users greater insight into the story of their data.

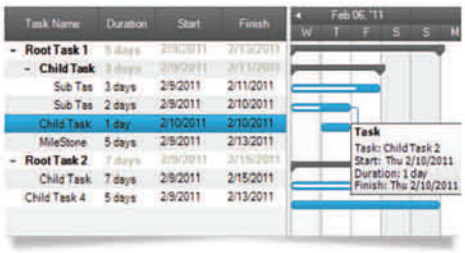


OLAP PIVOT GRID DATA VISUALIZATION

Work with multidimensional data from your OLAP cubes, data warehouses and Microsoft® SQL Server® Analysis Services.

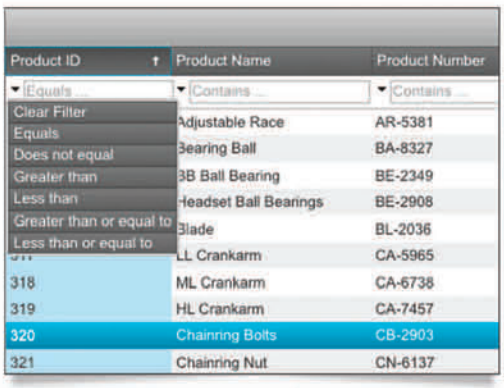
ASP.NET GAUGE

Whether it's for a sidebar gadget or an internal portal such as SharePoint®, gauges play a crucial role on any dashboard.



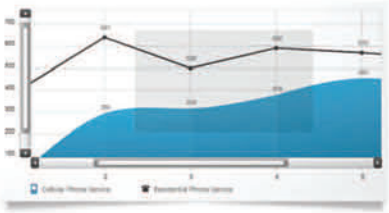
WINDOWS FORMS GANTT CHARTS

Deliver a Microsoft Project-style user experience to your users, with the composite tabular/timeline view of your scheduling data.



jQUERY

The most robust and forward-thinking product we have based on emerging Web technologies including HTML5 and CSS 3.



CHARTING

Make business charting quick and easy with fast, interactive, and vivid visuals across every .NET platform.

EXPORT TO MICROSOFT® WORD

New class library can create Word documents and stream xamGrid™ contents from Silverlight to a Word document.

SILVERLIGHT DATA VISUALIZATION

Use the Silverlight Data Visualization treemap control to communicate differences in data points with different pattern identifications.

SCAN HERE
for an exclusive
look at Ultimate!
www.infragistics.com/ult

TAKE YOUR APPLICATIONS TO THE NEXT LEVEL WITH OUR TOOLS

INFRAGISTICS.COM/ULTIMATE

INFRAGISTICS™
DESIGN / DEVELOP / EXPERIENCE

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

msdn magazine

JUNE 2011 VOLUME 26 NUMBER 6

MSDN Magazine (ISSN 1528-4859) is published monthly by 1105 Media, Inc., 9201 Oakdale Avenue, Ste. 101, Chatsworth, CA 91311. Periodicals postage paid at Chatsworth, CA 91311-9998, and at additional mailing offices. Annual subscription rates payable in US funds are: U.S. $35.00, International $60.00. Annual digital subscription rates payable in U.S. funds are: U.S. $25.00, International $25.00. Single copies/back issues: U.S. $10, all others $12. Send orders with payment to: *MSDN Magazine*, P.O. Box 3167, Carol Stream, IL 60132, email MSDNmag@1105service.com or call (847) 763-9560. **POSTMASTER:** Send address changes to *MSDN Magazine*, P.O. Box 2166, Skokie, IL 60076. Canada Publications Mail Agreement No: 40612608. Return Undeliverable Canadian Addresses to Circulation Dept. or XPO Returns: P.O. Box 201, Richmond Hill, ON L4B 4R5, Canada.

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

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

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

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

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

Microsoft

BPA WORLDWIDE

Printed in the USA

LEADTOOLS Document Imaging Suite SDK v17.0

by LEAD Technologies

- Libraries for C/C++, .NET, Silverlight, Windows Phone, WPF, WCF & WF
- High Performance OCR, ICR, MICR & OMR
- 1D & 2D Barcodes (Read/Write)
- Forms Recognition/Processing
- PDF, PDF/A and XPS
- Document Cleanup
- Advanced Compression (CCITT G3/G4, JBIG2, MRC, ABIC, ABC)
- High-Speed Scanning
- Print Capture and Document Writers

Paradise # L05 03301A02

$4,018.99

programmers.com/LEAD

UltraEdit

The #1 Best Selling Text Editor in the World

by IDM Computer Solutions

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

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

Named User 1-24 Users Paradise # I84 01201A01

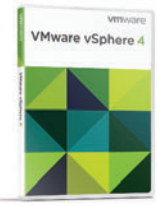
$59.95

programmers.com/idm

VMware vSphere Essentials Kit Bundle

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

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



for 3 hosts Paradise # V55 85101C02

$446.99

programmers.com/vSphere

Embarcadero Delphi XE

The Fastest Way to Build Native Windows Applications

by Embarcadero

Embarcadero® Delphi® XE is the fastest way to deliver ultra-rich, ultra-fast Windows applications. Dramatically reduce coding time and create applications 5x faster with component-based development and a fully visual two-way RAD IDE. Speed development across multiple Windows and database platforms — for GUI desktop applications, interactive touch-screen, kiosk, and database-driven multi-tier, cloud, and Web applications.

SPECIAL PRICE!

Paradise # CGI 32401A01

$1,977.99

programmers.com/embarcadero

Apple Mac Pro MC560LL/A Workstation

by Apple

The easy-access interior of the Mac Pro feels like the well-organized workstation it is. No rat's nest of components here. You don't need to turn the system on its side or struggle to reach into awkward spaces to make changes. Just remove the side panel for instant access to everything. Slide out the processor tray to add memory. Slide out drive bays to add storage. Slide a simple bar to change up to four expansion cards at once. And with plenty of I/O ports both front and back, you'll have room for all your external devices.

Paradise # ZHI BG5665

$2,498.99

programmers.com/apple

TX Text Control 16.0

Word Processing Components

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

New Version Released!

- .NET WinForms and WPF rich text box 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 12101A01

$1,109.99

Download a demo today.

programmers.com/textcontrol

ActiveReports 6

by GrapeCity PowerTools

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

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

NEW VERSION 6!

Professional Ed. Paradise # D03 04301A01

$1,310.99

programmers.com/grapecity

HP ProLiant DL380 G7 Servers

by Hewlett Packard

The HP ProLiant DL380 G7 Server continues to deliver on its heritage of engineering excellence with increased flexibility and performance, enterprise-class uptime and HP Insight Control manageability, 2 socket Intel® Xeon® performance, and 2U density for a variety of applications.

Paradise # ZHI CQ5948

$4,849.00

programmers.com/hp

Cisco® Catalyst® 2960-S Series Switches

Enhance productivity by using these switches to enable applications such as IP telephony, wireless, and video. These enterprise-class switches provide a borderless network experience that is easy to use and upgrade, as well as highly secure, sustainable, and available. The fixed-configuration access switches provide a lower total cost of ownership for enterprise, midmarket, and branch office networks.

48 ports of Gigabit Ethernet desktop connectivity with 4x1G SFP. Cisco FlexStack, a hot swappable module, provides true stacking with all switches in a stack acting as a single unit.

Paradise # ZHI CR3475

$4,099.00

programmers.com/cisco

StorageCraft ShadowProtect Desktop Edition 4.0

by StorageCraft Technology Corp

ShadowProtect Desktop provides fast and reliable disaster recovery, data protection and system migration to get desktops and laptops online as quickly as possible. ShadowProtect Desktop is an automated backup that works in the background. In the event of a disaster, its flexible recovery options allow you to complete granular recovery of files and folders or full bare metal recovery in a matter of minutes. Hardware Independent Restore (HIR) technology makes it quick and easy to recover to the same system, to dissimilar hardware or to and from virtual environments.

Minimum Quantity: 1 Paradise # SC5 02201E01

$83.99

programmers.com/storagecraft

Lenovo ThinkPad X220 4290

by Lenovo

Engineered for mobile professionals, the ThinkPad X220 features the quality and performance Lenovo users have come to expect and increased audio, video and communications features that respond to the increased use of laptops as a multimedia and communications tools in business. The ultra-portable ThinkPad X220 comes equipped with a full-powered Intel processor with outstanding graphic performance.

Paradise # ZHI GF0801

$2,169.00

programmers.com/lenovo

Win an iPad!

NO PURCHASE NECESSARY.

Use offer code WEBTRW06 when you place your order online or with your Programmer's Paradise representative and you'll automatically be entered into a drawing to win an iPad Wi-Fi 32GB.

For official rules, or to complete the online entry form:
programmers.com/tradewinds

EDITOR'S NOTE

KEITH WARD

The Best of Times

Last month, I wrote that your skills are never really outdated, no matter what platform you develop for. This month, following the MIX11 show (which is finishing up as I write this), I'd like to expand upon that positive thought: There has never been a better time, *in the history of IT*, to be a developer.

A bold claim? Yes. But there's ample evidence to back it up. Let's start with the best development trend since the Web became an everyday part of our lives: mobile devices.

Windows Phone 7, Android, iPhone and BlackBerry didn't exist as application venues a few years ago. And, in general, they're easier to develop for than, say, a standard Windows 7 desktop app. In fact, there are plenty of developers building apps for those devices by themselves, either freelance or as part of a larger company.

> One of the great things about developers is that they don't mind giving stuff away for free.

I talked to one developer who's written several Windows Phone 7 apps related to music. It took him no more than a week to write the basic app—more time was spent tweaking it, but the foundation was laid in a few blinks of the eye. That's been difficult to do in the past.

If you've got a great idea and get lucky enough to hit a nerve with your audience, you can make gobs of money on your 70 percent cut of each app. With most paid apps costing a buck, that means you make about 70 cents (on average; that's the normal deal in the mobile world).

In the old days of development, that kind of profit margin was laughable, and a product that promised such a tiny return would have never been approved. But we're now in the era of mass adoption on a mind-bending scale. Consider that Android recently passed the 3 billion app download mark, and iPhone has eclipsed 10 billion downloads. With Gartner bullish on the prospects for Windows Phone 7 going forward, that gives three thriving markets for your apps (if RIM can ever get its act together with BlackBerry, you can add a fourth). Even if many of those apps are free, that still leaves tons of paid apps, and the ability to sell within apps with things like upgrades.

The mobile category is itself expanding, too, opening up more avenues. The iPad demonstrated vividly the possibilities of the tablet form factor once and for all. Once Apple set the market (again), other companies started getting in the tablet game, of course. That means mobile devs have even more opportunity.

The other major reason to believe we're in the golden age of development is the explosion of online resources. Way back when, learning a new coding language usually meant a trip to Borders (remember them?) or Barnes & Noble to buy a book.

Of course, you can still get books—even in paper form. But Kindles, Nooks and the iPad are now the preferred book-reading method for many. Beyond books, however, is an entire universe of information online at sites like msdn.microsoft.com (and the broader MSDN Network), codeproject.com, stackoverflow.com, personal blogs and so on. Almost every conceivable question you may have has likely been asked and answered already.

And most of it won't cost you. One of the great things about developers is that they don't mind giving stuff away for free. Code samples, new approaches to building an app, ways to make your code more powerful and efficient—it's all out there, and most of it is yours for the taking, without plunking down a dime. I can't think of a single industry that's as philanthropic in nature.

Related to that idea is the social-networking boom. Through sites like LinkedIn, Facebook, Twitter and others, there's more ability than ever to meet colleagues, ask questions and get answers, and generally connect with this community. This is the greatest resource of them all. Again, a decade ago you didn't have any of it. Think about that.

Remember all this next time you're complaining about the long hours, low pay, insane deadlines, psychotic bosses and other frustrations of being a developer in 2011.

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

It's not really that helpful...

Unless success is important to you.

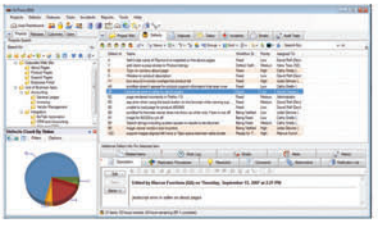
Project management and bug tracking for Agile & Scrum dev teams.

Hosted

OnTime Now!

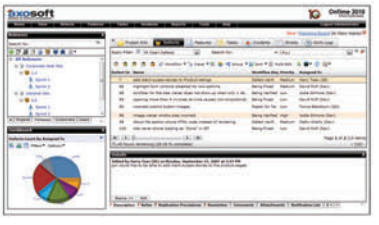
OnTime Now! Managing an Agile or Scrum dev team using a cloud-based solution has never been easier or more cost effective. Start a Free 30-Day Trial in seconds and see why Axosoft is a leading provider.

Windows



OnTime for Windows is installed in your own server environment. It uses a .NET & SQL back-end, integrates easily with Visual Studio, and is a total joy to use in the rich Windows client. Includes free SDK / APIs.

Web



OnTime Web provides you with an anywhere-anytime solution, whether you go with OnTime hosted or installed. This is what a web app is supposed to be like -- loaded with features and fully capable.

Free 1-user License • Free Download • Free Team Trial • Free Web Demo

Some of our awards:

axosoft.com
SOFTWARE FOR SOFTWARE DEVELOPMENT™
800.653.0024

DINO ESPOSITO

Invariants and Inheritance in Code Contracts

In past installments of this column I discussed two of the most common types of software contracts—preconditions and postconditions—and analyzed their syntax and semantics from the perspective of the Code Contracts API in the Microsoft .NET Framework 4. This month, I'll first introduce the third most important type of contract—the invariant—and then proceed to examine the behavior of contract-based classes when you apply inheritance.

Invariants

In general terms, an invariant is a condition that always holds true in a given context. Applied to object-oriented software, an invariant indicates a condition that always evaluates to true on each instance of a class. The invariant is a formidable tool that promptly informs you whenever the state of any instance of a given class becomes invalid. Put another way, an invariant contract formally defines the conditions under which an instance of a class is reckoned to be in a good state. As emphatic as it may sound, this is just the first concept you need to understand initially, and then implement, when you model a business domain through classes. Domain-driven design (DDD) is now the proven methodology to model complex business scenarios, and it assigns invariant logic a prominent position in the design. DDD, in fact, strongly recommends you should never deal with instances of classes that are in an invalid state. Likewise, DDD recommends that you write factories of your classes that return objects in a valid state and also that your objects return in a valid state after each operation.

DDD is a methodology and leaves actual implementation of the contracts up to you. In the .NET Framework 4, Code Contracts greatly help to realize a successful implementation, with minimal effort of your own. Let's learn more about invariants in the .NET Framework 4.

Where's the Invariant Logic?

Are invariants somehow related to good object modeling and software design? Deep understanding of the domain is essential. A deep understanding of the domain guides you naturally to finding your invariants. And some classes just don't need invariants. Lack of invariants is not an alarming symptom, per se. A class that has no constraints on what it's expected to hold and what it's expected to do simply has no invariant conditions. If this is what results from your analysis, then it's more than OK.

Imagine a class that represents news to be published. The class will likely have a title, an abstract and a publication date. Where are invariants here? Well, it depends on the business domain. Is date of publication required? If yes, you should ensure that the news always holds a valid date and the definition of what is a "valid date" also comes from the context. If the date is optional, then you can save an invariant and ensure that the content of the property is validated before use in the context in which it's being used. The same can be said for the title and abstract. Does it make sense to have news with neither title nor content? If it makes sense in the business scenario you're considering, you have a class with no invariants. Otherwise, be ready to add checks against nullness of title and content.

More generally, a class with no behavior and acting as a container of loosely related data may likely

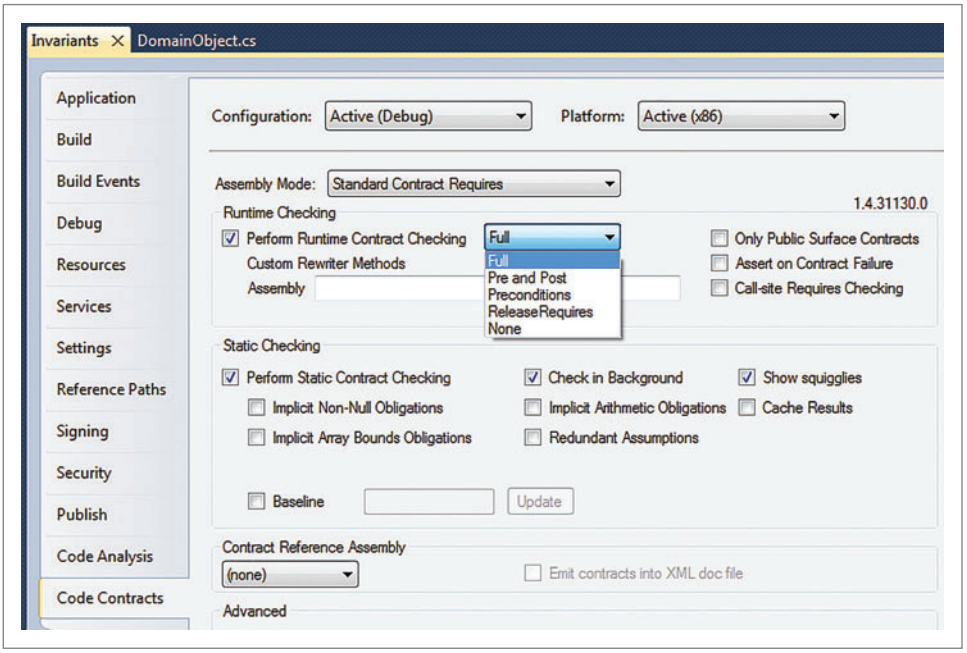
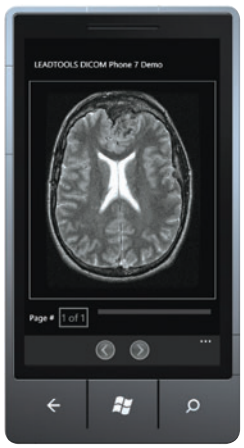


Figure 1 **Invariants Require Full Runtime Checking for Contracts**

LEADTOOLS® version 17

The World Leader in Imaging SDKs



Powered by Microsoft® Silverlight®

Windows Phone

Silverlight Imaging SDK

- **Load, display and save 100+ formats including PDF, JPEG2000 and TIFF**
- **Pure managed Silverlight binaries for Silverlight 3, 4 and Windows Phone**
- **Multi-touch support for Silverlight**
- **Annotation and markup**
- **Load, view, process and save DICOM**
- **Apply visual effects with pixel shaders**

The LEADTOOLS Silverlight SDK gives Web and Windows Phone developers the ability to load, display, process, convert and save many different image formats that are not intrinsic to the Silverlight framework. Because of the 100% managed Silverlight binaries, images can be processed and annotated directly within the Silverlight application without the need for external server calls.

The SDK comes with several viewing components including Image Viewer, Image List and Pan Window. The viewer is equipped with automated interactive tools including scroll, pan, scale, zoom and magnifying glass.

Adding Annotations and Markup to your Silverlight application is a cinch with LEADTOOLS fully automated annotations. Load, save, create and edit popular annotation types like line, ruler, polygon, curve, note, highlight, redaction and more.

Create highly interactive medical imaging applications with LEADTOOLS support for DICOM. Load, save and modify tags from DICOM data sets and display 12/16 bpp grayscale images with Window Leveling. LEADTOOLS is currently the only Windows Phone imaging toolkit on the market with support for DICOM data sets.

LEADTOOLS includes sample projects to demonstrate all of the SDK's capabilities including the display of over 100 image formats and incorporation of more than 200 image processing functions.

Free 60 Day Evaluation! **www.LEADTOOLS.com**
(800) 637-1840

LEAD TECHNOLOGIES INCORPORATED

Figure 2 **Invariants and Invariant-Aware Constructors**

```
public class News
{
  public News(String title, String body)
  {
    Contract.Requires<ArgumentException>(
      !String.IsNullOrEmpty(title));
    Contract.Requires<ArgumentException>(
      !String.IsNullOrEmpty(body));

    Title = title;
    Body = body;
  }

  public String Title { get; set; }
  public String Body { get; set; }

  [ContractInvariantMethod]
  private void ObjectInvariant()
  {
    Contract.Invariant(!String.IsNullOrEmpty(Title));
    Contract.Invariant(!String.IsNullOrEmpty(Body));
  }
}
```

be lacking invariants. If in doubt, I suggest you ask the question, "Can I store any value here?" for each property of the class, whether it's public, protected or private if set through methods. This should help to concretely understand whether you're missing important points of the model.

Like many other aspects of design, finding invariants is more fruitful if done early in the design process. Adding an invariant late in the development process is always possible, but it will cost you more in terms of refactoring. If you do that, be careful and cautious with regression.

Invariants in Code Contracts

In the .NET Framework 4, an invariant contract for a class is the collection of the conditions that should always hold true for any instance of the class. When you add contracts to a class, the preconditions are for finding bugs in the caller of the class, whereas postconditions and invariants are about finding bugs in your class and its subclasses.

You define the invariant contract through one or more ad hoc methods. These are instance methods, private, returning void and decorated with a special attribute—the ContractInvariantMethod attribute. In addition, the invariant methods aren't allowed to contain code other than the calls necessary to define invariant conditions. For example, you can't add any kind of logic in invariant methods, whether pure or not. You can't even add logic to simply log the state of the class. Here's how to define an invariant contract for a class:

```
public class News {
  public String Title {get; set;}
  public String Body {get; set;}

  [ContractInvariantMethod]
  private void ObjectInvariant()
  {
    Contract.Invariant(!String.IsNullOrEmpty(Title));
    Contract.Invariant(!String.IsNullOrEmpty(Body));
  }
}
```

The News class puts as invariant conditions that Title and Body will never be null or empty. Note that for this code to work, you need to enable full runtime checking in the project configuration for the various builds as appropriate (see **Figure 1**) .

Now try the following simple code:

```
var n = new News();
```

You'll be surprised to receive a contract failed exception. You successfully created a new instance of the News class; it's bad luck it was in an invalid state. A new perspective of invariants is required.

In DDD, invariants are associated with the concept of a factory. A factory is just a public method responsible for creating instances of a class. In DDD, each factory is responsible for returning instances of domain entities in a valid state. The bottom line is that when you use invariants, you should guarantee that conditions are met at any given time.

But what specific time? Both DDD and the actual implementation of Code Contracts agree that invariants be checked at the exit from any public method, including constructors and setters. **Figure 2** shows a revised version of the News class that adds a constructor. A factory is the same as a constructor except that—being a static method—it can be given a custom and context-sensitive name and result in more readable code.

The code that consumes the class News can be the following:

```
var n = new News("Title", "This is the news");
```

This code won't throw any exception because the instance is created and returned in a state that meets invariants. What if you add the following line:

```
var n = new News("Title", "This is the news");
n.Title = "";
```

> A factory is just a public method responsible for creating instances of a class.

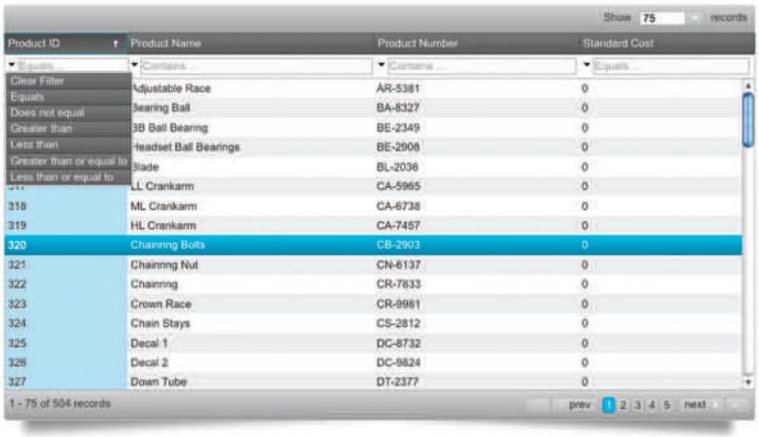
Setting the Title property to the empty string puts the object in an invalid state. Because invariants are checked at the exit of public methods—and property setters are public methods—you get an exception again. It's interesting to note that if you use public fields instead of public properties, then invariants aren't checked and the code runs just fine. Your object, however, is in an invalid state.

Note that having objects in an invalid state isn't necessarily a source of trouble. In large systems, however, to stay on the safe side, you might want to manage things so you automatically get exceptions if an invalid state is caught. This helps you manage

Figure 3 **Invariant-Based Root Class for a Domain Model**

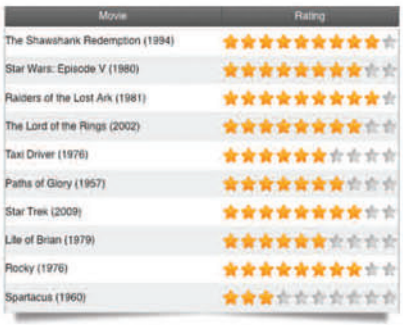
```
public abstract class DomainObject
{
  public abstract Boolean IsValid();

  [Pure]
  private Boolean IsValidState()
  {
    return IsValid();
  }

  [ContractInvariantMethod]
  private void ObjectInvariant()
  {
    Contract.Invariant(IsValidState());
  }
}
```

NetAdvantage®
for jQuery



IG GRID ALL FEATURES ENABLED

The grid's sophisticated filtering and sorting features allow your users to find the records they need to work with.

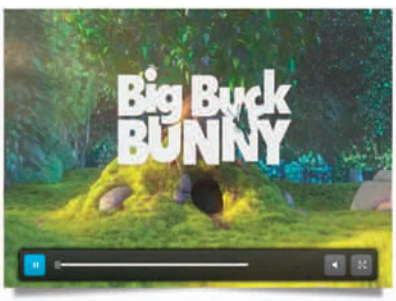


RATING

Enable a social experience for users, by letting them rate their favorite movies or anything else with our rating control.

IG GRID ON HTML PAGE

High performance grid lets users page through any OData source, even streaming movie descriptions.

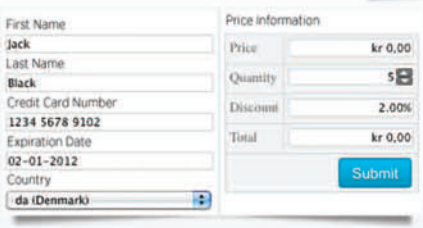


IG VIDEO PLAYER MVC

When a user finds what they want to watch, our HTML5 video player adds streaming video right into your own apps.

IG EDITORS

Robust data entry and editing controls assist users with entering data in the formats your application needs.



UPLOADERS

Accept file uploads of any kind of content straight from your users with background file transfers.

SCAN HERE
for an exclusive
look at jQuery!
www.infragistics.com/jq

TAKE YOUR WEB APPLICATIONS TO THE NEXT LEVEL WITH OUR TOOLS
INFRAGISTICS.COM/JQUERY

INFRAGISTICS™
DESIGN / DEVELOP / EXPERIENCE

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

Figure 4 **Overriding a Method Used by Invariants**

```
public class Customer : DomainObject
{
  private Int32 _id;
  private String _companyName, _contact;

  public Customer(Int32 id, String company)
  {
    Contract.Requires(id > 0);
    Contract.Requires(company.Length > 5);
    Contract.Requires(!String.IsNullOrWhiteSpace(company));

    Id = id;
    CompanyName = company;
  }
  ...
  public override bool IsValid()
  {
    return (Id > 0 && !String.IsNullOrWhiteSpace(CompanyName));
  }
}
```

development and conduct your tests. In small applications, invariants may not be a necessity even when some emerge from the analysis.

Although invariants must be verified at the exit from a public method, within the body of any public method the state can be temporarily invalid. What matters is that invariants hold true before and after the execution of public methods.

How do you prevent the object from getting into an invalid state? A static analysis tool such as the Microsoft Static Code Checker would be able to detect that a given assignment is going to violate an invariant. Invariants protect you from broken behavior but can also help identify poorly specified inputs. By specifying those properly, you can more easily find bugs in the code using a given class.

Contract Inheritance

Figure 3 shows another class with an invariant method defined. This class can serve as the root of a domain model.

In the DomainObject class, the invariant condition is expressed through a private method declared as a pure method (that is, not altering the state). Internally, the private method calls an abstract method that derived classes will use to indicate their own invariants. **Figure 4** shows a possible class derived from DomainObject that overrides the IsValid method.

Figure 5 **Overriding a Method Used by Invariants**

```
public abstract class DomainObject
{
  protected Boolean Initialized;
  public abstract Boolean IsValid();

  [Pure]
  private Boolean IsInValidState()
  {
    return !Initialized || IsValid();
  }

  [ContractInvariantMethod]
  private void ObjectInvariant()
  {
    Contract.Invariant(IsInValidState());
  }
}

public class Customer : DomainObject
{
  public Customer(Int32 id, String company)
  {
     ...
    Id = id;
    CompanyName = company;
    Initialized = true;
  }
     ...
}
```

The solution seems elegant and effective. Let's try to get a new instance of the Customer class passing valid data:

```
var c = new Customer(1, "DinoEs");
```

If we stop looking at the Customer constructor, everything looks fine. However, because Customer inherits from DomainObject, the DomainObject constructor is invoked and the invariant is checked. Because IsValid on DomainObject is virtual (actually, abstract) the call is redirected to IsValid as defined on Customer. Unfortunately, the check occurs on an instance that hasn't been completely initialized. You get an exception, but it's not your fault. (In the latest release of Code Contracts, this issue has been resolved and invariant checking on constructors is delayed until the outermost constructor has been called.)

This scenario maps to a known issue: don't call virtual members from a constructor. In this case, it happens not because you directly coded this way, but as a side effect of contract inheritance. You have two solutions: dropping the abstract IsValid from the base class or resorting to the code in **Figure 5**.

The Initialized protected member acts as a guard and won't call into overridden IsValid until the actual object is initialized. The Initialized member can be a field or a property. If it's a property, though, you get a second tour of the invariants—which isn't strictly necessary, as everything already has been checked once. In this respect, using a field results in slightly faster code.

Contract inheritance is automatic in the sense that a derived class automatically receives the contracts defined on its base class. In this way, a derived class adds its own preconditions to the preconditions of the base class. The same occurs for postconditions and invariants. When dealing with an inheritance chain, the intermediate language rewriter sums up contracts and calls them in the right order where and when appropriate.

Be Careful

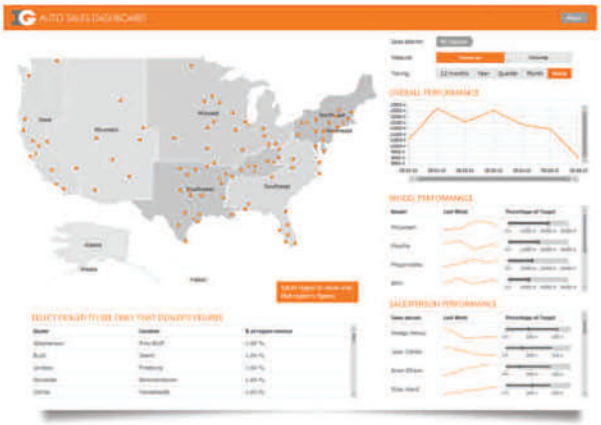
Invariants aren't bulletproof. Sometimes invariants can help you in one way and cause problems in another, especially if used in every class and in the context of a class hierarchy. Although you should always endeavor to identify invariant logic in your classes, if implementing invariants hits corner cases, then I'd say you'd better drop them from implementation. However, keep in mind that if you hit corner cases, they're likely due to complexity in the model. Invariants are the tools you need to manage implementation problems; they aren't the problem themselves.

There's at least one reason why summing up contracts across a class hierarchy is a delicate operation. It's too long to discuss it here, but it makes good fodder for next month's article. ■

DINO ESPOSITO *is the author of "Programming Microsoft ASP.NET 4" (Microsoft Press, 2011) and coauthor of "Microsoft .NET: Architecting Applications for the Enterprise" (Microsoft Press, 2008). Based in Italy, Esposito is a frequent speaker at industry events worldwide. You can follow him on Twitter at twitter.com/despos.*

THANKS to the following technical experts for reviewing this article: *Manuel Fahndrich and Brian Grunkemeyer*

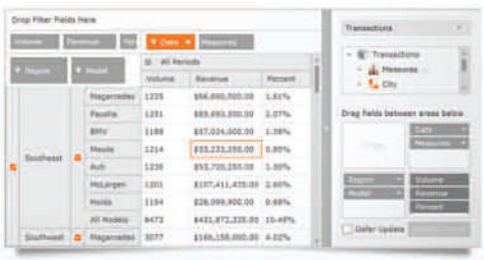
NetAdvantage®
for Data Visualization



INTERACTIVE DASHBOARDS
Build rich dashboards that visualize business data to empower decision makers.

MAP OUT ANYTHING AND EVERYTHING
Use xamMap™ to get an instant view and show anything including seating charts, floor plans, warehouse contents, and yes—geographic maps, too!

MEDIA TIMELINE
The timeline highlights how easily users can search, browse and play popular videos from YouTube.



OLAP PIVOT GRID DATA VISUALIZATION
Work with multidimensional data from your OLAP cubes, data warehouses and Microsoft® SQL Server® Analysis Services.

INFRAGISTICS MOTION FRAMEWORK™
Create an immersive and animated user experience that tells the story of your data over time like no other visualization can.

SCAN HERE
for an exclusive
look at Data Visualization!
www.infragistics.com/dvis

TAKE YOUR APPLICATIONS TO
THE NEXT LEVEL WITH OUR TOOLS
INFRAGISTICS.COM/DV

INFRAGISTICS™
DESIGN / DEVELOP / EXPERIENCE

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

JULIE LERMAN

Demystifying Entity Framework Strategies: Loading Related Data

In last month's Data Points column, I provided some high-level guidance for choosing a modeling workflow strategy from the Database First, Model First and Code First options. This month, I'll cover another important choice you'll need to make: how to retrieve related data from your database. You can use eager loading, explicit loading, lazy loading or even query projections.

This won't be a one-time decision, however, because different scenarios in your application may require different data-loading strategies. Therefore, it's good to be aware of each strategy so that you can choose the right one for the job.

As an example, let's say you have an application that keeps track of family pets and your model has a Family class and a Pet class with a one-to-many relationship between Family and Pet. Say you want to retrieve information about a family and their pets.

In the next column, I'll continue this series by addressing the various choices you have for querying the Entity Framework using LINQ to Entities, Entity SQL and variations on each of those options. But in this column, I'll use only LINQ to Entities for each of the examples.

> Different scenarios in your application may require different data-loading strategies.

Eager Loading in a Single Database Trip

Eager loading lets you bring all of the data back from the database in one trip. The Entity Framework provides the Include method to enable this. Include takes a string representing a navigation path to related data. Here's an example of the Include method that will return graphs, each containing a Family and a collection of their Pets:

```
from f in context.Families.Include("Pets") select f
```

If your model has another entity called VetVisit and that has a one-to-many relationship with Pet, you can bring families, their pets and their pet's vet visits back all at once:

```
from f in context.Families.Include("Pets.VetVisits") select f
```

Results of eager loading are returned as object graphs, as shown in **Figure 1**.

Include is pretty flexible. You can use multiple navigation paths at once and you can also navigate to parent entities or through many-to-many relationships.

Eager loading with Include is very convenient, indeed, but if overused—if you put many Includes in a single query or many navigation paths in a single Include—it can detract from query performance pretty rapidly. The native query that the Entity Framework builds will have many joins, and to accommodate returning your requested graphs, the shape of the database results might be much more complex than necessary or may return more results than necessary. This doesn't mean you should avoid Include, but that you should profile your Entity Framework queries to be sure that you aren't generating poorly performing queries. In cases where the native queries are particularly gruesome, you should rethink your query strategy for the area of the application that's generating that query.

Lazy Loading in Additional Database Trips

Often when retrieving data, you don't want or need the related data right away, or you may not want related data for all of the results. For example, you might need to pull all of the families into your app, but then only retrieve pets for some of those people. Does it make sense in this case to eager load all of the pets for every person entity with Include? Probably not.

The Entity Framework offers two ways to load related data after the fact. The first is called lazy loading and, with the appropriate settings, it happens automatically.

With lazy loading, you simply need to make some reference to the related data and the Entity Framework will check to see whether it's been loaded into memory. If not, the Entity Framework will create and execute a query behind the scenes, populating the related data.

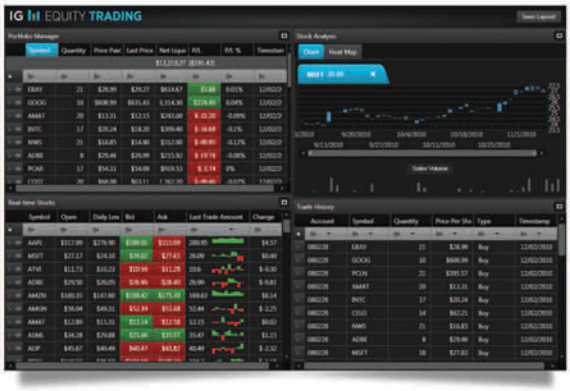
For example, if you execute a query to get some Family objects, then trigger the Entity Framework to get the Pets for one of those Families simply by making mention, so to speak, of the Pets property, the Entity Framework will retrieve the Pets for you:

```
var theFamilies= context.Families.ToList();
var petsForOneFamily = theFamilies[0].Pets;
```

Figure 1 **Object Graph Returned by Eager Loading Query**

Id	Name	Pets					
		Id	Name	Type	VetVisits		
2	LermanJ	2	Sampson	Dog	1	2/1/2011	Excellent
					5	4/1/2011	Nail Clipping
		4	Sissy	Cat	1	3/2/2011	Excellent
3	GeigerA	3	Pokey	Turtle	3	2/5/2011	Excellent
		4	Riki	Cat	6	4/8/2011	Excellent

NetAdvantage® PERFORMANCE

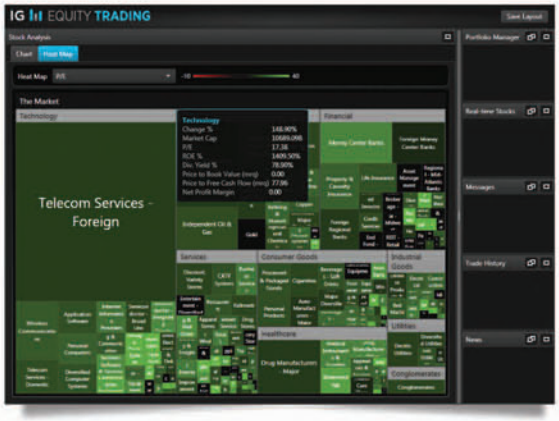


REAL-TIME USER INTERFACES

Refresh your app's user interface tick-by-tick on every value change in the fastest applications imaginable.

ACROSS ALL PLATFORMS

Charts and grids featuring blazing speed whether you are using ASP.NET, Silverlight, WPF or any other .NET platform.



DEEP DRILLDOWN

Dive into the deepest details of your data with crisp, rapidly-updating visual controls designed to handle it all.

TAKE IT ON THE ROAD

Our jQuery controls for iPhone/iPad/Android and Windows Phone 7 support means your UI is always on the go!

SCAN HERE
for an exclusive
look at Performance!
www.infragistics.com/perf

TAKE YOUR APPLICATIONS TO
THE NEXT LEVEL WITH OUR TOOLS
INFRAGISTICS.COM/PERFORMANCE

INFRAGISTICS™
DESIGN / DEVELOP / EXPERIENCE

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

In the Entity Framework, lazy loading is enabled or disabled using the ObjectContext ContextOptions.LazyLoadingEnabled property. By default, Visual Studio will define newly created models to set LazyLoadingEnabled to true, the result being that it's enabled by default with new models.

Having lazy loading enabled by default when you instantiate a context may be a great choice for your application, but can also be a problem for developers who aren't aware of this behavior. You may trigger extra trips to the database without realizing you're doing so. It's up to you to be aware of whether you're using lazy loading, and you can explicitly choose to use it or not—enabling or disabling it in your code by setting LazyLoadingEnabled to true or false—as needed.

Lazy loading is driven by the EntityCollection and EntityReference classes and therefore won't be available by default when you're using Plain Old CLR Object (POCO) classes—even if LazyLoadingEnabled is true. However, the Entity Framework dynamic proxy behavior, triggered by marking navigation properties as virtual or Overridable, will create a runtime proxy that will allow your POCOs to lazy load as well.

Lazy loading is a great feature to have available in the Entity Framework, but only if you're aware of when it's active and consider making choices of when it's appropriate and when it's not. For example, the *MSDN Magazine* article, "Using the Entity Framework to Reduce Network Latency to SQL Azure" (msdn.microsoft.com/magazine/gg309181), highlights performance implications of using lazy loading against a cloud database from an on-premises server. Profiling the database queries executed by the Entity Framework is an important part of your strategy to help you choose between loading strategies.

> Lazy loading is a great feature to have available in the Entity Framework.

It's also important to be aware of when lazy loading is disabled. If LazyLoadingEnabled is set to false, the reference to theFamilies[0].Pets in the previous example would not trigger a database query and would report that the family has no pets, even though there may be some in the database. So if you're relying on lazy loading, be sure that you've got it enabled.

Explicitly Loading in Additional Database Trips

You may want to leave lazy loading disabled and have more explicit control over when related data is loaded. In addition to explicitly loading with Include, the Entity Framework allows you to selectively and explicitly retrieve related data using one of its Load methods.

Figure 2 **Projected Anonymous Types with Family and Pets Properties**

Family		Pets		
Id	Name	Id	Name	Type
2	LermanJ			
3	GeigerA	3	Pokey	Turtle
		4	Riki	Cat

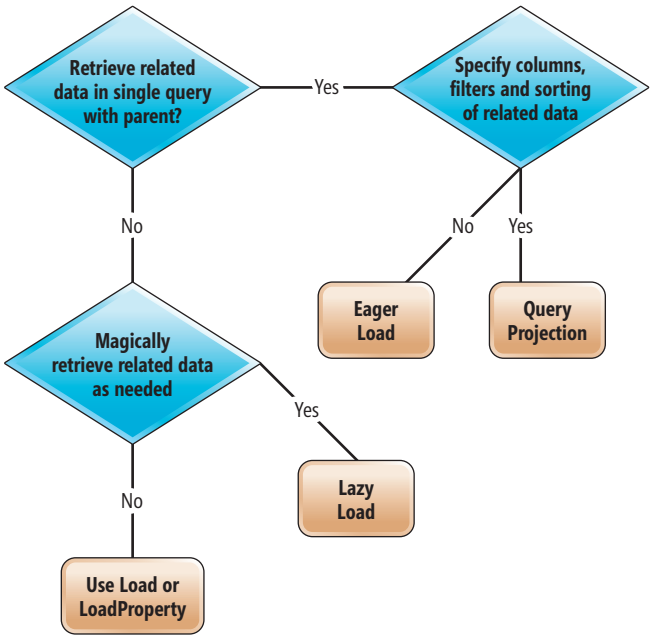


Figure 3 **Your First Pass at Loading Strategy Decisions**

If you generate entity classes using the default code-generation template, the classes will inherit from EntityObject, with related data exposed in an EntityCollection or an EntityReference. Both of these types have a Load method that you can call to force the Entity Framework to retrieve the related data. Here's an example of loading an EntityCollection of Pets objects. Notice that Load doesn't have a return value:

```
var theFamilies = context.Families.ToList();
theFamilies[0].Pets.Load();
var petsForOneFamily = theFamilies[0].Pets;
```

The Entity Framework will create and execute a query that populates the related property—the Pets collection for the Family—and then you can work with the Pets.

The second way to explicitly Load is from the ObjectContext, rather than from EntityCollection or EntityReference. If you're relying on POCO support in the Entity Framework, your navigation properties won't be EntityCollections or EntityReferences, and therefore won't have the Load method. Instead, you can use the ObjectContext.LoadProperty method. LoadProperty uses generics to identify the type that you're loading from and then a lambda expression to specify which navigation property to load. Here's an example of using LoadProperty to retrieve the Pets for a particular person instance:

```
context.LoadProperty<Family>(familyInstance, f => f.Pets)
```

Query Projections as an Alternative to Loading

Don't forget that you also have the option to use projections in your queries. For example, you can write a query to retrieve entities, but filter which related data is retrieved:

```
var famsAndPets=from family in context.Families
  select new {family,Pets=family.Pets.Any(p=>p.Type=="Reptile")};
```

This will return all of the families and the pets for any of those families that have any reptiles—all in a single trip to the database. But rather than a graph of family with their pets, the famsAndPets query will return a set of anonymous types with one property for Family and another for Pets (see **Figure 2**).

Evaluate the Pros and Cons

You now have four strategies available for retrieving related data. They need not be mutually exclusive in your application. You may very well find cause for each of these different features in various scenarios throughout your applications. The pros and cons of each strategy should be considered before choosing the right one for each case.

Eager loading with Include is useful for scenarios where you know in advance that you want the related data for all of the core data being queried. But remember the two potential downsides. If you have too many Includes or navigation paths, the Entity Framework may generate a poorly performing query. And you should be careful about returning more related data than necessary thanks to the ease of coding with Include.

Lazy loading very conveniently retrieves related data behind the scenes for you in response to code that simply makes mention of that related data. It, too, makes coding simpler, but you should be conscientious about how much interaction it's causing with the database. You may cause 40 trips to the database when only one or two were necessary.

> You also have the option to use projections in your queries.

Explicit loading gives you more control over when related data is retrieved (and which data), but if you don't know about this option, your application could be misinformed about the presence of related data in the database. Many developers find it cumbersome to explicitly load, while others are happy to have the granular control it provides.

Using projection in your queries potentially gives you the best of both worlds, selectively retrieving related data in a single query. However, if you're returning anonymous types from these projections, you may find those more cumbersome to work with, as the objects aren't tracked by the Entity Framework state manager, so they aren't updateable.

Figure 3 shows a decision-making flowchart that you can use for your first pass at choosing a strategy. But you should also be considerate of performance. Take advantage of query profiling and performance-testing tools to ensure that you've made the right data-loading choices. ■

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. She blogs at thedatafarm.com/blog and is the author of the highly acclaimed book, "Programming Entity Framework" (O'Reilly Media, 2010). Follow her on Twitter at twitter.com/julielerman.*

THANKS to the following technical expert for reviewing this article: *Tim Laverty*

JOSEPH FULTZ

Windows Azure Storage: Access for All

Windows Azure Storage is far from a device-specific technology, and that's a good thing. This month, I'll take a look at developing on three mobile platforms: Windows Phone 7, jQuery and Android.

To that end, I'll create a simple application for each that will make one REST call to get an image list from a predetermined Windows Azure Storage container and display the thumbnails in a filmstrip, with the balance of the screen displaying the selected image as seen in **Figure 1**.

Preparing the Storage Container

I'll need a Windows Azure Storage account and the primary access key for the tool I use for uploads. In the case of secure access from the client I'm developing I would also need it. That information can found in the Windows Azure Platform Management Portal.

I grabbed a few random images from the Internet and my computer to upload. Instead of writing upload code, I used the Windows Azure Storage Explorer, found at azurestorageexplorer.codeplex.com. For reasons explained later, images need to be less than about 1MB. If using this code exactly, it's best to stay at 512KB or less. I create a container named Filecabinet; once the container is created and the images uploaded, the Windows Azure piece is ready.

Platform Paradigms

Each of the platforms brings with it certain constructs, enablers and constraints. For the Silverlight and the Android client, I took the familiar model of marshaling the data in and object collection for consumption by the UI. While jQuery has support for templates, in this case I found it easier to simply fetch the XML and generate the needed HTML via jQuery directly, making the jQuery implementation rather flat. I won't go into detail about its flow, but for the other two examples I want to give a little more background. (Preparation steps for all three platforms can be found in the online version of this article at msdn.microsoft.com/magazine/hh184836.)

Windows Phone 7 and Silverlight

If you're familiar with Silverlight development you'll have no problem creating a new Windows Phone 7 application project in Visual Studio 2010. If not, the things you'll need to understand for this example are observable collections (bit.ly/18sPUF), general XAML controls (such as StackPanel and ListBox) and the WebClient.

At the start, the application's main page makes a REST call to Windows Azure Storage, which is asynchronous by design.

Code download available at code.msdn.microsoft.com/mag201106Cloudy.

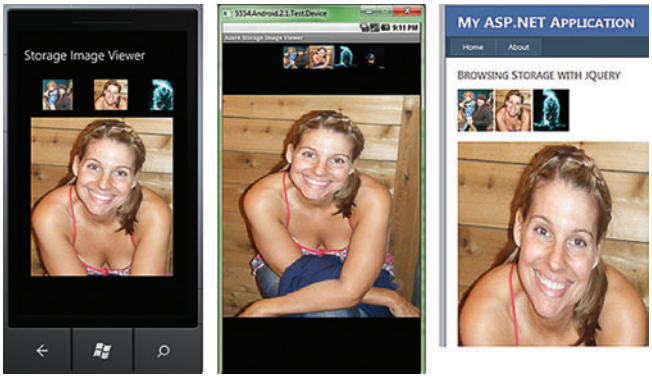


Figure 1 **Side-by-Side Storage Image Viewers**

Windows Phone 7 forces this paradigm as a means to ensure that any one app will not end up blocking and locking the device. The data retrieved from the call will be placed into the data context and will be an ObservableCollection<> type.

This allows the container to be notified and pick up changes when the data in the collection is updated, without me having to explicitly execute a refresh. While Silverlight can be very complex for complex UIs, it also provides a low barrier of entry for relatively simple tasks such as this one. With built-in support for binding and service calls added to the support for touch and physics in the UI, the filmstrip to zoom view is no more difficult than writing an ASP.NET page with a data-bound Grid.

Android Activity

Android introduces its own platform paradigm. Fortunately, it isn't something far-fetched or hard to understand. For those who are primarily .NET developers, it's easy to map to familiar constructs, terms and technologies. In Android, pretty much anything you want to do with the user is represented as an Activity object. The Activity could be considered one of the four basic elements that may be combined to make an application: Activity, Service, Broadcast Receiver and Content Provider.

For simplicity, I've kept all code in Activity. In a more realistic implementation, the code to fetch and manipulate the images from Windows Azure Storage would be implemented in a Service.

This sample is more akin to putting the database access in the code behind the form. From a Windows perspective, you might think of an Activity as a Windows Forms that has a means—in fact, the *need*—to save and restore state. An Activity has a lifecycle much like Windows Forms; an application could be made of

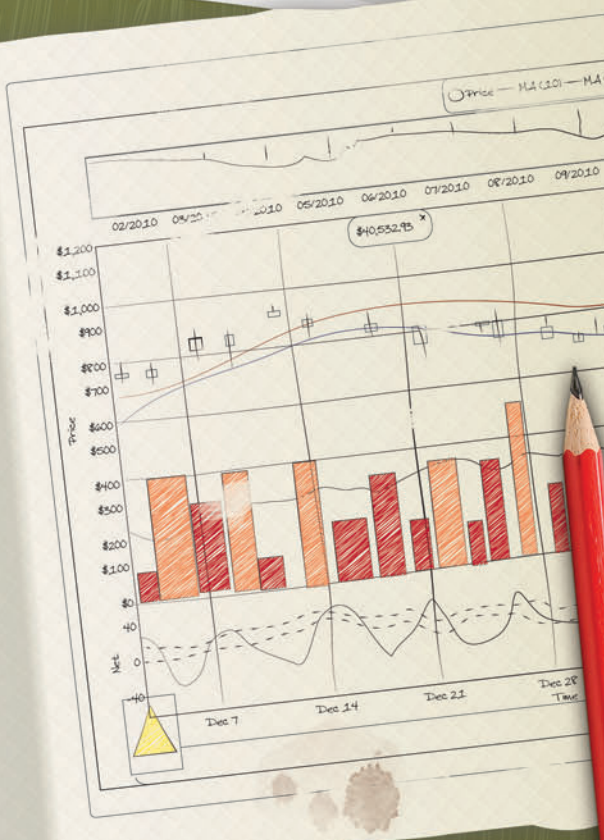
Your Guide to .NET Charts

Ahead of Schedule and Under Budget

With the right tools you can turn your long development list of requirements into a finished solution, from conception to completion. Developers, just like you, continue to turn to ComponentOne Charts™ for their enterprise solutions. Get rock solid, advanced charts for all .NET platforms; download ComponentOne Studio® Enterprise today.

Requirements:

- ✓ Financial & scientific charts
- ✓ Stacked charts
- ✓ Interactivity - zoom, animation, drag-and-drop
- ✓ Tooltips and markers
- ✓ Trend lines
- ✓ ~~Fast~~ Really fast
- ✓ Live updates
- ✓ 2D & 3D
- ✓ Multiple platforms - Silverlight, WPF, ASP.NET and WinForms



ComponentOne® Studio® Enterprise

Download your FREE Trial @ c1.ms/fastcharts

ComponentOne

Figure 2 **Object to Match XML Returned from Storage Call**

```
<EnumerationResults ContainerName="http://jofultz.blob.core.windows.net/filecabinet">
<Blobs>
<Blob>
<Name>2010-12-12+10.40.06.jpg</Name>
<Url>http://jofultz.blob.core.windows.net/filecabinet/2010-12-12+10.40.06.jpg</Url>
<LastModified>Sun, 02 Jan 2011 02:24:24 GMT</LastModified>
<Etag>0x8CD783B4E56116C</Etag>
<Size>263506</Size>
<ContentType>image/jpeg</ContentType>
<ContentEncoding />
<ContentLanguage />
</Blob>
<Blob>
<Name>cookie-monster.jpg</Name>
<Url>http://jofultz.blob.core.windows.net/filecabinet/cookie-monster.jpg</Url>
<LastModified>Mon, 27 Dec 2010 09:40:18 GMT</LastModified>
<Etag>0x8CD73C13542256C</Etag>
<Size>265995</Size>
<ContentType>image/jpeg</ContentType>
<ContentEncoding />
<ContentLanguage />
</Blob>
<Blob>
```

one-to-many Activities, but only one will be interacting with the user at a time. For more information on the fundamentals of Android applications, go to bit.ly/d3c7C.

This sample app will consist of a single Activity (bit.ly/GmWui), a BaseAdapter (bit.ly/g0J2Qx) and a custom object to hold information about the images.

Creating the UIs

The common tasks in each of the UI paradigms include making the REST call, marshaling the return to a useful datatype, binding and displaying, and handling the click event to show the zoom view of the image. First, I want to review the data coming back from the REST Get, and what's needed for each example to get the data to something more easily consumable.

The Data

Within the application code I prefer to work with objects and not constantly have to manipulate XML, so I create an object to match the XML returned from the storage call. It looks like the code in **Figure 2**.

I then create a representative object to hold the information I want. Mostly I'm concerned with the Name and URL fields. **Figure 3** provides a look at the declaration for the objects used for each example.

Creating the UIs

Fortunately, the way the UI is defined in Android and Silverlight is similar in their use of markup, so understanding one nearly guarantees understanding the other. Additionally, they both use an XML markup to define the UI.

The jQuery implementation will simply use HTML and generate some elements dynamically. The big difference between the three when writing markup is knowledge of the controls/elements to be used.

Starting with the Windows Phone 7 XAML, I open up the MainPage.xaml and add a <Grid/>, which will contain a <ListBox /> for the filmstrip. The list box will contain an <ItemTemplate/> and <DataTemplate /> that will control how data will render for the filmstrip. Also in the grid will be an <Image/> control for the zoom view of the image.

For Android, I'll set up a similar UI construct using a LinearLayout that contains two widgets: Gallery and ImageView. For those familiar with Silverlight or Windows Presentation Foundation (WPF), a LinearLayout is like a StackPanel. Widgets are controls and this would be roughly equivalent to a StackPanel that contains two elements: a horizontal ListBox of Images and an Image control. Note that this is almost exactly what's used in the Silverlight example. The Gallery will provide a filmstrip-style view of thumbnails and the ImageView will show a zoomed view of the selected thumbnail.

For the jQuery example, I'll have an empty <p/> with the id assigned the name "output"; that will be where the thumbnails views will be dynamically added via the jQuery. The Silverlight markup is a bit longer due to the handling of the binding and rendering in the markup. A little more code on the Android side handles the data loading. (You can see the markups in the online version of the article.)

The Main Code

The next step is to create the code that contacts Windows Azure Storage for information about the blobs. The fetch request takes the form of http://[your storage account].blob.core.windows.net/[your container]?comp=list; my request looks like http://jofultz.blob.core.windows.net/filecabinet?comp=list.

Because the results are XML, in Silverlight I used LINQ to provide a nice object graph and access pattern. I wanted something like that for Java, but in the end, wanting to skip the debugging of unfamiliar technology, I chose to just work with the Document Object Model (DOM)-style interface provided by the DocumentBuilder API. Because this was a Read-only operation against a few elements, it wasn't a big deal. For the jQuery, it's just a selector-style query for which the path is much like simply dealing with the DOM.

The methods for fetching are pretty straightforward: Because the container is public, I don't need to deal with a proxy or with any special authentication.

Figure 3 **Object Declaration for Objects Used in Each Client Example**

Silverlight	Android	jQuery

`public static ObservableCollection<StorageImage> ImageInfoList {get;set;}`
 	
`...`
 	
`public class StorageImage`
`{`
`public string Name{get;set;}`
`public string Url {get;set;}`
`public BitmapImage Image {get;set;}`
`}`
	`public ArrayList<BlobInfo> BlobList= null;`
	
	`...`
	
	`public class BlobInfo {`
	`public String Name;`
	`public String Url;`
	`public Bitmap ImageBitmap;}`
		None
		
		The XML is consumed directly.

For both Silverlight and jQuery I provide a callback to handle the response, as the request is made asynchronously and the response is passed to the callback to handle. For Android, the request is issued synchronously. For jQuery and Silverlight, I'll need one more supporting method in each of them to handle the data and get it either in a consumable format or, in the jQuery case, render it directly.

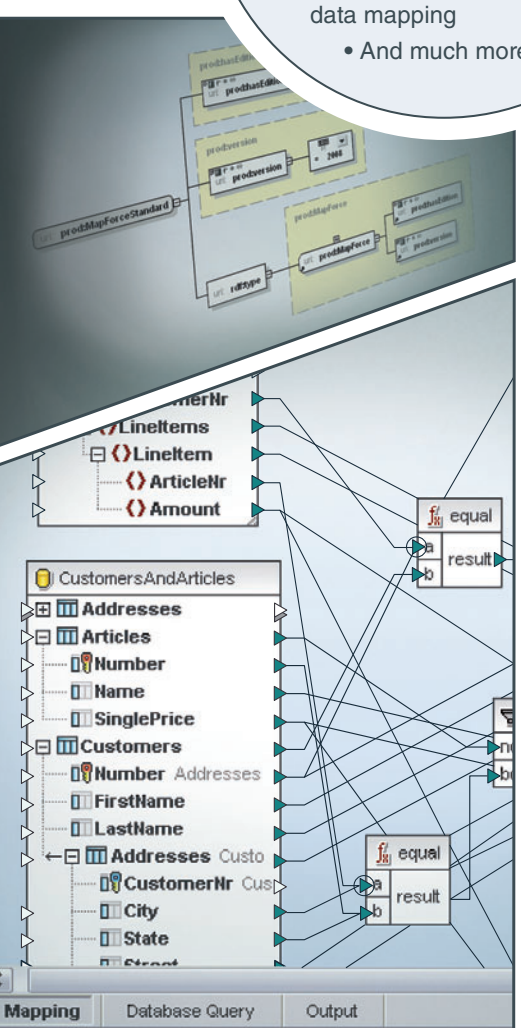
Visualize software works of art with the complete set of tools from Altova®

ALTOVA® missionkit®

The MissionKit®, is an integrated suite of UML, XML, and data integration tools for today's software architect.

New in Version 2011:

- UML modeling of SQL databases
- Support for BPMN 2.0
- Code generation from State Machine diagrams
- Cutting edge chart & report creation
- Direct file output for ETL projects
- Enhancements for EDI data mapping
- And much more



The Altova MissionKit includes multiple tools for software architects:

UModel® – UML tool for software modeling

- Support for all UML diagram types, SQL database diagrams, BPMN, SysML
- Reverse engineering and code generation in Java, C#, VB.NET

XMLSpy® – XML editor & development environment

- Support for all XML-based technologies
- Royalty-free Java, C#, C++ code generation

MapForce® – graphical data mapping tool

- Mapping between XML, databases, EDI, flat files, XBRL, Excel 2007+, Web services
- Royalty-free Java, C#, C++ code generation

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**

Figure 4 **GetBlobsCompleted Method**

```
IEnumerable<StorageImage> ImageList = from Blob in xDoc.Descendants("Blob")
select new StorageImage()
  {
    Name = Blob.Descendants("Name").First().Value,
    Url = Blob.Descendants("Url").First().Value,
    Image = new BitmapImage(
      new Uri(Blob.Descendants("Url").First().Value, UriKind.Absolute))
  };

MainPage.ImageInfoList.Clear();
foreach (StorageImage CurImg in ImageList)
{
  MainPage.ImageInfoList.Add(CurImg);
}
```

Next, I need to handle the response in each example. I'll go through each one to make each of the samples as clear as possible. Starting with the Windows Phone 7 example, I implement the GetBlobsCompleted method. I use LINQ syntax to tease the XML into an enumerable list of my custom object, as shown in **Figure 4**.

I have to move the objects into an ObservableCollection<StorageImage>. I first clear the list, then loop through the LINQ result and add the objects to my ObservableCollection. Because the collection is in the DataContext and the ListBox is bound to the DataContext, it automatically picks up the changes and updates the contents of the ListBox.

Moving on to Java/Android, I don't have something as nice as LINQ to use, so I'm stuck doing more work teasing the data out of the DOM manually. The method MarshalToBlobInfos will take an XML Document object and parse it, building an ArrayList of BlobInfos.

This looks much like any other DOM-style access in .NET, JavaScript and so on, as you can see in **Figure 5**.

Finally, I write the jQuery used to work through the result and render the needed HTML. It's by far the shortest bit of code. But, depending on comfort level, it may be the most obtuse:

```
function RenderData(xml) {

  $(xml).find("Blob").each(
    function () {
      $("#output").append("<img margin=50 height=70 width=60 src=" +
      $(this).find("Url").text() +
      " title=" + $(this).find("Url").text() + " id=" +
      $(this).find("Name").text() + ") /> ");

  });
$("img").each(
  function () {
    $(this).click(
      function () {
        $("#zoomview").attr("src", $(this).attr("src"));
      });
  });
}
```

This asks for each "Blob" element of the XML; for each I write an <img /> tag into the element, whose id is assigned the value of "output." I use the similar selector construct to pull out the value of each element within the current "Blob" element needed to properly render the <img /> tag. I then create a selector for each resulting <img /> element and add a click handler to it.

So far my Silverlight code makes a REST call to Windows Azure Storage, parses the result and displays the filmstrip. The final step is to add a click handler for the filmstrip to show the zoom view of the image. I add a handler for SelectionChanged that grabs the currently selected StorageImage and assigns the Image control Source property the value of its Image property:

Figure 5 **The MarshalToBlobInfos Method**

```
public ArrayList<BlobInfo> MarshalToBlobInfos(Document doc)
  {
    ArrayList<BlobInfo> returnBlobs = new ArrayList<BlobInfo>();

  try {
    Element root = doc.getDocumentElement();
    NodeList items = root.getElementsByTagName("Blob");
    for (int idxBlobs=0;idxBlobs<items.getLength();idxBlobs++){
      BlobInfo blob = new BlobInfo();
      Node item = items.item(idxBlobs);
      NodeList blobProperties = item.getChildNodes();
      for (int idxProps=0;idxProps<blobProperties.getLength();idxProps++){
        Node property = blobProperties.item(idxProps);
        String name = property.getNodeName();
          if (name.equalsIgnoreCase("Name"))
            blob.Name = property.getFirstChild().getNodeValue();
          } else if (name.equalsIgnoreCase("Url")){
              blob.Url = property.getFirstChild().getNodeValue();
          }
      }
      returnBlobs.add(blob);
    }
  } catch (Exception e) {
      throw new RuntimeException(e);
  }
  return returnBlobs;
  }
```

```
private void ImageList_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
  ListBox lb = (ListBox)sender;
  StorageImage img = (StorageImage) lb.SelectedItem;
  ImageViewer.Source = img.Image;
}
```

That's it for Silverlight on Windows Phone 7. Android will take more work.

Thanks to the REST interface of Windows Azure Storage and the straightforward XML response, it's simple to make the call and parse the results whether I'm using .NET, JavaScript or, in this case, Java on Android. With the two main supporting methods in place, I'll add the code to the onCreate method of the Activity. Within the Activity, after the line to restore existing state, I'll add the lines of code to make the call to Windows Azure Storage and parse the results:

```
InputStream blobstream = GetBlobs();
DocumentBuilder docBuilder =
  DocumentBuilderFactory.newInstance().newDocumentBuilder();
Document doc = docBuilder.parse(blobstream);
BlobList = MarshalToBlobInfos(doc);
```

If the code were to run at this point I'd have no UI, but I want the filmstrip from the Windows Phone 7 example. The next step is to hook up the UI elements. I'll call setContentView, passing in a resource id. In this case, it's the resource id for the main.xml.

If I needed to change the view of an activity, I could call it passing in a different resource. In the previous example of the main.xml markup, the Gallery and the ImageView have the ids of Thumbnails and ZoomView, respectively. I'll use those ids to reference the widgets (controls) and assign a local variable for each reference. I'll place the following code right after the previous code:

```
setContentView(R.layout.main);
mZoomView = (ImageView)findViewById(R.id.ZoomView);
mThumbnails = (Gallery) findViewById(R.id.Thumbnails);
```

Now I need to create an adapter to help retrieve, format and lay out each item in the collection. Before I can set the adapter for the Thumbnails Gallery I have to create one. I add a new class to the

Schema Compare

Compare and synchronize SQL Server schemas

devart

www.devart.com

Advanced *Intellisense* with SQL formatting for SQL Server Management Studio

dbForge for SQL Server

development and management tools

SQL Complete

Data Studio

Explore and analyze large data sets in SQL Server databases

Data Compare

Query Builder

Create any queries visually, without code typing

Compare and synchronize table data of SQL Server

Figure 6 **The getView Implementation**

```
public View getView(int position, View convertView, ViewGroup parent)
{
  ImageView imageView;
  if (convertView == null)
  {
    imageView = new ImageView(mContext);
    imageView.setLayoutParams(new Gallery.LayoutParams (85, 70));
    imageView.setScaleType(ImageView.ScaleType.FIT_XY);
    imageView.setPadding(4, 4, 4, 4);
  }
  else
  {
    imageView = (ImageView) convertView;
  }
  BlobInfo CurrentBlob = mBlobList.get(position);

  if (CurrentBlob.ImageBitmap == null)
  {
    Bitmap bm = getImageBitmap(CurrentBlob.Url);
    imageView.setImageBitmap(bm);
    CurrentBlob.ImageBitmap = bm;
  }
  else
  {
    imageView.setImageBitmap(CurrentBlob.ImageBitmap)
  }
  return imageView;
}
```

project named ImageAdapter, which extends BaseAdapter. When I set it as the adapter for the Activity it should call count and subsequently call getView for each item passing in the position of the item, which I will use as the ArrayList index. I'll pass a pointer to the Activity into the ImageAdapter, because I'll need contextual information—in particular, I'll need the ArrayList that represents the images. The declaration and constructor for the ImageAdapter look like this:

```
public class ImageAdapter extends BaseAdapter
{
  private Context mContext;
  private ArrayList<BlobInfo> mBlobList=null;
  public ImageAdapter(Context c)
  {
    mContext = c;
    mBlobList = ((AzureImageViewer) mContext).BlobList;

  }
  public int getCount()
  {
    int count = mBlobList.size();
    return count;

  }
```

In the getView method, implemented as part of the adapter, I fetch the image, create a thumbnail and save the original image to the related BlobInfo object. For an actual implementation, one would only want to pre-fetch a number of images and clean them up as the user scrolled, keeping a moving window of available images based on the position of the filmstrip. The thumbnails would need to be fetched by a background thread much like the pattern required in the Silverlight and jQuery examples.

Figure 7 **Fetching the Image**

```
private Bitmap getImageBitmap(String url) {
  Bitmap ImageBitmap = null;
  try {
    URL ImageURL = new URL(url);
    URLConnection ImageConnection = ImageURL.openConnection();
    ImageConnection.connect();
    InputStream ImageStream = ImageConnection.getInputStream();
    BufferedInputStream BufferedStream = new BufferedInputStream(ImageStream);
    Log.e("Available buffer: ", String.valueOf(BufferedStream.available()));
    ImageBitmap = BitmapFactory.decodeStream(BufferedStream);
    BufferedStream.close();
    ImageStream.close();
  } catch (Exception e) {
    Log.e("Error getting bitmap", e.getMessage());
  }
  return ImageBitmap;
}
```

Additionally, the fetching code for the item specifically might be better placed in the getItem override. With those caveats, **Figure 6** shows the getView implementation.

The initial state of the view is done by the first conditional statement by either configuring it or by using a view passed to the function. By the time the if-else is passed, the view is ready—but I still don't have the image. To help with that, I use the position parameter to fetch the appropriate BlobInfo object and then the URL property to fetch the image (in my case they're all .jpgs), as shown in **Figure 7**.

To fetch the image, I create a new connection based on the URL, then create a BufferedStream to pass to the BitmapFactory.decodeStream method. However, there seems to be an issue with decodeStream and using the BufferedStream object. For this sample I'm going to go ahead and use it, knowing that nothing I've tested that is 1MB or greater has worked; what's worse, it just silently fails.

The best way to avoid these issues is to write code to manually read the input stream, then pass that to the BitmapFactory object. Once the BitmapFactory has done its work, I return the bitmap, set the ImageBitmap and assign the bitmap to the related BlobInfo object ImageBitmap property. If I've already fetched it, I just use the current value in the BlobInfo object.

When it runs I should see a filmstrip of images; as of yet, though, I can't zoom into an image, and nothing happens upon selecting a thumbnail. So I'll add a click listener for the thumbnails and, based on the position of the selected item, reference the BlobInfo ArrayList and use the resulting BlobInfo ImageBitmap property as the parameter to the ZoomView (of type ImageView) setImageBitmap call:

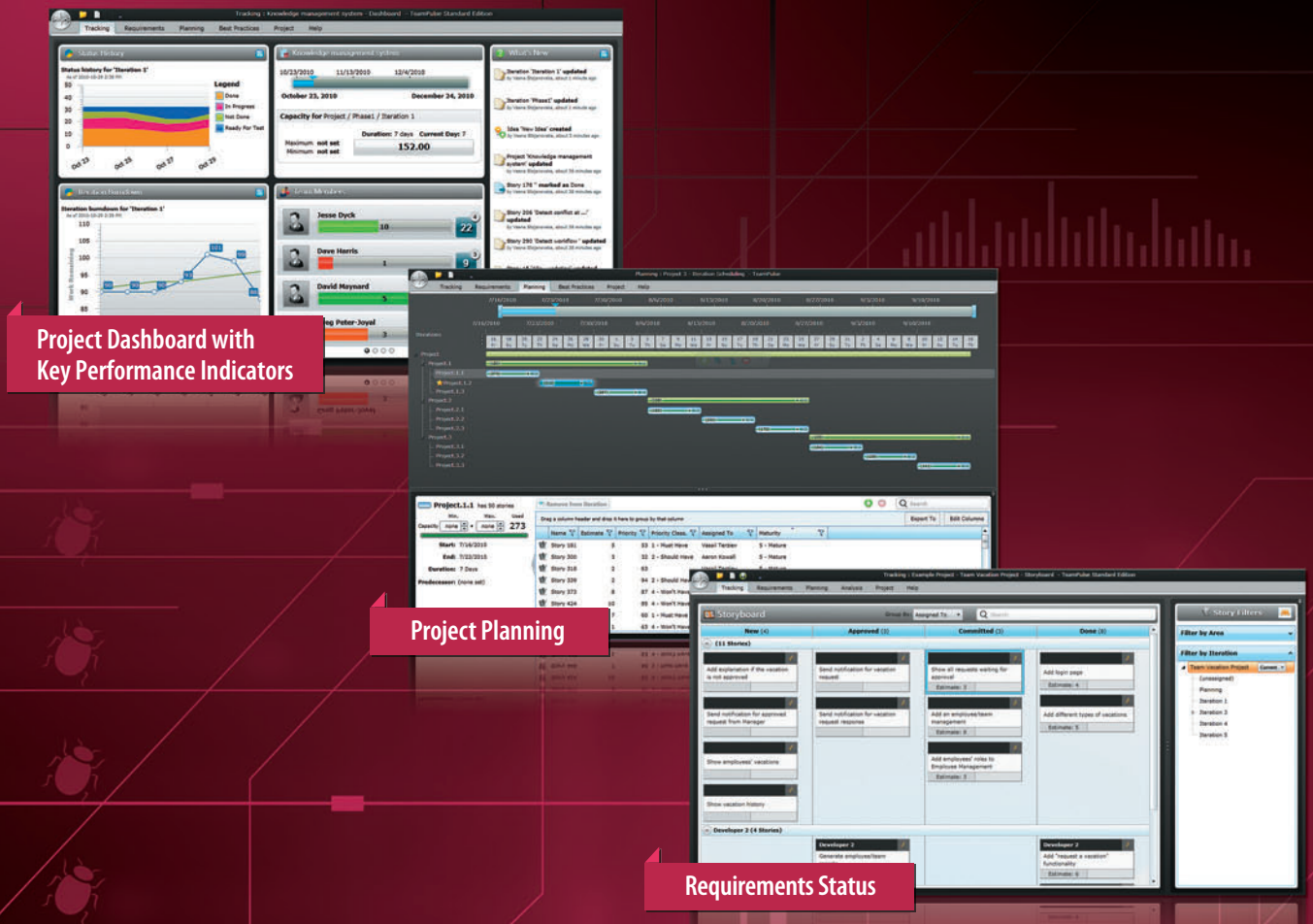
```
  mThumbnails.setOnItemClickListener(new OnItemClickListener()
{
    public void onItemClick (AdapterView<?> parent, View v, int position, long id)
    {
      BlobInfo CurrentBlob = BlobList.get(position);
      mZoomView.setImageBitmap(CurrentBlob.ImageBitmap);
    }
  });
```

Universal Donor

The Android sample was the heaviest implementation, as compared to the quick and simple Windows Phone 7 and jQuery examples. That could be due to my familiarity level, but looking at the code I had to write across the three, I think there's more to it than just comfort level. The fact is that using Windows Azure breathes life and power into applications, regardless of the device platform.

Interfacing with Windows Azure Storage couldn't be much simpler without having someone do it for you. In this case, the access came down to fetching a list via REST and then issuing a subsequent HTTP Get based on the content of the return to fetch the item directly. ■

Joseph Fultz *is a software architect at AMD, helping to define the overall architecture and strategy for portal and services infrastructure and implementations. Previously he was a software architect for Microsoft working with its top-tier enterprise and ISV customers defining architecture and designing solutions.*



Forget spreadsheets!

You deserve better tools to manage your software projects!

Meet **Telerik TeamPulse** – a new generation of highly visual project management tools that make it easier for you and your team to manage software development projects. TeamPulse allows your team to be more expressive in the way they **capture requirements**, **manage project plans**, **assign and track work**, and more importantly, be continually connected with each other. Without reams of flat, disconnected and complex spreadsheets TeamPulse supports your entire development lifecycle, from idea to release, by ensuring a constant flow of context, deep insight, and real-time collaboration.

Try now at: www.telerik.com/pulse

2010
Microsoft Central & Eastern Europe
PARTNER OF THE YEAR
Winner

telerik
deliver more than expected

Agile C++ Development and Testing with Visual Studio and TFS

John Socha-Leialoha

You're a developer or tester working on an application built in Visual C++. As a developer, wouldn't it be great if you could be more productive, produce higher-quality code, and rewrite your code as needed to improve the architecture without fear of breaking anything? And as a tester, wouldn't you love to spend less time writing and maintaining tests so you have time for other test activities?

In this article, I'll present a number of techniques that our team here at Microsoft has been using to build applications.

Our team is fairly small. We have 10 people working on three different projects at the same time. These projects are written in both C# and C++. The C++ code is mostly for the programs that must run inside Windows PE, which is a stripped-down version of Windows often used for OS installation. It's also used as part of a Microsoft System Center Configuration Manager task sequence to run tasks that can't run inside the full OS, such as capturing a hard disk to a virtual hard disk (VHD) file. That's a lot for a small team, so we need to be productive.

This article discusses:

- Agile development
- Test-driven development
- Gated check-ins
- Writing C++ unit tests
- Setup and teardown code
- C++ fixtures
- Code coverage

Technologies discussed:

C++, Team Foundation Server, Visual Studio 2010

Code download available at:

code.msdn.microsoft.com/mag201106Agile

Our team uses Visual Studio 2010 and Team Foundation Server (TFS) 2010. We use TFS 2010 for version control, work tracking, continuous integration, code-coverage gathering and reporting.

When and Why Our Team Writes Tests

I'll start by looking at why our team writes tests (the answers might be different for your team). The specific answer is a little different for our developers and testers, but perhaps not as different as you might at first think. Here are my goals as a developer:

- No build breaks
- No regressions
- Refactor with confidence
- Modify the architecture with confidence
- Drive design through test-driven development (TDD)

Of course, quality is the big "why" behind these goals. When these goals are met, life as a developer is a lot more productive and fun than when they're not.

For our testers, I'm going to focus on just one aspect of an Agile tester: writing automated tests. The goals for our testers when they write automated tests include no regressions, acceptance-driven development, and gathering and reporting code coverage.

Of course, our testers do much more than just write automated tests. Our testers are responsible for code-coverage gathering because we want code-coverage numbers to include the results from all tests, not just unit tests (more on this later).

In this article, I'm going to cover the different tools and techniques our team uses to achieve the goals stated here.

Eliminating Build Breaks with Gated Check-Ins

In the past, our team used branches to ensure that our testers always had a stable build to test. However, there is overhead associated with maintaining the branches. Now that we have gated check-ins, we only use branching for releases, which is a nice change.

Using gated check-ins requires that you've set up build control and one or more build agents. I'm not going to cover this topic here, but you can find details on the MSDN Library page, "Administering Team Foundation Build," at bit.ly/jzA8Ff.

Once you have build agents set up and running, you can create a new build definition for gated check-ins by following these steps from within Visual Studio:

1. Click View in the menu bar and click Team Explorer to ensure the Team Explorer tool window is visible.
2. Expand your team project and right-click Build.
3. Click New Build Definition.
4. Click Trigger on the left and select Gated Check-in, as shown in **Figure 1**.
5. Click Build Defaults and select the build controller.
6. Click Process and select the items to build.

Once you've saved this build definition—we called ours "Gated Checkin"—you'll see a new dialog box after you submit your check-in (see **Figure 2**). Clicking Build Changes creates a shelveset and submits it to the build server. If there are no build errors and all the unit tests pass, TFS will check in your changes for you. Otherwise, it rejects the check-in.

Gated check-ins are really nice because they ensure you never have build breaks. They also ensure that all unit tests pass. It's all too easy for a developer to forget to run all the tests before check-in. But with gated check-ins, that's a thing of the past.

Writing C++ Unit Tests

Now that you know how to run your unit tests as part of a gated check-in, let's look at one way you can write these unit tests for native C++ code.

I'm a big fan of TDD for several reasons. It helps me focus on behavior, which keeps my designs simpler. I also have a safety net in the form of tests that define the behavioral contract. I can refactor without fear of introducing bugs that are a result of accidentally violating the behavioral contract. And I know that some other developer won't break required behavior they didn't know about.

One of the developers on the team had a way to use the built-in test runner (mstest) to test C++ code. He was writing Microsoft .NET Framework unit tests using C++/CLI that called public functions exposed by a native C++ DLL. What I present in this section takes that approach much further, allowing you to directly instantiate native C++ classes that are internal to your production code. In other words, you can test more than just the public interface.

The solution is to put the production code into a static library that can be linked into the unit test DLLs as well as into the production EXE or DLL, as shown in **Figure 3**.

Here are the steps required to set up your projects to follow this procedure. Start by creating the static library:

1. In Visual Studio, click File, click New and click Project.
2. Click Visual C++ in the Installed Templates list (you'll need to expand Other Languages).
3. Click Win32 Project in the list of project types.
4. Enter the name of your project and click OK.
5. Click Next, click Static library and then click Finish.

Now create the test DLL. Setting up a test project requires a few more steps. You need to create the project, but also give it access to the code and header files in the static library.

Start by right-clicking the solution in the Solution Explorer window. Click Add, then click New Project. Click Test under the Visual C++ node in the template list. Type the name of the project (our team adds UnitTests to the end of the project name) and click OK.

Right-click the new project in Solution Explorer and click Properties. Click Common Properties in the tree on the left. Click Add New Reference. Click the Projects tab, select the project with your static library and click OK to dismiss the Add Reference dialog.

Expand the Configuration Properties node in the tree on the left, then expand the C/C++ node. Click General under the C/C++ node. Click the Configuration combo box and select All Configurations to ensure you change both Debug and Release versions.

Click Additional Include Libraries and enter a path to your static library, where you'll need to substitute your static library name for MyStaticLib:

```
$(SolutionDir)\MyStaticLib;%(AdditionalIncludeDirectories)
```

Click the Common Language Runtime Support property in the same property list and change it to Common Language Runtime Support (/clr).

Click on the General section under Configuration Properties and change the TargetName property to $(ProjectName). By default, this is set to DefaultTest for all test projects, but it should be the name of your project. Click OK.

You'll want to repeat the first part of this procedure to add the static library to your production EXE or DLL.

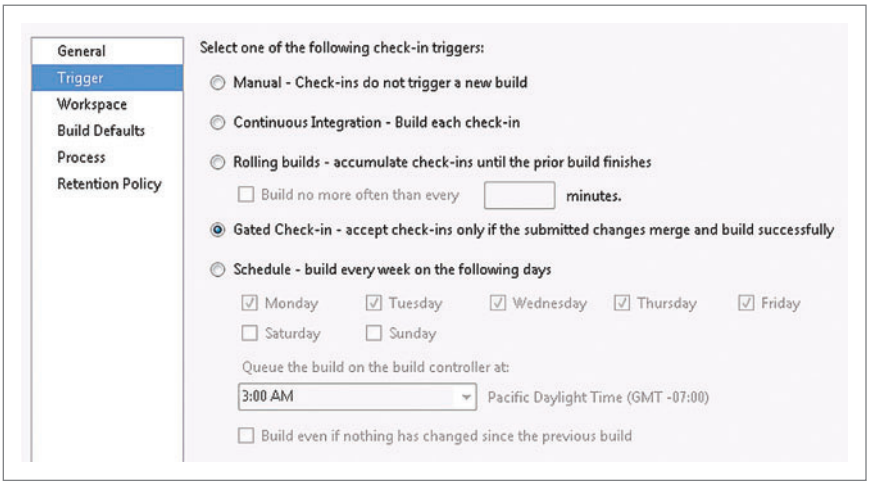


Figure 1 **Select the Gated Check-in Option for Your New Build Definition**

Writing Your First Unit Test

You should now have everything you need in order to write a new unit test. Your test methods will be .NET methods written in C++, so the syntax will be a little different than native C++. If you know C#, you'll find it's a blend between C++ and C# syntax in many ways. For more details, check out the MSDN Library documentation, "Language Features for Targeting the CLR," at bit.ly/iOKbR0.

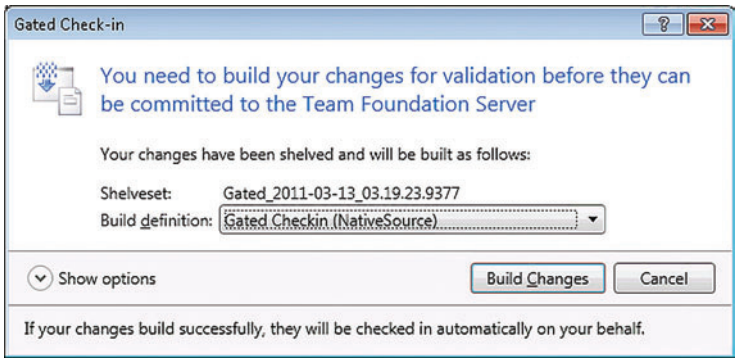


Figure 2 **Gated Check-in Dialog Box**

Let's say you have a class definition you're going to test that looks something like this:

```
#pragma once
class MyClass {
  public:
    MyClass(void);
    ~MyClass(void);

    int SomeValue(int input);
};
```

Now you want to write a test for the SomeValue method to specify behavior for this method. **Figure 4** shows what a simple unit test might look like, showing the entire .cpp file.

If you aren't familiar with writing unit tests, I'm using a pattern known as Arrange, Act, Assert. The Arrange part sets up the preconditions for the scenario you want to test. Act is where you call the method you're testing. Assert is where you check that the method behaved the way you want. I like to add a comment in front of each section for readability, and to make it easy to find the Act section.

Test methods are marked by the TestMethod attribute, as you can see in **Figure 4**. These methods, in turn, must be contained inside a class marked with the TestClass attribute.

Notice that the first line in the test method creates a new instance of the native C++ class. I like to use the unique_ptr standard C++ library class to ensure this instance is deleted automatically at the end of the test method. Therefore, you can clearly see that you can mix native C++ with your CLI/C++ .NET code. There are, of course, restrictions, which I'll outline in the next section.

Again, if you haven't written .NET tests before, the Assert class has a number of useful methods you can use to check different conditions. I like to use the generic version to be explicit about the data type I expect from the result.

Taking Full Advantage of C++/CLI Tests

As I mentioned, there are some limitations you'll need to be aware of when you mix native C++ code with C++/CLI code. The differences are a result of the difference in memory management between the two code bases. Native C++ uses the C++ new operator to allocate memory, and you're responsible for freeing that memory yourself. Once you allocate a piece of memory, your data will always be in the same place.

On the other hand, pointers in C++/CLI code have a very different behavior because of the garbage-collection model it inherits from the .NET Framework. You create new .NET objects in C++/CLI using the gcnew operator instead of the new operator, which returns an object handle, not a pointer to the object. Handles are basically pointers to a pointer. When the garbage collection moves managed objects around in memory, it updates the handles with the new location.

You have to be very careful when mixing managed and native pointers. I'll cover some of these differences, and give you tips and tricks to get the most out of C++/CLI tests for native C++ objects.

Let's say you have a method you want to test that returns a pointer to a string. In C++ you might represent the string pointer with LPCTSTR. But a .NET string is represented by String^ in C++/CLI. The caret after the class name signifies a handle to a managed object.

Here's an example of how you might test the value of a string returned by a method call:

```
// Act
LPCTSTR actual = pSomething->GetString(1);

// Assert
Assert::AreEqual<String^>("Test", gcnew String(actual));
```

The last line contains all the details. There's an AreEqual method that accepts managed strings, but there's no corresponding method for native C++ strings. As a result, you need to use managed strings. The first parameter to the AreEqual method is a managed string, so it's actually a Unicode string even though it's not marked as a Unicode string using _T or L, for example.

The String class has a constructor that accepts a C++ string, so you can create a new managed string that will contain the actual value from the method you're testing, at which point AreEqual ensures they're the same value.

The Assert class has two methods that might look very attractive: IsNull and IsNotNull. However, the parameter for these methods is a handle, not an object pointer, which means you can only use them with managed objects. Instead, you can use the IsTrue method, like this:

```
Assert::IsTrue(pSomething != nullptr, "Should not be null");
```

This accomplishes the same thing, but with slightly more code. I add a comment so the expectation is clear in the message that appears in the test results window, which you can see in **Figure 5**.

Sharing Setup and Teardown Code

Your test code should be treated like production code. In other words, you should refactor tests just as much as production code in order to keep the test code easier to maintain. At some point you may have some common setup and teardown code for all the test methods in a test class. You can designate a method that will run before each test, as well as a method that runs after each test (you can have just one of these, both or neither).

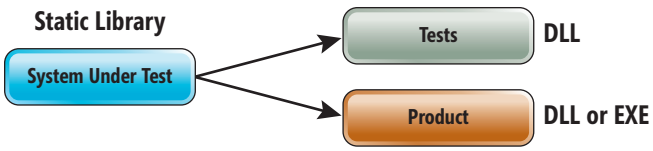


Figure 3 **The Tests and Product Share the Same Code via a Static Library**

Figure 4 **A Simple Unit Test**

```
#include "stdafx.h"
#include "MyClass.h"
#include <memory>
using namespace System;
using namespace Microsoft::VisualStudio::TestTools::UnitTesting;

namespace MyCodeTests {
  [TestClass]
  public ref class MyClassFixture {
    public:
      [TestMethod]
      void ShouldReturnOne_WhenSomeValue_GivenZero() {
        // Arrange
        std::unique_ptr<MyClass> pSomething(new MyClass);

        // Act
        int actual = pSomething->SomeValue(0);

        // Assert
        Assert::AreEqual<int>(1, actual);
      }
  };
}
```

The TestInitialize attribute marks a method that will be run before each test method in your test class. Likewise, TestCleanup marks a method that runs after each test method in your test class. Here's an example:

```
[TestInitialize]
void Initialize() {
  m_pInstance = new MyClass;
}

[TestCleanup]
void Cleanup() {
  delete m_pInstance;
}

MyClass *m_pInstance;
```

First, notice that I used a simple pointer to the class for m_pInstance. Why didn't I use unique_ptr to manage the lifetime?

The answer, again, has to do with mixing native C++ and C++/CLI. Instance variables in C++/CLI are part of a managed object, and therefore can only be handles to managed objects, pointers to native objects or value types. You have to go back to the basics of new and delete to manage the lifetime of your native C++ instances.

Using Pointers to Instance Variables

If you're using COM, you might run into a situation where you want to write something like this:

```
[TestMethod]
Void Test() {
  ...
  HRESULT hr = pSomething->GetWidget(&m_pUnk);
  ...
}

IUnknown *m_pUnk;
```

This won't compile, and it will produce an error message like this:

```
cannot convert parameter 1 from 'cli::interior_ptr<Type>' to 'IUnknown **'
```

The address of a C++/CLI instance variable has the type interior_ptr<IUnknown *> in this case, which isn't a type compatible with native C++ code. Why, you ask? I just wanted a pointer.

The test class is a managed class, so instances of this class can be moved in memory by the garbage collector. So if you had a pointer to an instance variable, and then the object moved, the pointer would become invalid.

You can lock the object for the duration of your native call like this:

```
cli::pin_ptr<IUnknown *> ppUnk = &m_pUnk;
HRESULT hr = pSomething->GetWidget(ppUnk);
```

The first line locks the instance until the variable goes out of scope, which then allows you to pass a pointer to the instance variable to a native C++, even though that variable is contained inside a managed test class.

Writing Testable Code

At the beginning of this article I mentioned the importance of writing testable code. I use TDD to ensure my code is testable, but some developers prefer to write tests soon after they write their code. In either case, it's important to think not just about unit tests, but about the entire test stack.

Mike Cohn, a well-known and prolific author on Agile, has drawn a test-automation pyramid that provides an idea of the types of tests and how many should be at each level. Developers should write all or most of the unit and component tests, and perhaps some integration tests. For details about this pyramid of testing, read Cohn's blog post, "The Forgotten Layer of the Test Automation Pyramid" (bit.ly/eRZU2p).

Testers are typically responsible for writing acceptance and UI tests. These are also sometimes called end-to-end, or E2E, tests. In Cohn's pyramid, the UI triangle is smallest compared with the areas for the other types of tests. The idea is that you want to write as few automated UI tests as you can. Automated UI tests tend to be fragile and expensive to write and maintain. Small changes to the UI can easily break UI tests.

If your code isn't written to be testable, you can easily end up with an inverted pyramid, where most of the automated tests are UI tests. This is a bad situation, but the bottom line is that it's a developer's job to ensure that testers can write integration and acceptance tests below the UI.

Additionally, for whatever reason, most of the testers I've run across are very comfortable writing tests in C#, but shy away from writing tests in C++. As a result, our team needed a bridge between the C++ code under test and the automated tests. The bridge is in the form of fixtures, which are C++/CLI classes that appear to the C# code to be just like any other managed class.

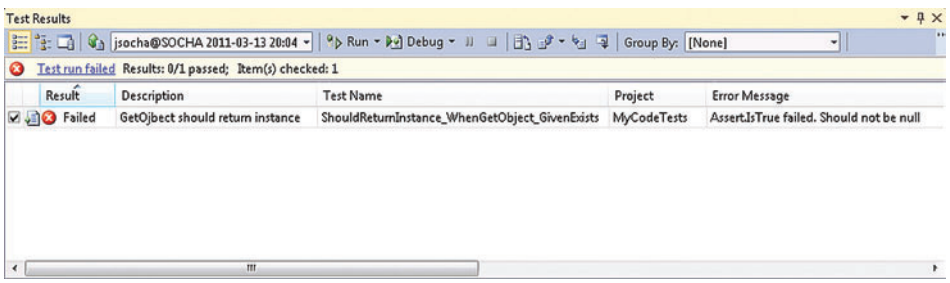


Figure 5 **Test Results Showing the Additional Comment in the Error Message**

Building C# to C++ Fixtures

The techniques here aren't much different from the ones I covered for writing C++/CLI tests. They both use the same type of mixed-mode code. The difference is how they're used in the end.

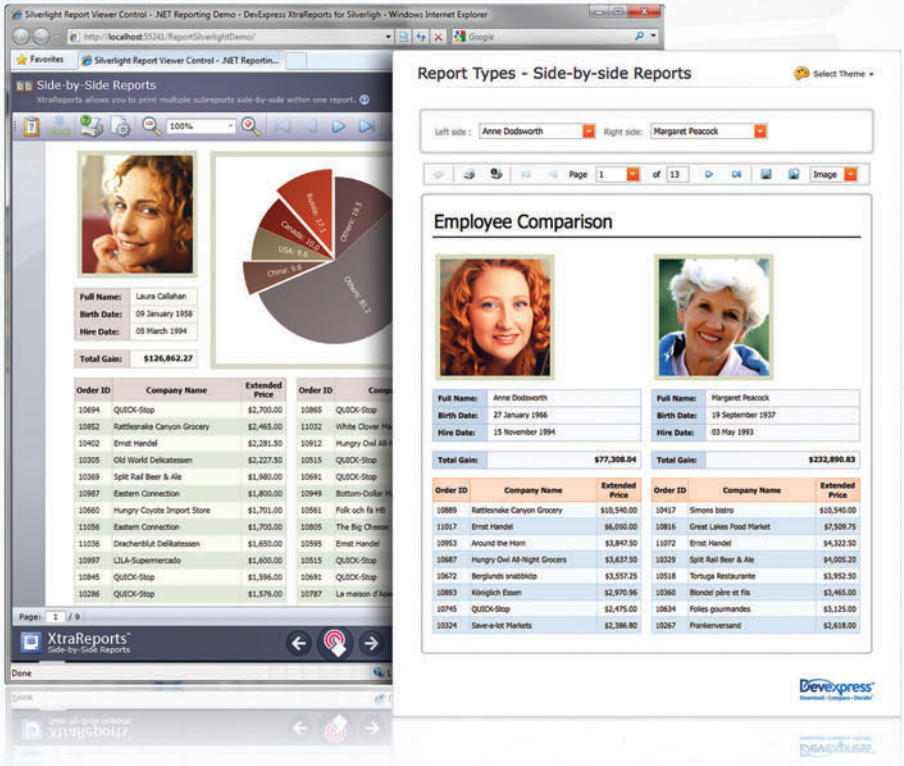
ca·pa·ble (rə'pôrting)

– *adjective*

1. having power and ability; efficient; competent
2. having the tools required for a specific task

– *synonyms*

accomplished, efficient, experienced



A Reporting Library should be able to accommodate the structure of your data. Our banding system allows for the skillful construction of elegant reports that conform to you and your data–without imposing restrictions on your ability to address customer needs in the shortest possible time.

Experience the power of the XtraReports Suite today.
Download your FREE trial – www.DevExpress.com/FreeEval.

Devexpress™
Download • Compare • Decide!

Figure 6 **C# to C++ Test Fixture**

```
#include "stdafx.h"
#include "MyClass.h"
using namespace System;

namespace MyCodeFixtures {
  public ref class MyCodeFixture {
    public:
      MyCodeFixture() {
        m_pInstance = new MyClass;
      }

      ~MyCodeFixture() {
        delete m_pInstance;
      }

      !MyCodeFixture() {
        delete m_pInstance;
      }

      int DoSomething(int val) {
        return m_pInstance->SomeValue(val);
      }

      MyClass *m_pInstance;
  };
}
```

The first step is to create a new project that will contain your fixtures:

1. Right-click the solution node in Solution Explorer, click Add and click New Project.
2. Under Other Languages, Visual C++, CLR, click Class Library.
3. Enter the name to use for this project, and click OK.
4. Repeat the steps for creating a test project to add a reference and the include files.

The fixture class itself will look somewhat similar to the test class, but without the various attributes (see **Figure 6**).

Notice that there's no header file! This is one of my favorite features of C++/CLI. Because this class library builds a managed assembly, information about classes is stored as .NET-type information, so you don't need header files.

This class also contains both a destructor and a Finalizer. The destructor here really isn't the destructor. Instead, the compiler rewrites the destructor into an implementation of the Dispose method in the IDisposable interface. Any C++/CLI class that has a desctructor, therefore, implements the IDisposable interface.

The !MyCodeFixture method is the finalizer, which is called by the garbage collector when it decides to free this object, unless you previously called the Dispose method. You can either employ the using statement to control the lifetime of your embedded native C++ object, or you can let the garbage collector handle the lifetime. You can find more details about this behavior in the MSDN Library article, "Changes in Destructor Semantics" at bit.ly/kW8knr.

Figure 7 **A C# Unit Testing System**

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using MyCodeFixtures;

namespace MyCodeTests2 {
  [TestClass]
  public class UnitTest1 {
    [TestMethod]
    public void TestMethod1() {
      // Arrange
      using (MyCodeFixture fixture = new MyCodeFixture()) {
        // Act
        int result = fixture.DoSomething(1);

        // Assert
        Assert.AreEqual<int>(1, result);
      }
    }
  }
}
```

Once you have a C++/CLI fixture class, you can write a C# unit test that looks something like **Figure 7**.

I like employing a using statement to explicitly control the lifetime of the fixture object instead of relying on the garbage collector. This is especially important in test methods to ensure that tests don't interact with other tests.

Capturing and Reporting Code Coverage

The final piece I outlined at the start of this article is code coverage. My team's goal is to have code coverage automatically captured by the build server, published to TFS and easily available to everyone.

My first step was to find out how to capture C++ code coverage from running tests. Searching the Web, I found an informative blog post by Emil Gustafsson titled "Native C++ Code Coverage Reports Using Visual Studio 2008 Team System" (bit.ly/eJ5cqv). This post shows the steps that are required to capture code-coverage information. I turned this into a CMD file I can run at any time on my development machine to capture code-coverage information:

```
"%VSINSTALLDIR%\Team Tools\Performance Tools\vsinstr.exe" Tests.dll /COVERAGE
"%VSINSTALLDIR%\Team Tools\Performance Tools\vsperfcmd.exe" /
START:COVERAGE /WaitStart /OUTPUT:coverage
mstest /testcontainer:Tests.dll /resultsfile:Results.trx
"%VSINSTALLDIR%\Team Tools\Performance Tools\vsperfcmd.exe" /SHUTDOWN
```

You'll want to replace *Tests.dll* with the actual name of your DLL that contains tests. You'll also need to prepare your DLLs to be instrumented:

1. Right-click the test project in the Solution Explorer window.
2. Click Properties.
3. Select the Debug configuration.
4. Expand Configuration Properties, then expand Linker and click Advanced.
5. Change the Profile property to Yes (/PROFILE).
6. Click OK.

These steps enable profiling, which you need turned on in order to instrument the assemblies so you can capture code-coverage information.

Rebuild your project and run the CMD file. This should create a coverage file. Load this coverage file into Visual Studio to ensure you're able to capture code coverage from your tests.

Performing these steps on the build server and publishing the results to TFS requires a custom build template. TFS build templates are stored in version control and belong to a specific team project. You'll find a folder called BuildProcessTemplates under each team project that will most likely have several build templates.

To use the custom build template included in the download, open the Source Control Explorer window. Navigate to the BuildProcessTemplates folder in your team project and ensure you have it mapped to a directory on your computer. Copy the BuildCCTemplate.xaml file into this mapped location. Add this template to source control and check it in.

Template files must be checked in before you can use them in build definitions.

flex·i·ble (rə'pôrting)

– *adjective*

1. easily modified; adaptable
2. demonstrating the ability to adapt to new and different requirements

– *synonyms*

adaptable, modifiable, adjustable

A Reporting Library must be flexible so as to fully accommodate your platform requirements. The XtraReports Suite allows you to target WinForms, ASP.NET, WPF or Silverlight–without the need to re-create individual reports from scratch.

Experience the power of the XtraReports Suite today.
Download your FREE trial – www.DevExpress.com/FreeEval.

DevExpress™
Download • Compare • Decide!

Now that you have the build template checked in, you can create a build definition to run code coverage. C++ code coverage is gathered using the vsperfmd command, as shown earlier. Vsperfmd listens for code-coverage information for all instrumented executables that are run while vsperfcmd is running. Therefore, you don't want to have other instrumented tests run at the same time. You should also ensure you have only one build agent running on the machine that will process these code-coverage runs.

I created a build definition that would run nightly. You can do the same by following these steps:

1. In the Team Explorer window, expand the node for your team project.
2. Right-click Builds, which is a node under your team project.
3. Click New Build Definition.
4. In the Trigger section, click Schedule and select the days on which you want to run code coverage.
5. In the Process section, click Show details in the section called Build process template at the top and then select the build template you checked into source control.
6. Fill out the other required sections and save.

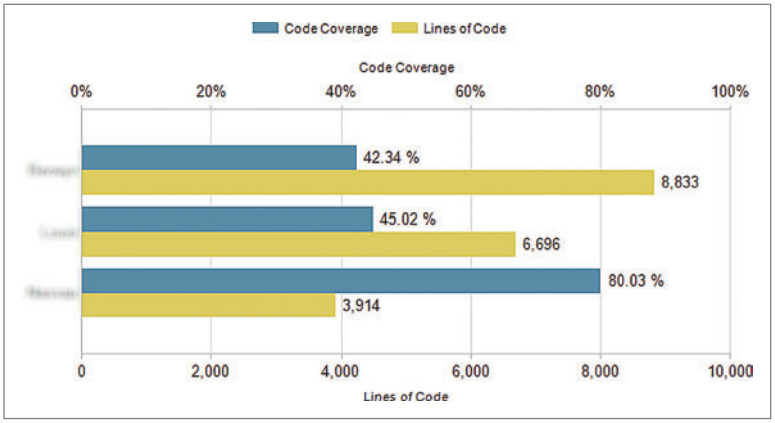


Figure 8 **The Code-Coverage Report**

Adding a Test Settings File

The build definition will also need a test settings file. This is an XML file that lists the DLLs for which you want to capture and publish results. Here are the steps to set up this file for code coverage:

1. Double-click the Local.test settings file to open the Test Settings dialog box.
2. Click Data and Diagnostics in the list on the left side.
3. Click Code Coverage and check the check box.
4. Click the Configure button above the list.
5. Check the box next to your DLL that contains your tests (which also contains the code the tests are testing).
6. Uncheck Instrument assemblies in place, because the build definition will handle this.
7. Click OK, Apply and then Close.

If you want to build more than one solution or you have more than one test project, you'll need a copy of the test settings file that includes the names of all the assemblies that should be monitored for code coverage.

To do that, copy the test settings file to the root of your branch and give it a descriptive name, such as CC.testsettings. Edit the XML. The file will contain at least one CodeCoverageItem element from the previous steps. You'll want to add one entry for each DLL you want captured. Note that the paths are relative to the location of the project file, not the location of the test settings file. Check this file into source control.

Finally, you need to modify the build definition to use this test settings file:

1. In the Team Explorer window, expand the node for your team project, then expand Builds.
2. Right-click the build definition you created earlier.
3. Click Edit Build Definition.
4. In the Process section, expand Automated Tests, then
1. Test Assembly and click on TestSettings File. Click the … button and select the test settings file we created earlier.
5. Save your changes.

You can test this build definition by right-clicking and selecting Queue New Build to start a new build right away.

Reporting Code Coverage

I created a custom SQL Server Reporting Services report that displays code coverage, as shown in **Figure 8** (I've blurred the names of actual projects to protect the guilty). This report uses a SQL query to read data in the TFS warehouse and display the combined results for both C++ and C# code.

I won't go into all of the details on how this report works, but there are a couple of aspects I do want to mention. The database contains too much information from the C++ code coverage for two reasons: test method code is included in the results and standard C++ libraries (that are in header files) are included in the results.

I added code in the SQL query that filters out this extra data. If you look at the SQL inside the report, you'll see this:

```
and CodeElementName not like 'std::%'
and CodeElementName not like 'stdext::%'
and CodeElementName not like '`anonymous namespace'':%'
and CodeElementName not like '_bstr_t%'
and CodeElementName not like '_com_error%'
and CodeElementName not like '%Tests::%'
```

These lines exclude code-coverage results for specific namespaces (std, stdext and anonymous) and a couple of classes shipped with Visual C++ (_bstr_t and _com_error), as well as any code that's inside a namespace that ends with Tests.

The latter, excluding namespaces that end with Tests, excludes any methods that are in test classes. When you create a new test project, because the project name ends with Tests, all test classes by default will be within a namespace that ends with Tests. You can add other classes or namespaces here that you want excluded.

I've only scratched the surface of what you can do—be sure to follow our progress on my blog at blogs.msdn.com/jsocha. ■

JOHN SOCHA-LEIALOHA *is a developer in the Management Platforms & Service Delivery group at Microsoft. His past achievements include writing the Norton Commander (in C and assembler) and writing "Peter Norton's Assembly Language Book" (Brady, 1987).*

THANKS to the following technical expert for reviewing this article: *Rong Lu*

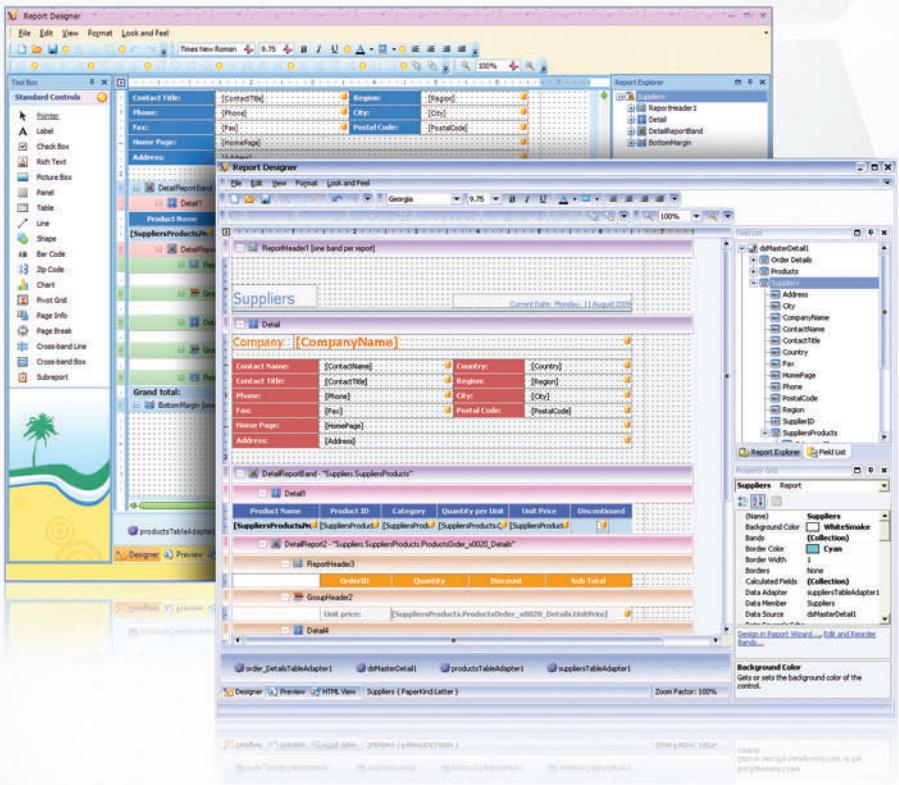
de·pend·a·ble (rə'pôrting)

– *adjective*

1. trustworthy; reliable
2. able to rely upon an individual or organization in order to achieve your goals

– *synonyms*

solid, faithful, responsible



Though the award-winning XtraReports Suite delivers on its promise of multi-platform reporting, technical questions and issues will undoubtedly arise. Our support team is second to none when it comes to timeliness and quality–we are your extended team and want to do everything possible to ensure your success in the marketplace.

Experience the power of the XtraReports Suite today.
Download your FREE trial – www.DevExpress.com/FreeEval.

Make Agile Work for You in TFS 2010

Chris Adams

This is the story of one team's road to Agile using Team Foundation Server (TFS) 2010.

In the Agile Manifesto, there are several keywords that highlight how Agile teams should work together. They include valuing individuals (and their interactions) over processes and tools. Teams use these values as one of the reasons for moving to Agile development, among others. In the past 10 or so years since the manifesto, various flavors of Agile were developed. I'll talk more about a couple of options within TFS 2010 that offer the opportunity to adopt one of two flavors of Agile process templates, and how our team at Microsoft uses them.

The key focus of our software development team here at Microsoft when it was reorganized a few years back was to ensure we used tools that you, our customers, can use. This prevented us from utilizing internal tools that have existed for years at Microsoft. Our development team, which is in the Management & Security Division (MSD), is focused on building software that enables Microsoft to provide services to a breadth of users: IT users, IT administrators and engineers, and online service operators. In short, our team is potentially just like yours—building software that enhances or improves the businesses on which customers rely.

Prior to our adoption of Agile, we didn't value customer feedback, had artificial boundaries between team members due to titles and were unable to deliver accurate status to our management team. If we didn't change, our overall effectiveness would dwindle until we failed completely.

In our shift to Agile, we wanted to focus on interactions with one another, our customers and, finally, our process. Prior to this change, we spent time focused on the reverse—process, customers and then interactions. The result over the past two years is that our focus today is on empowering our team, restoring our focus on customers and executing on the right solutions at the right time. Let me share how.

This article discusses:

- An introduction to MSF Agile
- Utilizing user stories to drive business value
- Planning iterations in MSF Agile
- Product planning and iteration backlog tracking tools
- Team-specific customizations
- Using TFS Integration with Windows SharePoint Services
- SharePoint and TFS 2010 projects
- Agile retrospectives
- The switch to Scrum 1.0
- Using the iteration workbook
- Assigning work to team members

Technologies discussed:

Team Foundation Server 2010, Visual Studio Scrum 1.0, Microsoft Solutions Framework for Agile Software Development version 5.0

What Worked for Us Might Work for You

We aren't a unique software team. We're a young team striving to develop the highest-quality software that leads directly to business value. None of us have been on this team for more than three years. And as I mentioned, we were struggling.

In March 2010, we decided that things had to change. We needed a system that worked with us, rather than against us, and allowed us to focus on delivering customer value. Beyond this, the system had to provide the ability to monitor and track progress at various levels throughout our team and our larger organization.

We wanted to develop in an Agile way, but we didn't know how.

Our Introduction to MSF Agile Version 5.0

We were introduced to TFS 2010 early in the beta stage and decided to create one of our projects using Microsoft Solutions Framework Agile version 5.0 (hereafter MSF Agile for brevity). We were looking for a process template that would help us execute on our core objectives: enable effective product and sprint planning; focus on getting a rhythm to our development; and, most importantly, increase our interaction between developers, testers and program

trans·par·ent (rə'pôrting)

– adjective

1. free from the unknown; open; detectable
2. delivering without secrecy and vagueness

– synonyms

clear, visible

Unlike competing products which do not ship 100% of the source code when purchased, the XtraReports Suite is fully transparent and not a black box with a myriad of unknowns. When you integrate DevExpress Reporting, you can rest assured that every single line of code is available for review.

Experience the power of the XtraReports Suite today.
Download your FREE trial – www.DevExpress.com/FreeEval.

Devexpress™
Download • Compare • Decide!

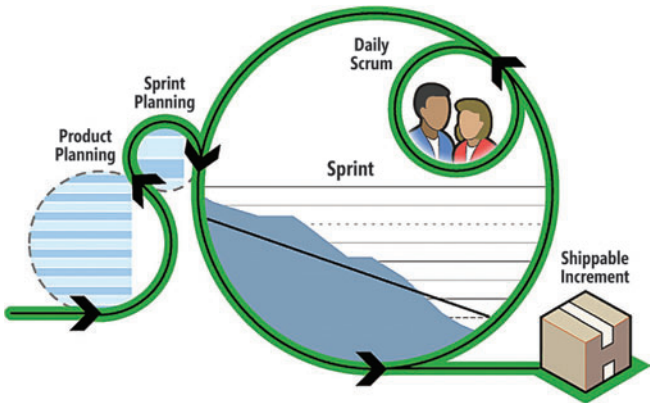


Figure 1 **The MSF Agile Process**

managers (PMs). The lifecycle (see **Figure 1**) provided to us in MSF Agile seemed to fit exactly what we were looking for.

MSF Agile provides several work item types (WITs) to help guide us into Agile. We first examined each WIT (as defined in **Figure 2**) to understand how to use them effectively.

A key point I'd like to make is, although we studied each WIT, we didn't use all of them at first. Instead, we focused most of our attention on user stories, tasks and bugs. It wasn't until several months later—or even a year later—that we started introducing some of the other WITs such as test cases. To date, our team still doesn't use the issue WIT, but that doesn't mean it wouldn't work for you. Like many teams learning Agile, we used what we liked and discarded what we didn't. The key point I want to make is that not everyone fully adopts Agile software development, and TFS 2010 provides this flexibility with the MSF Agile template.

Utilizing User Story WITs to Drive Business Value

We learned that an Agile team should spend significant effort on user stories, but we didn't learn this overnight. At first, our user stories were quite large and spanned multiple iterations. They were more themes that drove tasks than user stories that drove incremental value.

After some time, we learned that a good user story has three key elements. First, there's the title, which is a short reminder of what the story is about. Using a short title makes it easier for us to stack rank user stories because it's easier to scan short titles. Second, there's the description, written in the form "As a <type of user> I want <some goal> so that <some reason>," which provides the context for the story. That is, why does it add value, and for whom does it add value? This is the customer value! Third, there's acceptance criteria, the value of which we only learned later, when we switched over to the Microsoft Visual Studio Scrum 1.0 (Scrum 1.0 hereafter for brevity) process template. Acceptance criteria provides clarity to the team about what we planned to deliver.

As we matured in our adoption of Agile, we learned more about user stories and how to have key conversations with our stakeholders and customers. We couldn't just focus on writing strong user stories. We also had to discuss whether our team had correctly stack ranked the user stories. This led to further growth as a team as we started having good, productive conversations with our customers about the order of our work.

Software teams such as ours often decide themselves what's important. This is contrary to the Agile Manifesto, which emphasizes that everything is about the customer. To bridge that gap, we use the stack-rank field in the user story WIT to define the order in which we work on building value. This field drives the conversations with our customers and partners to ensure that our rankings are aligned with theirs. Using a simple TFS query, our stakeholders and customers can see an ordered list of work needing to be delivered to meet customer needs (this is also published on our dashboard, which I'll discuss later).

Planning Iterations in MSF Agile

As a team, we wanted to overcome our past history of having features grow in scope or size with little or no warning. And we wanted to focus on value instead of features. We needed to accurately plan the work, allocate the right amount of resources and effectively take interruptions into account.

Work on Agile teams is divided into iterations of a fixed length. But how long should an iteration be? After some discussion, we chose four-week iterations because we didn't think we could deliver "ship-ready" software in fewer than four weeks. A few iterations later, we discovered that it was hard to get all the work done in an iteration, so we tried switching to six weeks. However, this made the feedback loop with customers too long, so we switched back to four-week iterations. About 10 months ago, we switched to two-week sprints, which allow us to get feedback sooner.

After selecting the user stories that delivered the most value to our customers, we assigned them to iterations. Our team had to learn to focus on building small, incremental components rather than large, often monolithic, features. Smaller components gave our team the ability to estimate work at a more granular level, such as within five- to 10-hour increments. It also improved the effectiveness of our testers,

Figure 2 **Work Item Type Definitions in MSF Agile**

Work Item Type	Purpose
User Story	Tracks an activity the user will be able to perform with the product.
Task	Tracks work that needs to be done.
Bug	Describes a divergence between required and actual behavior, and tracks the work done to correct the defect and verify the correction.
Issue	Tracks an obstacle to progress.
Test Case	Describes a set of steps and expects outcomes to be tested.
Shared Steps	Defines a reusable set of test steps and outcomes.

Figure 3 **Managing Planning Estimates in MSF Agile**

MSF Agile Field	Purpose
Original Estimate	Initial value for remaining work—set once, when work begins.
Remaining Work	An estimate of the number of hours of work remaining to complete a task.
Completed Work	The number of hours of work that have been spent on a task.

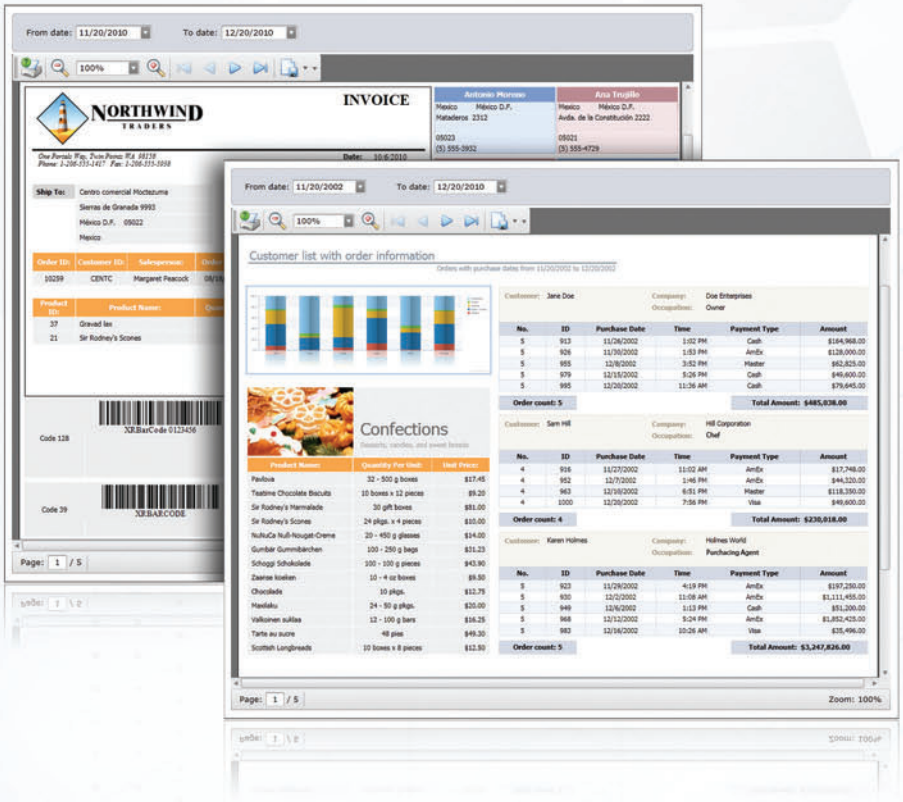
af·ford·able (rə'pôrting)

– adjective

1. within a developer's financial means; reasonably priced
2. all the benefits, without the high price

– synonyms

fair, reasonable



A feature-complete and multi-platform Reporting Library should not force you to spend thousands of dollars–it should be affordable and ship with the capabilities you need to get the job done… and offer the services you need to achieve mastery. **Prices start at $349.99**

Experience the power of the XtraReports Suite today.
Download your FREE trial – www.DevExpress.com/FreeEval.

Devexpress™
Download • Compare • Decide!

as they often didn't have long testing cycles because components were smaller.

For resourcing purposes, our team used to spend a significant amount of time focusing on the "design" of our features and the subsequent costs of building them. We replaced this in MSF Agile using story points for each story and then using the Original Estimate field to put a value on the work. We did this estimating using planning poker in our sprint planning (for more on planning poker, see planningpoker.com). Each day, we'd track our progress against our iteration goals by requiring team members to update their Completed Work and the Remaining Work fields (see **Figure 3**).

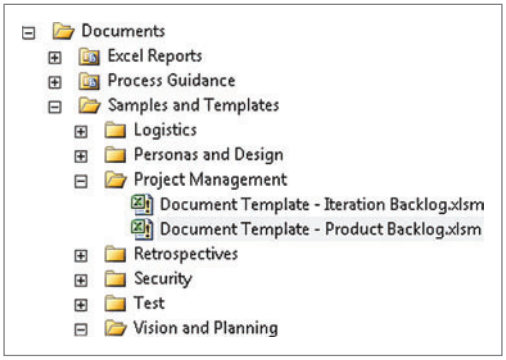


Figure 4 **Workbook Templates for Planning in MSF Agile**

This is the basis for accurately tracking day-to-day work, and we found that we became better and more efficient as we progressed as an Agile team.

As our team matured, updating our work estimates daily became habit and part of our normal process.

Product Planning and Iteration Backlog Tracking Tools in MSF Agile

Using the Product Planning and Iteration Backlog workbooks in MSF Agile (see **Figure 4**), we were able to improve our ability to estimate the work we could take on and successfully complete.

We weren't discouraged by our failures during the early iterations—we just focused on improving. During iteration planning as a team, we assigned user stories to iterations using the Product Planning workbooks. We tried to have iterations set up prior to starting sprint planning to save time. (For more information on creating iterations, see bit.ly/1akyZA.) As we learned more about our team's average velocity—how many story points we completed in previous iterations—we became better at balancing stories between iterations with some level of confidence we could finish by a specific date.

One of the most valuable parts of our transition to Agile is directly related to our use of the Iteration Backlog workbook. This workbook provides several Microsoft Excel tabs that help during iteration planning. I'll discuss the purpose of each tab.

The workbook starts on the Iteration Backlog tab that shows your user stories and any subsequent tasks related to those user stories. This tab is bound to a TFS tree query, which allows you to easily see the user story and all the tasks as children in a familiar tree view (for more information on tree query types, see bit.ly/l0tv1K). You'll need to update the query for each sprint so the first tab shows all items currently assigned to an iteration. You can manipulate data on this tab and republish it to the TFS server without leaving Excel, which is how we created the tasks under a user story during sprint planning (for more details, see bit.ly/1mPN4k).

The second tab, Area & Iteration, allows us to set Start Dates, End Dates and the Iteration path for our iteration, as shown in **Figure 5**.

For complex projects, you can utilize the Area path to scope the workbook. As a smaller team, we rarely scope to this level. In a larger team with multiple areas, you'll likely have different copies of the workbook for each Area path.

The third tab is Interruptions, which allows you to account for days off, such as vacations or holidays, when team members can't contribute to your project. This tab, though, will only allow you to set interruptions for team members who are currently assigned a work item in the iteration. Thus, before moving to this tab, ensure that you've accurately tasked out the work and assigned team members to it (on Tab 1, the Iteration Backlog), otherwise you'll have a blank list in the dropdown for team members.

Our team finds the fourth tab to be invaluable. Called Capacity, this tab provides information on over and under capacity for each team member based on the remaining work and days in the iteration. The Interruptions tab information, mentioned earlier, is taken into account on this tab.

The Capacity tab makes it much easier to load balance, by moving tasks between team members. This was what we lacked, as a team, prior to adopting Agile—the ability to do this reassignment early on instead of resorting to last-minute scrambling when it was often too late.

To calculate capacity, the workbook takes each team member's overall daily capacity (hours per day) and multiplies this by the number of days remaining in the iteration. The net result, as shown in **Figure 6**, allows us to start from day one of the iteration (for example, iteration planning) knowing where we stand.

It's a dynamic view that we examine in our daily stand-up. In the case of **Figure 6**, each team member is over capacity, leading the team to understand that it wouldn't be possible to complete all the work planned in that iteration.

I put together a walkthrough of how to use the workbook, and I've shared more details on my blog at bit.ly/fZpzWx.

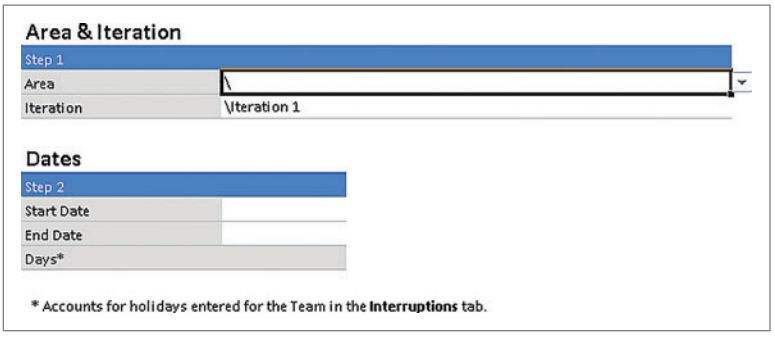


Figure 5 **Start and End Dates in the Iteration Backlog Workbook in MSF Agile**

Making Team-Specific Customizations

The building of software isn't always confined to a single team. In reality, multiple teams usually work together. The challenge of cross-team software development is the existence of different "team" processes. Some teams follow waterfall software development practices, while others use Scrum. The uniqueness of each team often

Team Foundation Server turns Agile.

Really.

Urban Turtle is a Microsoft Team Foundation Server add-on that extends Team Web Access with a planning board and a task board. Urban Turtle provides a compelling Scrum project management solution seamlessly integrated into Microsoft Team Foundation Server.

urbanturtle.com/msdn

Hey! We support these two templates! Isn't that cool?

TFS + urban TURTLE = efficient Scrum teams

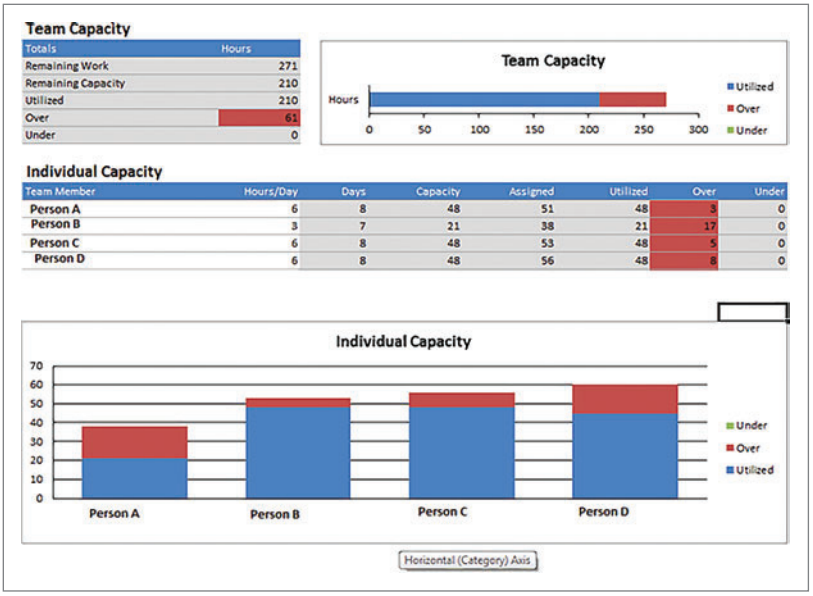


Figure 6 **MSF Agile Team Capacity Planning**

leads to challenges, and our team certainly wasn't isolated and able to avoid this. Instead, one of our first projects using MSF Agile required interactions between two teams using different iteration lengths and different styles, though both were Agile-based. This is where the flexibility of TFS 2010 aided our team, as it offers a rich and powerful ability to customize any WIT (or even make a brand-new one). We didn't need a new WIT, but instead just needed to tweak one.

We were developing a new feature called User-Driven Installation (bit.ly/kjySZk) in tandem with the Microsoft Deployment Toolkit (MDT) team. It empowered end users to change their Windows 7 desktops or laptops. The MDT team utilized Scrum—using proprietary tools only available within Microsoft—while our team utilized TFS 2010. Beyond this, a great deal of interaction between the teams focused on bugs as our team delivered new features and bug fixes while their team owned all testing.

The teams worked together over multiple iterations only to see the first few iterations struggle to meet the stated output (for example, shippable software). Our velocity was low. We, as a team, were using Agile several months prior to working with our partner and had a rhythm of finishing our iterations with all work completed. Why, all of a sudden, did we stop producing everything we planned for in the sprint?

The change, ironically, was our failure to include bug estimates during sprint planning. Thus, we often would take on new features along with all the bugs our partner team asked us to fix. Because the Bug WIT didn't have time-based fields such as Remaining Work and Completed Work, our burndown never alerted us that we had work that wouldn't get completed.

The customization features of TFS 2010 made tweaking this WIT to meet our needs rather simple. We updated our bug WIT to include the time fields, added them to the Bug form and were then able to track both bugs and tasks via the iteration burndown. For more information on how to add a custom field, such as planning fields to a WIT, see my blog at bit.ly/gb1BOb.

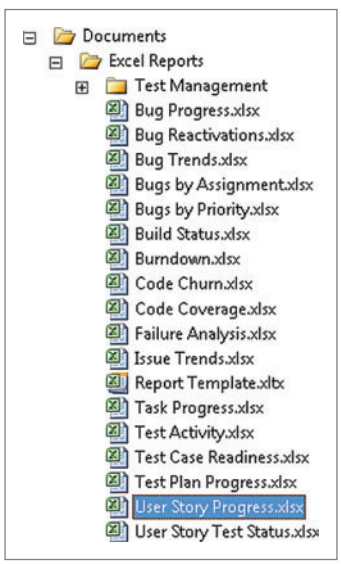


Figure 7 **Excel Reports in MSF Agile**

TFS Integration with Windows SharePoint Services

As you recall, we needed to improve our team's ability to track progress and report it accordingly to the team, our partners and our management.

TFS 2010 provides out-of-the-box integration to leverage a new Windows SharePoint Service (WSS) or an existing one. The creation of an MSF Agile project includes (if so desired) the creation of a SharePoint site that contains a nice, customizable dashboard. As part of our effort to improve our communication within the team and with customers, we leveraged the built-in dashboards that came with TFS. The compelling aspect of the dashboard and SharePoint integration was flexibility. The dashboard that came with the standard TFS project, for example, didn't provide everything desired within that dashboard view. We wanted more.

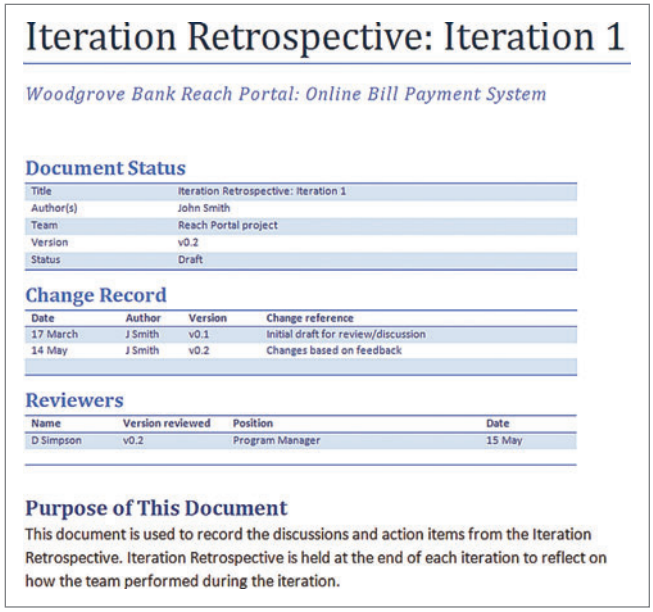
The dashboards allowed us to evolve away from daily e-mail sharing of progress reports such as Burndown, Velocity or Bug Counts, and instead provide our management and partners access to our SharePoint site where they could see our progress in near-real time.

As with any organization, we have different people with different desires when it comes to what they're interested in seeing. We customized our dashboard to include additional reports that, by default, weren't enabled on the dashboard. For example, our management team meets with our team monthly to review the work we've done and to provide feedback to us as we're moving along the lifecycle of our Agile projects. During one of these meetings, one of our key leaders asked if he could see the user stories we were working on, our progress and also user stories in our backlog that weren't currently within the sprint. To learn more about how we were able utilize the dashboard to meet this leader's need, see my blog post at bit.ly/jJ6XUp.

Unleash the Power of SharePoint and TFS 2010 Projects

If you or your company already have an enterprise version of Microsoft Office SharePoint Server (MOSS), you can unleash a new level of functionality by using MSF Agile projects linked with MOSS. As mentioned earlier, TFS 2010 ships with WSS, but WSS lacks some nice enterprise-ready features.

We wanted to utilize some of the great Excel reports that shipped with MSF Agile. The reports, as shown in **Figure 7**, offer a lot of capabilities, such as utilizing Excel Web Services (EWS) to host the reports and their data in Web Parts on our project dashboards.



Iteration Retrospective: Iteration 1

Woodgrove Bank Reach Portal: Online Bill Payment System

Document Status

Title	Iteration Retrospective: Iteration 1
Author(s)	John Smith
Team	Reach Portal project
Version	v0.2
Status	Draft

Change Record

Date	Author	Version	Change reference
17 March	J Smith	v0.1	Initial draft for review/discussion
14 May	J Smith	v0.2	Changes based on feedback

Reviewers

Name	Version reviewed	Position	Date
D Simpson	v0.2	Program Manager	15 May

Purpose of This Document

This document is used to record the discussions and action items from the Iteration Retrospective. Iteration Retrospective is held at the end of each iteration to reflect on how the team performed during the iteration.

Figure 8 **An MSF Agile Iteration Retrospective Template**

This functionality isn't available unless your team has a SharePoint 2007 or 2010 Enterprise server (or servers) available for the team project.

When TFS 2010 detects a SharePoint Enterprise server during creation of an MSF Agile project, it will create a dashboard that includes both EWS reports and SQL Server Reporting Services (SSRS) reports. This flexibility gave us a lot of capabilities because we gained the ability to use reports such as Code Churn and Bugs by Assignment, along with the reports provided in MSF Agile, such as Burndown and Burn Rate.

A key issue our team had while maintaining the SharePoint dashboard was management—specifically, who would own the task of keeping the reports up-to-date on the dashboard. For example, when our team was iterating at two-week intervals, the Burndown start date and end date needed updating every two weeks, otherwise the dashboard data wasn't up-to-date. We maintained the dashboard for several months, but over time we were looking for ways to either automate the creation of these reports or a simpler way to track our iterations. It wasn't long into our transition (nearly one year) that Scrum 1.0, downloadable from bit.ly/fdojaD, was released.

Figure 9 **Work Item Type Definitions in Scrum 1.0**

Work Item Type	Purpose
Product Backlog Item	Tracks an activity a user will be able to perform with the product.
Task	Tracks work that needs to be done.
Bug	Describes a divergence between required and actual behavior, and tracks the work done to correct the defect and verify the correction.
Impediment	Tracks an obstacle to progress.
Test Case	Server-side data for a set of steps to be tested.
Shared Steps	Server-side data for a reusable set of test steps.
Sprint	A sprint whereby work is performed taken from the product backlog with definitive start and end dates.

Agile: Retrospectives Are Invaluable

As we've evolved over time as an Agile team, we've spent countless hours together focused on improving our team, process and execution. This usually occurred when we had completed our iterations, and it ended with our sprint review (checked against our sprint goals) as well as the retrospectives. As an Agile team, it's often easy to overlook retrospectives, and we did at times, making us refocus energy on them.

In MSF Agile, TFS provides you the ability to track your retrospectives through an iteration backlog template provided on the SharePoint site of your project. This workbook allows you to track your velocity, what you did well and what you could improve upon going forward (see **Figure 8**).

We, as a team, have taken great pride in not only focusing on improving in retrospectives but also continually re-evaluating everything we do. Although much of our learning occurs in retrospective look-backs, we also tweak our Agile processes as technology evolves. In our case, TFS evolved when Microsoft released a new process template called Scrum 1.0 and we couldn't ignore it—in fact, we embraced it.

Our Transition to Scrum 1.0

Our team, although not looking to work in a strictly Scrum-based fashion, resembles a Scrum team in many ways. Prior to adopting MSF Agile, we utilized terms such as Product Backlog Items (PBIs) and then went through the transition to user stories.

As we did before, we decided to select one of our shorter projects to evaluate the new Scrum 1.0 process template. Much like we did when adopting MSF Agile, we spent time familiarizing ourselves with the WITs available in the Scrum 1.0 template. As shown in **Figure 9**, the terminology is slightly different, yet many of the concepts are similar.

A big change we quickly noticed in Scrum 1.0 was a WIT called Sprint. It allows Scrum 1.0 teams to define specific starting and ending dates for their sprints, track their goals for that sprint and store their notes from their retrospective within TFS 2010. As I mentioned earlier, MSF Agile offered similar functionality in the Iteration Backlog workbook and a Microsoft Word template for retrospectives. The ability to have sprints, their goals and the retrospectives as a work item that then has work assigned to it was a compelling change for us. This was the primary reason we moved from MSF Agile to Scrum 1.0.

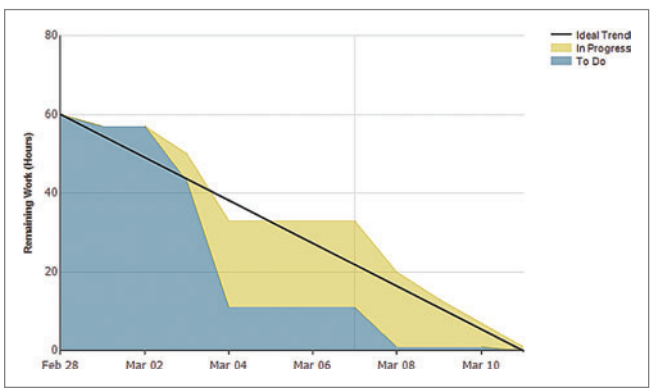


Figure 10 **Sample Sprint Burndown in Scrum 1.0**

THESE GUYS STAND ON THEIR OWN.

That's because we focus on the most important controls, not dozens of generic, bundled ones.

XCEED **DataGrid** for WPF

Mature, feature-packed, and lightning-fast. The most adopted and trusted WPF datagrid around!

XCEED **DataGrid** for Silverlight

The only Silverlight datagrid on the market with fast remote data retrieval and a completely fluid UI!

XCEED **Ultimate ListBox** for Silverlight

The same data virtualization and smooth-scrolling as our Silverlight datagrid, packed in the streamlined format of a listbox.

Try them live at **xceed.com**

XCEED
MULTI-TALENTED COMPONENTS

Figure 11 **Work Item Type Comparisons in MSF Agile vs. Scrum 1.0**

MSF Agile	Scrum 1.0
User Story	Product Backlog Item
Task	Task
Bug	Bug
Issue	Impediment
Test Case	Test Case
Shared Steps	Shared Steps
	Sprint

The PMs on our team who manage the features dashboards, reports and other iteration artifacts found themselves often spending significant time updating reports to reflect current point-in-time views for our team and partners. On the other hand, Scrum 1.0 has a Sprint WIT where we assign dates for our sprint. The Burndown report then uses the information in the Sprint work items to show the correct date range (see **Figure 10**).

It was simple, seamless and fit right in with work we could do during sprint planning. We no longer had to customize reports to set the start date or the end date—now it was done for us when we created the Sprint work item before sprint planning.

To recap, MSF Agile and Scrum 1.0 offer different options to Agile teams. The decision about which to use is up to you, of course, just like it was up to our team. **Figure 11** is a table that outlines WITs from each process template and how they align with each other. Again, the first thing I'd recommend to any new team adopting Agile software development using TFS 2010 is to learn in-depth how to use each WIT.

To learn more about the differences between the process templates, see my blog at bit.ly/i5tF2G.

Utilizing Iteration Workbook in Scrum 1.0 Projects

As our team evolved and more projects were created in TFS 2010, we found ourselves missing the iteration workbook. It helped us tremendously in understanding interruptions, and, more importantly, the team's capacity at a sprint level. Because of this, as a team we customized the workbook to be compatible with Scrum 1.0.

The key focal point of this change isn't to say that Scrum 1.0 didn't work for us. Rather, it's just to note that we missed a lot of the functionality provided within the workbook. My teammate John Socha-Leialoha (blogs.msdn.com/jsocha) outlined how to modify the Iteration Backlog workbook to work with the Scrum 1.0 template. The workbook doesn't work natively, partly because it uses data stored in fields not available in Scrum. In his post, he focused on helping others learn how our team got the workbook to work with our Scrum 1.0 projects. The net result was that our team was able to utilize the workbook during planning and stand-ups to track capacity. The one downside we found is that Scrum 1.0, by default, doesn't assign work to individuals during planning. Instead, the work is stack ranked and pulled directly off the backlog in an ordered approach. Thus, to be compatible with the workbook, we had to assign all work to individuals so that we could see capacity.

Discipline-Based Tracking vs. Assigned-To in Scrum 1.0

As I mentioned, assigning work to team members is problematic. Assigning work to individuals worked well for our projects that had one developer, one tester and one PM. Where we struggled, though, was in projects where we had capacity that varied based on our team dynamics. For example, one feature could have three developers, two testers and one PM. How do you "split up" work to individuals when it depends on who's available at that time?

Our team made a big change when we switched to the Scrum 1.0 template in that we moved from assigning work to individuals to a focus on discipline-based assignment. In the first projects we ran on Scrum 1.0, we found that we failed because we assigned work to individuals instead of to disciplines. We talked about this as a team and decided to use a discipline field, called Activity, but we renamed it to Discipline in the work item (see **Figure 12**).

This gave us the ability to see, at a glance, the capacity for developers, testers and PMs, so as we had resources added or taken from feature teams, we could adjust.

Putting It All Together

Our journey to Agile isn't complete; there are many additional opportunities for us to streamline our processes. The misnomer about Agile is that Agile teams lack discipline. We as a team feel this is completely inaccurate. Instead, we've learned that Agile teams have a high level of discipline. Prior to moving to Agile, our team lacked an effective process and, unfortunately, discipline. The move to TFS 2010 and the MSF Agile process template started our maturation and created a spirit that focuses on learning and adapting.

Our adaptation has helped us evolve from using the MSF Agile process template to where we are today: we now utilize the Scrum 1.0 process template and we feel, as a team, that we're almost "there." I hope that our journey has enlightened you, and motivated you to move your team to TFS 2010 and one of these process templates.

Without TFS 2010, though, we wouldn't have moved our team to where it is today. In the end, this is a story about one team's successful move to Agile using TFS 2010. ■

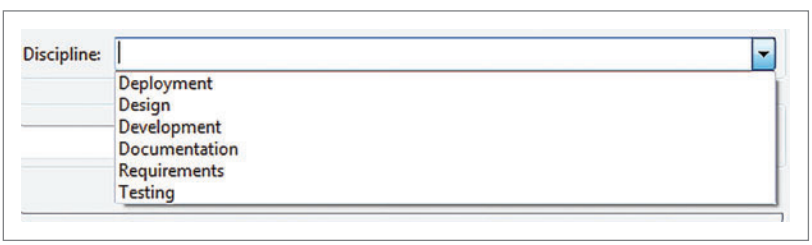


Figure 12 **Discipline-Based Assignments in Scrum 1.0**

CHRIS ADAMS *is a group program manager at Microsoft, leading a software development team that builds software based on Microsoft enterprise-management tools. Prior to his current role, he spent seven years focusing on IIS (blogs.iis.net/chrisad) as a program manager in capacities such as deployment, supportability, UI and core architecture. He's an avid blogger at blogs.technet.com/chrad and focuses on Windows, System Center and Visual Studio. You can reach him at chrad@microsoft.com.*

THANKS to the following technical experts for reviewing this article: *Aaron Bjork and John Socha-Leialoha*

we are Countersoft
you are Control

GEMINI
Project Platform

The most versatile project management platform for software development and testing teams around the world

The award-winning Gemini Project Platform is the .NET based project platform that uniquely allows teams to work the way they want to work with more functionality and easy customization for optimum team performance and predictable results.

BOOK A DEMO / FREE TRIAL / 3-for-FREE
Seeing is believing! We'll talk about YOU.
www.**geminiplatform**.com

ATLAS
Product Support Optimized

The first comprehensive solution to address ISVs' profitability AND customer experience by targeting product support

Patent pending Atlas addresses the area of the software business that matters most: the product support eco-system. Atlas is the logical alternative to revenue-sapping, customer-irritating forums. Atlas offers smarter community-based support and integrates FAQ/KB, documentation and how-to videos.

DOWNLOAD NOW
and lead from the front.
www.**atlasanswer**.com

COUNTERSOFT ENABLING COLLECTIVE CAPABILITY www.countersoft.com

Contact // Europe/Asia: +44 (0)1753 824000 // US/Canada: 800.927.5568 // E: sales@countersoft.com

Writing a Debugging Tools for Windows Extension, Part 3: Clients and Callbacks

Andrew Richards

In this third installment about the Debugger API, I'm going to delve deeper into the relationship a debugger extension can have with the debugger. I'll give you an overview of the architecture of debugger clients and debugger callbacks. In doing so, we'll get into the nitty-gritty of the DEBUG_OUTPUT_XXX and DEBUG_OUTCTL_XXX constants.

I'll use this foundation to implement an encapsulation of the Son of Strike (SOS) debugger extension. I'll enhance the SOS output with Debugger Markup Language (DML) and demonstrate how you can leverage the built-in debugger commands and other extensions to retrieve information required by your extensions.

This article discusses:

- Output control
- Debugger architecture
- Multiple debugger clients
- Output callbacks
- Execution
- What to execute

Technologies discussed:

Debugging Tools for Windows, Debugger Markup Language

Code download available at:

code.msdn.microsoft.com/mag201105DebugOutput

Before continuing, be sure you've read the previous two installments to understand what a debugger extension is (including how I'm building and testing the examples in this article) and how to correctly produce output. They can be found at msdn.microsoft.com/magazine/gg650659 and msdn.microsoft.com/magazine/hh148143.

Output Control

There are two ways to produce output in an extension: the IDebugControl::Output* functions and the IDebugControl::ControlledOutput* functions. The Output functions are just a simplification of the ControlledOuput functions. They retrieve the current output control setting and pass this to ControlledOutput as the first parameter:

```
HRESULT ControlledOutput(
  [in]  ULONG OutputControl,
  [in]  ULONG Mask,
  [in]  PCSTR Format,
        ...
);
```

I gave an overview of the DEBUG_OUTCTL_XXX constants used for the OutputControl parameter in the second article. In this article, I'm going to focus on the lower bits that control the scope of the output. One of the DEBUG_OUTCTL_XXX scope-based constants must be used for the lower bits. The value directs where the output is to go. This can be to all debugger clients (DEBUG_OUTCTL_ALL_CLIENTS), just the IDebugClient associated with the IDebugControl interface (DEBUG_OUTCTL_THIS_CLIENT), all other clients (DEBUG_OUTCTL_ALL_OTHER_CLIENTS),

nowhere at all (DEBUG_OUTCTL_IGNORE) or just the log file (DEBUG_OUTCTL_LOG_ONLY).

In Example06 in the accompanying code download (which is the same as the previous article), the !outctlpassed command (see **Figure 1**) is based on the passed IDebugClient interface.

The function does a QueryInterface for the IDebugControl interface (of the passed IDebugClient). Then ControlledOutput is called with each of the output control scopes. The Mask is set to normal (DEBUG_OUTPUT_NORMAL) so that it isn't excluded from output. By default, only a Mask value of verbose (DEBUG_OUTPUT_VERBOSE) is excluded from the WinDBG Output window. To test the command, I use the test_windbg.cmd script. In the launched WinDBG, I open the log file (.logopen), run the !outctlpassed command and close the log file (.logclose), as shown here:

```
0:000> .logopen outctlpassed.txt
Opened log file 'outctlpassed.txt'
0:000> !outctlpassed
DEBUG_OUTCTL_THIS_CLIENT
DEBUG_OUTCTL_ALL_CLIENTS
0:000> .logclose
Closing open log file outctlpassed.txt
```

> Debugging is based on four layers: the Debugger Helper, the Debugger Engine, the Debugger Client and Debugger Extensions.

The !outctlpassed command only includes DEBUG_OUTCTL_THIS_CLIENT and DEBUG_OUTCTL_ALL_CLIENTS output. The DEBUG_OUTCTL_ALL_OTHER_CLIENTS, DEBUG_OUTCTL_IGNORE and DEBUG_OUTCTL_LOG_ONLY output controls are omitted in the Output window. But if you look in the log file, there's a different result, as shown here:

```
Opened log file 'outctlpassed.txt'
0:000> !outctlpassed
DEBUG_OUTCTL_THIS_CLIENT
DEBUG_OUTCTL_ALL_CLIENTS
DEBUG_OUTCTL_ALL_OTHER_CLIENTS
DEBUG_OUTCTL_LOG_ONLY
0:000> .logclose
Closing open log file outctlpassed.txt
```

The only output control missing from the log file is DEBUG_OUTCTL_IGNORE.

Debugger Architecture

To understand what happened to the missing output in the WinDBG Output window and the log file, we need to delve in to the debugger architecture. Debugging is based on four layers: the Debugger Helper, the Debugger Engine, the Debugger Client and Debugger Extensions.

At the bottom is the Debugger Helper (dbghelp.dll). This library contains all of the functionality to resolve symbols.

The next layer is the Debugger Engine (dbgeng.dll). This library handles the debugging session and, in particular, provides the support to debug a remote target. Included in this library is the handling of the debugger interfaces (for example, IDebugClient, IDebugControl and so on).

The next layer is the Debugger Client (client) layer. At this layer, input and output are processed as the client sees fit. The WinDBG and NTSD debuggers are in this layer. A client uses the Debugger Engine DebugCreate and DebugConnect functions to create an IDebugClient object that's attached to the desired target. The target can be local or remote (through the Debugger Engine proxy support).

The last layer is Debugger Extensions (extension). A client invokes commands exported by the Debugger Extension DLL. The IDebugClient interface that the Debugger Client receives from the Debugger Engine is the same interface that the client passes to extensions. Although invoked by a client, an extension only interacts with the Debugger Engine. Extensions can share the

Figure 1 **!outctlpassed**

```
HRESULT CALLBACK
outctlpassed(PDEBUG_CLIENT pDebugClient, PCSTR args)
{
  UNREFERENCED_PARAMETER(args);

  IDebugControl* pDebugControl;
  if (SUCCEEDED(pDebugClient->
    QueryInterface(__uuidof(IDebugControl), (void **)&pDebugControl)))
  {
    pDebugControl->ControlledOutput(DEBUG_OUTCTL_THIS_CLIENT,
      DEBUG_OUTPUT_NORMAL, "DEBUG_OUTCTL_THIS_CLIENT\n");
    pDebugControl->ControlledOutput(DEBUG_OUTCTL_ALL_CLIENTS,
      DEBUG_OUTPUT_NORMAL, "DEBUG_OUTCTL_ALL_CLIENTS\n");
    pDebugControl->ControlledOutput(DEBUG_OUTCTL_ALL_OTHER_CLIENTS,
      DEBUG_OUTPUT_NORMAL, "DEBUG_OUTCTL_ALL_OTHER_CLIENTS\n");
    pDebugControl->ControlledOutput(DEBUG_OUTCTL_IGNORE,
      DEBUG_OUTPUT_NORMAL, "DEBUG_OUTCTL_IGNORE\n");
    pDebugControl->ControlledOutput(DEBUG_OUTCTL_LOG_ONLY,
      DEBUG_OUTPUT_NORMAL, "DEBUG_OUTCTL_LOG_ONLY\n");

    pDebugControl->Release();
  }
  return S_OK;
}
```

Figure 2 **!outctlcreate**

```
HRESULT CALLBACK
outctlcreate(PDEBUG_CLIENT pDebugClient, PCSTR args)
{
  UNREFERENCED_PARAMETER(args);

  IDebugClient* pDebugCreated;
  if (SUCCEEDED(pDebugClient->CreateClient(&pDebugCreated)))
  {
    IDebugControl* pDebugControl;
    if (SUCCEEDED(pDebugCreated->QueryInterface(__uuidof(IDebugControl),
      (void **)&pDebugControl)))
    {
      pDebugControl->ControlledOutput(DEBUG_OUTCTL_THIS_CLIENT,
        DEBUG_OUTPUT_NORMAL, "DEBUG_OUTCTL_THIS_CLIENT\n");
      pDebugControl->ControlledOutput(DEBUG_OUTCTL_ALL_CLIENTS,
        DEBUG_OUTPUT_NORMAL, "DEBUG_OUTCTL_ALL_CLIENTS\n");
      pDebugControl->ControlledOutput(DEBUG_OUTCTL_ALL_OTHER_CLIENTS,
        DEBUG_OUTPUT_NORMAL, "DEBUG_OUTCTL_ALL_OTHER_CLIENTS\n");
      pDebugControl->ControlledOutput(DEBUG_OUTCTL_IGNORE,
        DEBUG_OUTPUT_NORMAL, "DEBUG_OUTCTL_IGNORE\n");
      pDebugControl->ControlledOutput(DEBUG_OUTCTL_LOG_ONLY,
        DEBUG_OUTPUT_NORMAL, "DEBUG_OUTCTL_LOG_ONLY\n");
      pDebugControl->Release();
    }
    pDebugCreated->Release();
  }
  return S_OK;
}
```

Figure 3 **!outctlthis**

```
HRESULT CALLBACK
outctlthis(PDEBUG_CLIENT pDebugClient, PCSTR args)
{
  UNREFERENCED_PARAMETER(args);

  IDebugClient* pDebugCreated;
  if (SUCCEEDED(pDebugClient->CreateClient(&pDebugCreated)))
  {
    IDebugControl* pDebugControl;
    if (SUCCEEDED(pDebugCreated->QueryInterface(__uuidof(IDebugControl),
      (void **)&pDebugControl)))
    {
      pDebugControl->ControlledOutput(DEBUG_OUTCTL_THIS_CLIENT |
        DEBUG_OUTCTL_NOT_LOGGED, DEBUG_OUTPUT_NORMAL,
        "DEBUG_OUTCTL_THIS_CLIENT | DEBUG_OUTCTL_NOT_LOGGED\n");
      pDebugControl->Release();
    }
    pDebugCreated->Release();
  }
  return S_OK;
}
```

IDebugClient interface with the client as long as they don't damage its configuration. Alternatively, they can make their own client (via IDebugClient::CreateClient) and configure it as needed.

Multiple Debugger Clients

Multiple clients (IDebugClient objects) can be attached to the same debugging session; it's the session that's singular, not the clients. Each client has its own private interface to the Debugger Engine, and it's this ability that we're going to leverage.

Initially, when you attach WinDBG or NTSD to a process, there's just one client—the IDebugClient interface returned by DebugCreate or DebugConnect. When an extension calls IDebugClient::CreateClient against this returned interface, the Debugger Engine makes another client that's associated with the session. The new client is attached to the same debugging session, but it's in a default state. Callbacks aren't set on it and the output mask is the default value (everything except Verbose).

In **Figure 2**, the !outctlcreate does the same output as !outctlpassed, but is based on a newly created IDebugClient object.

As before, to test the command, I use the test_windbg.cmd script. In the launched WinDBG, I open the log file (.logopen), run the !outctlcreate command and close the log file (.logclose), as shown here:

```
0:000> .logopen outctlcreate.txt
Opened log file 'outctlcreate.txt'
0:000> !outctlcreate
DEBUG_OUTCTL_ALL_CLIENTS
DEBUG_OUTCTL_ALL_OTHER_CLIENTS
0:000> .logclose
Closing open log file outctlcreate.txt
```

The !outctlcreate command only includes the DEBUG_OUTCTL_ALL_CLIENTS and DEBUG_OUTCTL_ALL_OTHER_CLIENTS output. The "all others" output replaces the "this" output because the WinDBG client is now the "other" client. The log file results are still the same, as shown here:

```
Opened log file 'outctlcreate.txt'
0:000> !outctlcreate
DEBUG_OUTCTL_THIS_CLIENT
DEBUG_OUTCTL_ALL_CLIENTS
DEBUG_OUTCTL_ALL_OTHER_CLIENTS
DEBUG_OUTCTL_LOG_ONLY
0:000> .logclose
Closing open log file outctlcreate.txt
```

Inside the Debugging Engine, the ControlledOutput call is being conditionally acted upon. The engine invokes the output callback on all clients that match the output control criteria. So, if you make your own client and use the DEBUG_OUTCTL_THIS_CLIENT output control, the output will stay local (that is, not displayed in WinDBG). Note that the output still goes to the log file. If you add the "not logged" high bit (DEBUG_OUTCTL_NOT_LOGGED), you can stop this, too.

Once again, the !outctlthis command does just "this" and "not logged" output (DEBUG_OUTCTL_THIS_CLIENT | DEBUG_OUTCTL_NOT_LOGGED) on a created client, as shown in **Figure 3**.

As before, to test the command, I use the test_windbg.cmd script. In the launched WinDBG, I open the log file (.logopen), run the !outctlthis command and close the log file (.logclose), as shown here:

```
0:000> .logopen outctlthis.txt
Opened log file 'outctlthis.txt'
0:000> !outctlthis
0:000> .logclose
Closing open log file outctlthis.txt
```

The log file doesn't record the command's output this time. Note that it does record the execution because prompt output is being created by the WinDBG client prior to the invoking of the extension, as shown here:

```
Opened log file 'outctlthis.txt'
0:000> !outctlthis
0:000> .logclose
Closing open log file outctlthis.txt
```

In effect, I've redirected output to NULL because the client created doesn't have any associated output callback—not exactly a useful thing. But if I change the code to use the IDebugControl::Execute function, I can now execute any command without it being displayed in WinDBG or being logged, as shown here:

```
pDebugControl->Execute(DEBUG_OUTCTL_THIS_CLIENT | DEBUG_OUTCTL_NOT_LOGGED,
  args, DEBUG_EXECUTE_NOT_LOGGED | DEBUG_EXECUTE_NO_REPEAT);
```

When using Execute in this way, you need to set the DEBUG_EXECUTE_NOT_LOGGED and DEBUG_EXECUTE_NO_REPEAT flags so that the prompt output isn't generated in the log file and the command isn't saved for repetition (for example, for when the user executes a blank command by pressing Enter in the command prompt).

As shown in **Figure 4**, at the end the !outctlexecute command silently executes the supplied argument.

Figure 4 **!outctlexecute**

```
HRESULT CALLBACK
outctlexecute(PDEBUG_CLIENT pDebugClient, PCSTR args)
{
  IDebugClient* pDebugCreated;
  if (SUCCEEDED(pDebugClient->CreateClient(&pDebugCreated)))
  {
    IDebugControl* pDebugControl;
    if (SUCCEEDED(pDebugCreated->QueryInterface(__uuidof(IDebugControl),
      (void **)&pDebugControl)))
  {
    pDebugControl->Execute(DEBUG_OUTCTL_THIS_CLIENT | DEBUG_OUTCTL_NOT_LOGGED,
      args, DEBUG_EXECUTE_NOT_LOGGED | DEBUG_EXECUTE_NO_REPEAT);
    pDebugControl->Release();
  }
  pDebugCreated->Release();
  }
  return S_OK;
}
```

I'm executing a forced reload of symbols (.reload /f). This would usually produce lots of symbol output, but !outctlexecute has hidden the output, as shown here:

```
0:000> !outctlexecute .reload /f
```

Output Callbacks

When the Debugger Engine routes output requests to the clients (indicated by the output control), there are two additional constraints applied:

- Is the output of a type (mask) that's wanted?
- Is there an associated output callback?

> Multiple clients can be attached to the same debugging session.

When IDebugControl::Output or IDebugControl::ControlledOutput is invoked, the Debugging Engine compares the supplied mask with the client's output mask (IDebugClient::GetOutputMask). The two masks need to match for the output callback to be invoked.

Next, the Debugging Engine calls IDebugClient::GetOutputCallbacks to get an output callback interface (IDebugOutputCallbacks, IDebugOutputCallbacks2 or IDebugOutputCallbacksWide). If there's an output callback set on the client (previously done via IDebugClient::SetOutputCallbacks), the Debugging Engine invokes the callback—for example, IDebugOutputCallbacks::Output.

In Example07 in the accompanying code download, I've coded an IDebugOutputCallbacks implementation that supports 1MB of normal output and 1MB of error output. (The header is in **Figure 5** and the code is in **Figure 6**.)

The class appends the string to the normal or error buffer upon each Output call. Apart from the functions required for the IUnknown and IDebugOutputCallbacks interfaces, there are some additional functions to provide access to the buffers (BufferNormal and BufferError), to clear (Clear) them, and a function to get the supported output types (SupportedMask).

Each time Output or ControlledOutput is invoked (and the IDebugClient criteria are met), the callback's Output function is called. The mask passed is the same mask that's set on the original output call. My Output function does bitmask comparison on the passed mask, so that the text is associated with the correct buffer. It's up to the implementation of the callback to decide whether the output is stored separately, combined or ignored. A single IDebugClient::Execute call might result in hundreds of Output calls of varying types. Because I want to interpret the normal and error output of the entire Execute operation, I concatenate these strings into two associated buffers.

The IDebugOutputCallbacks interface is passed the string in ANSI and TEXT format. If the original text is in Unicode or in DML, the engine will do a conversion prior to the callback. To capture Unicode (wide) content, use the IDebugOutCallbacksWide interface instead. The only difference between these two interfaces is the Text parameter being passed as PCWSTR instead of PCSTR. Both of these interfaces only support the notification of TEXT-based output. They don't receive DML-formatted text.

To get DML content, you need to use the IDebugOutputCallbacks2 interface. This interface gives you control of how the text received by the callback is formatted: TEXT, DML or either one. The GetInterestMask function is used to define this. It's called when the interface is set on a client via IDebugClient::SetOutputCallbacks. You return whether you want to receive TEXT (DEBUG_OUTCBI_TEXT), DML (DEBUG_OUTCBI_DML) or either type of output (DEBUG_OUTCBI_ANY_FORMAT). Note that this isn't the same as the Output Mask.

The Output function has no purpose on this interface; it's an artifact of the (fake) interface inheritance of IDebugCallbacks. All output callback notification is via the Output2 function:

```
STDMETHOD(Output2)(ULONG Which, ULONG Flags, ULONG64 Arg, PCWSTR Text)
```

The Which parameter specifies whether TEXT (DEBUG_OUTCB_TEXT) or DML (DEBUG_OUTCB_DML) is being passed, or that a flush (DEBUG_OUTCB_EXPLICIT_FLUSH) is desired (IDebugClient::FlushCallbacks). Note that these are the CB* constants; not the CBI* constants used for the interest mask.

Figure 5 **OutputCallbacks.h**

```
#ifndef __OUTPUTCALLBACKS_H__
#define __OUTPUTCALLBACKS_H__

#include "dbgexts.h"

class COutputCallbacks : public IDebugOutputCallbacks
{
  private:
    long m_ref;
    PCHAR m_pBufferNormal;
    size_t m_nBufferNormal;
    PCHAR m_pBufferError;
    size_t m_nBufferError;

  public:
    COutputCallbacks()
          {
      m_ref = 1;
      m_pBufferNormal = NULL;
      m_nBufferNormal = 0;
      m_pBufferError = NULL;
      m_nBufferError = 0;
    }

    ~COutputCallbacks()
    {
      Clear();
    }

    // IUnknown
    STDMETHOD(QueryInterface)(__in REFIID InterfaceId, __out PVOID* Interface);
    STDMETHOD_(ULONG, AddRef)();
    STDMETHOD_(ULONG, Release)();

    // IDebugOutputCallbacks
    STDMETHOD(Output)(__in ULONG Mask, __in PCSTR Text);

    // Helpers
    ULONG SupportedMask() { return DEBUG_OUTPUT_NORMAL | DEBUG_OUTPUT_ERROR; }
    PCHAR BufferNormal() { return m_pBufferNormal; }
    PCHAR BufferError() { return m_pBufferError; }
    void Clear();
};

#endif // #ifndef __OUTPUTCALLBACKS_H__
```

Figure 6 **OutputCallbacks.cpp**

```
#include "dbgexts.h"
#include "outputcallbacks.h"

#define MAX_OUTPUTCALLBACKS_BUFFER 0x1000000  // 1Mb
#define MAX_OUTPUTCALLBACKS_LENGTH 0x0FFFFFF  // 1Mb - 1

STDMETHODIMP COutputCallbacks::QueryInterface(__in REFIID InterfaceId,
  __out PVOID* Interface)
{
  *Interface = NULL;
  if (IsEqualIID(InterfaceId, __uuidof(IUnknown)) || IsEqualIID(InterfaceId,
    __uuidof(IDebugOutputCallbacks)))
  {
    *Interface = (IDebugOutputCallbacks *)this;
    InterlockedIncrement(&m_ref);
    return S_OK;
  }
  else
  {
    return E_NOINTERFACE;
  }
}

STDMETHODIMP_(ULONG) COutputCallbacks::AddRef()
{
  return InterlockedIncrement(&m_ref);
}

STDMETHODIMP_(ULONG) COutputCallbacks::Release()
{
  if (InterlockedDecrement(&m_ref) == 0)
  {
    delete this;
    return 0;
  }
  return m_ref;
}

STDMETHODIMP COutputCallbacks::Output(__in ULONG Mask, __in PCSTR Text)
{
  if ((Mask & DEBUG_OUTPUT_NORMAL) == DEBUG_OUTPUT_NORMAL)
  {
    if (m_pBufferNormal == NULL)
    {
      m_nBufferNormal = 0;
      m_pBufferNormal = (PCHAR)malloc(sizeof(CHAR)*(MAX_OUTPUTCALLBACKS_BUFFER));
      if (m_pBufferNormal == NULL) return E_OUTOFMEMORY;
      m_pBufferNormal[0] = '\0';
      m_pBufferNormal[MAX_OUTPUTCALLBACKS_LENGTH] = '\0';
    }
    size_t len = strlen(Text);
    if (len > (MAX_OUTPUTCALLBACKS_LENGTH-m_nBufferNormal))
    {
      len = MAX_OUTPUTCALLBACKS_LENGTH-m_nBufferNormal;
    }
    if (len > 0)
    {
      memcpy(&m_pBufferNormal[m_nBufferNormal], Text, len);
      m_nBufferNormal += len;
      m_pBufferNormal[m_nBufferNormal] = '\0';
    }
  }
  if ((Mask & DEBUG_OUTPUT_ERROR) == DEBUG_OUTPUT_ERROR)
  {
    if (m_pBufferError == NULL)
    {
      m_nBufferError = 0;
      m_pBufferError = (PCHAR)malloc(sizeof(CHAR)*(MAX_OUTPUTCALLBACKS_BUFFER));
      if (m_pBufferError == NULL) return E_OUTOFMEMORY;
      m_pBufferError[0] = '\0';
      m_pBufferError[MAX_OUTPUTCALLBACKS_LENGTH] = '\0';
    }
    size_t len = strlen(Text);
    if (len >= (MAX_OUTPUTCALLBACKS_LENGTH-m_nBufferError))
    {
      len = MAX_OUTPUTCALLBACKS_LENGTH-m_nBufferError;
    }
    if (len > 0)
    {
      memcpy(&m_pBufferError[m_nBufferError], Text, len);
      m_nBufferError += len;
      m_pBufferError[m_nBufferError] = '\0';
    }
  }
  return S_OK;
}

void COutputCallbacks::Clear()
{
  if (m_pBufferNormal)
  {
    free(m_pBufferNormal);
    m_pBufferNormal = NULL;
    m_nBufferNormal = 0;
  }
  if (m_pBufferError)
  {
    free(m_pBufferError);
    m_pBufferError = NULL;
    m_nBufferError = 0;
  }
}
```

The Flags parameter specifies additional information in a bitmask. In particular, for DML content it specifies whether the content contains hyperlinks (DEBUG_OUTCBF_DML_HAS_TAGS) or has special characters (", &, < and >) that are encoded (DEBUG_OUTCBF_DML_HAS_SPECIAL_CHARACTERS). The parameter can also indicate that a flush (DEBUG_OUTCBF_COMBINED_EXPLICIT_FLUSH) is desired after the output has been processed. This occurs when Output and FlushCallbacks calls are combined by the engine.

> I implement my callbacks as referenced counted heap objects.

The Arg parameter contains the output mask for DML, TEXT and FLUSH callbacks.

Finally, the Text parameter is the text being outputted. This will be NULL when the callback is being invoked for a flush (DEBUG_OUTCB_EXPLICIT_FLUSH). The text is always passed as Unicode.

Execution

It doesn't take much code to add the callback to the !outctlexecute command (see **Figure 4**). The additional steps are to allocate the callback and to associate it with the created IDebugClient interface before the call to Execute.

I implement my callbacks as referenced counted heap objects. If you're using the Windows Driver Kit build environment, you'll need to disable "PREfast for Drivers" warning 28197 so that you don't get an OACR (Microsoft Auto Code Review) warning about the "new" allocation each time you compile the project. (PREfast doesn't detect that the class is reference counted.)

There are two parts to the association of the callback to the client: the output mask (SetOutputMask) and the output callback (SetOutputCallbacks). I have a helper function on my callback class

Our Name Says It All

activePDF®

Come and see why thousands of customers have trusted us over the last decade for all their server based PDF development needs.

. Convert over 400 files types to PDF
. High fidelity translation from PDF to Office
. ISO 32000 Support including PDF/X & PDF/A
. HTML5 to PDF
. True PDF Print Server
. Form Fill PDF
. Append, Stamp, Secure and Digitally Sign

activePDF®
Leading the iPaper Revolution

Compatible with Windows 7 | Windows Vista® | Microsoft Partner Silver Midmarket Solution Provider

Download our FREE evaluation from www.activepdf.com/MSDN

Call 1-866-GoTo-PDF | 949-582-9002 | Sales@activepdf.com

to return the supported output mask (COutputCallbacks::SupportedMask) so that I don't have to hardcode it. If I didn't set the mask, I'd run the risk of not receiving the expected output.

The !callbackecho command (see **Figure 7**) is implemented in the Example07 project in the accompanying download.

> The enumeration of loaded debugger extensions and their associated abilities isn't part of the debugging API.

It executes two commands ("vertarget" and the passed argument) and displays the normal and error output of each. When using the ControlledOutput function, you'll note that I use the DEBUG_OUTCTL_ALL_OTHER_CLIENTS output control. This directs the output to all other clients and avoids the output being captured by my associated output callback. If you're looping though captured output and are producing output of the same mask, be sure that you use the DEBUG_OUTCTL_ALL_OTHER_CLIENTS output control, otherwise you could easily get yourself into an infinite loop.

As before, I use the test_windbg.cmd script to test the command. In the launched WinDBG, I run the !callbackecho command and see the normal output of "vertarget" and the error output of "r @rip." (The register command fails because this is an x86 target and "rip" is a x64 register.) Note that in between the Execute calls, I call my Clear helper function to reset the buffers. This is what causes the "vertarget" output to be missing from the "r @rip" command output, as shown here:

```
0:000> !callbackecho r @rip
[Normal]
Windows 7 Version 7600 MP (2 procs) Free x86 compatible
Product: WinNt, suite: SingleUserTS
kernel32.dll version: 6.1.7600.16385 (win7_rtm.090713-1255)
Machine Name:
Debug session time: Sat Jan 29 22:01:59.080 2011 (UTC - 8:00)
System Uptime: 0 days 6:22:48.483
Process Uptime: 0 days 0:01:05.792
  Kernel time: 0 days 0:00:00.015
  User time: 0 days 0:00:00.000

[Error]

[Normal]

[Error]
           ^ Bad register error in 'r @rip'
```

You can actually implement the same functionality without making a client of your own. If you carefully swap in the output mask and the output callback of the passed client with your own values, you can achieve the same result. However, you need to be quite accurate when doing this. The passed IDebugClient is the client that the (WinDBG) debugger uses for output to the Output window. If you don't return it to the state that it started at, the debugger will no longer function correctly.

Figure 8 (from Example07 in the code download) shows the toggling approach using the !commandtoggle command.

It implements exactly the same functionality as !commandecho. To be able to call Execute and ControlledOutput, I frequently needed to toggle the output mask and output callback values between the passed values and my values to get the desired outcome. Note that the ControlledOutput is passed the DEBUG_OUTCTL_ALL_CLIENTS output control in this scenario, as the associated client is now a desired output target. Additionally, you'll note that I only implemented (for brevity) error handling on the initial calls.

In most cases, the reuse of a client can become quite cumbersome; this approach should be used with caution.

What to Execute?

The major reason I use this private output technique is to implement DML versions of TEXT-based commands, where I have no knowledge

Figure 7 **!callbackecho**

```
HRESULT CALLBACK
callbackecho(PDEBUG_CLIENT pDebugClient, PCSTR args)
{
  IDebugClient* pDebugCreated;
  if (SUCCEEDED(pDebugClient->CreateClient(&pDebugCreated)))
  {
    IDebugControl* pDebugControl;
    if (SUCCEEDED(pDebugCreated->QueryInterface(__uuidof(IDebugControl),
      (void **)&pDebugControl)))
    {
      // Create our IDebugOutputCallbacks-based object
      #pragma warning( disable : 28197 ) // PreFAST sees the 'new' as a leak due to
                                         // the use of AddRef/Release
      COutputCallbacks* pOutputCallbacks = new COutputCallbacks;
      if (pOutputCallbacks != NULL)
      {
        // Apply IDebugOutputCallbacks to the IDebugClient
        if (SUCCEEDED(pDebugCreated->
          SetOutputMask(pOutputCallbacks->SupportedMask())) &&
          SUCCEEDED(pDebugCreated->SetOutputCallbacks(pOutputCallbacks)))
      {
        // Execute and display 'vertarget'
        pDebugControl->Execute(DEBUG_OUTCTL_THIS_CLIENT | DEBUG_OUTCTL_NOT_LOGGED,
          "vertarget", DEBUG_EXECUTE_NOT_LOGGED | DEBUG_EXECUTE_NO_REPEAT);
        pDebugControl->ControlledOutput(DEBUG_OUTCTL_ALL_OTHER_CLIENTS,
          DEBUG_OUTPUT_NORMAL, "[Normal]\n%s\n", pOutputCallbacks->BufferNormal() ?
            pOutputCallbacks->BufferNormal() : "");
        pDebugControl->ControlledOutput(DEBUG_OUTCTL_ALL_OTHER_CLIENTS,
          DEBUG_OUTPUT_ERROR , "[Error] \n%s\n", pOutputCallbacks->BufferError() ?
          pOutputCallbacks->BufferError() : "");

        // Clear the callback buffers
        pOutputCallbacks->Clear();

        // Execute and display the passed command
        pDebugControl->Execute(DEBUG_OUTCTL_THIS_CLIENT | DEBUG_OUTCTL_NOT_LOGGED,
          args, DEBUG_EXECUTE_NOT_LOGGED | DEBUG_EXECUTE_NO_REPEAT);
        pDebugControl->ControlledOutput(DEBUG_OUTCTL_ALL_OTHER_CLIENTS,
          DEBUG_OUTPUT_NORMAL, "[Normal]\n%s\n", pOutputCallbacks->BufferNormal() ?
          pOutputCallbacks->BufferNormal() : "");
        pDebugControl->ControlledOutput(DEBUG_OUTCTL_ALL_OTHER_CLIENTS,
          DEBUG_OUTPUT_ERROR , "[Error] \n%s\n", pOutputCallbacks->BufferError() ?
          pOutputCallbacks->BufferError() : "");

        pDebugCreated->SetOutputCallbacks(NULL); // It's a best practice to
                                                 // release callbacks
                                                 // before the client release
        }
        pOutputCallbacks->Release();
      }
      pDebugControl->Release();
    }
    pDebugCreated->Release();
  }
  return S_OK;
}
```

DevSuite

Requirements Driven Quality Management

DevSpec

Use DevSpec to define your requirements

- Import and sync from Word, PDF, and Windows Explorer
- Collaborate with an integrated Wiki, transparent linking, and offline support
- Control requirement changes and view their potential impact
- Provide complete visibility into the entire software development lifecycle

DevTest

Use DevTest to manage your testing

- Create a central repository for your test cases, Knowledge items and automation scripts
- Schedule releases and test cycles using a wizard-driven interface
- Execute test assignments and submit defects from the same interface
- Track results with real-time dashboards and reports

DevTrack

Use DevTrack to track defects/issues

- Track each issue through a definable workflow
- SCM integration-track fixes against their source code deliverables
- Deploy a resolution across multiple releases, versions and products
- Reporting and metrics to illustrate the entire defect lifecycle

TechExcel

www.techexcel.com | 1-800-439-7782

Figure 8 **!callbacktoggle**

```
HRESULT CALLBACK
callbacktoggle(PDEBUG_CLIENT pDebugClient, PCSTR args)
{
  IDebugControl* pDebugControl;
  if (SUCCEEDED(pDebugClient->QueryInterface(__uuidof(IDebugControl),
    (void **)&pDebugControl)))
  {
    // Remember the original mask and callback
    ULONG ulOriginalMask;
    IDebugOutputCallbacks* pOriginalCallback;
    if (SUCCEEDED(pDebugClient->GetOutputMask(&ulOriginalMask)) &&
      SUCCEEDED(pDebugClient->GetOutputCallbacks(&pOriginalCallback)))
    {
      // Create our IDebugOutputCallbacks based object
      #pragma warning( disable : 28197 ) // PreFAST sees the 'new' as a leak due to
                                         // the use of AddRef/Release
      COutputCallbacks* pOutputCallbacks = new COutputCallbacks;
      if (pOutputCallbacks != NULL)
      {
        // Apply IDebugOutputCallbacks to the IDebugClient
        if (SUCCEEDED(pDebugClient->SetOutputMask(pOutputCallbacks->
          SupportedMask())) && (pDebugClient->
          SetOutputCallbacks(pOutputCallbacks)))
        {
          // Execute 'vertarget'
          pDebugControl->Execute(DEBUG_OUTCTL_THIS_CLIENT |
            DEBUG_OUTCTL_NOT_LOGGED, "vertarget",
            DEBUG_EXECUTE_NOT_LOGGED | DEBUG_EXECUTE_NO_REPEAT);

          // Revert the mask and callback so we can do some output
          pDebugClient->SetOutputMask(ulOriginalMask);
          pDebugClient->SetOutputCallbacks(pOriginalCallback);

          // Display 'vertarget'
          pDebugControl->ControlledOutput(DEBUG_OUTCTL_ALL_CLIENTS,
            DEBUG_OUTPUT_NORMAL, "[Normal]\n%s\n", pOutputCallbacks->
            BufferNormal() ? pOutputCallbacks->BufferNormal() : "");
          pDebugControl->ControlledOutput(DEBUG_OUTCTL_ALL_CLIENTS,
            DEBUG_OUTPUT_ERROR , "[Error] \n%s\n", pOutputCallbacks->
            BufferError() ? pOutputCallbacks->BufferError() : "");

          // Go back to our mask and callback so we can do the private callback
          pDebugClient->SetOutputMask(pOutputCallbacks->SupportedMask());
          pDebugClient->SetOutputCallbacks(pOutputCallbacks);

          // Clear the callback buffers
          pOutputCallbacks->Clear();

          // Execute the passed command
          pDebugControl->Execute(DEBUG_OUTCTL_THIS_CLIENT |
            DEBUG_OUTCTL_NOT_LOGGED, args, DEBUG_EXECUTE_NOT_LOGGED |
            DEBUG_EXECUTE_NO_REPEAT);

          // Revert the mask and callback so we can do some output
          pDebugClient->SetOutputMask(ulOriginalMask);
          pDebugClient->SetOutputCallbacks(pOriginalCallback);

          // Display the passed command
          pDebugControl->ControlledOutput(DEBUG_OUTCTL_ALL_CLIENTS,
            DEBUG_OUTPUT_NORMAL, "[Normal]\n%s\n", pOutputCallbacks->
            BufferNormal() ? pOutputCallbacks->BufferNormal() : "");
          pDebugControl->ControlledOutput(DEBUG_OUTCTL_ALL_CLIENTS,
            DEBUG_OUTPUT_ERROR , "[Error] \n%s\n", pOutputCallbacks->
            BufferError() ?  pOutputCallbacks->BufferError() : "");
        }

        // Revert the mask (again) for the case
        // where the mask was set but the callback wasn't
        pDebugClient->SetOutputMask(ulOriginalMask);
        pOutputCallbacks->Release();
      }
    }
    pDebugControl->Release();
  }
  return S_OK;
}
```

of their internals. In particular, I've "value-added" some of the SOS debugger commands with hyperlinks. Going through how I parse the output to decide what to mark up is beyond the scope of this article. I will, however, talk about how I determine the correct command to execute to produce the output I require for the parsing.

The enumeration of loaded debugger extensions and their associated abilities isn't part of the Debugger API. To make matters worse, the same debugger extension can be loaded multiple times from the same location on disk. If you run the test_windbg.cmd script with the .load myext and .chain commands, you'll see an example of this situation. The MyExt extension is loaded twice, once as myext and again as myext.dll:

```
0:000> .load myext
0:000> .chain
Extension DLL search Path:
  C:\Debuggers_x86\WINXP;C:\Debuggers_x86\winext;...
Extension DLL chain:
  myext: image 6.1.7600.16385, API 1.0.0, built Sat Jan 29 22:00:48 2011
    [path: C:\Debuggers_x86\myext.dll]
  myext.dll: image 6.1.7600.16385, API 1.0.0, built Sat Jan 29 22:00:48 2011
    [path: C:\Debuggers_x86\myext.dll]
  dbghelp: image 6.12.0002.633, API 6.1.6, built Mon Feb 01 12:08:26 2010
    [path: C:\Debuggers_x86\dbghelp.dll]
    ...
```

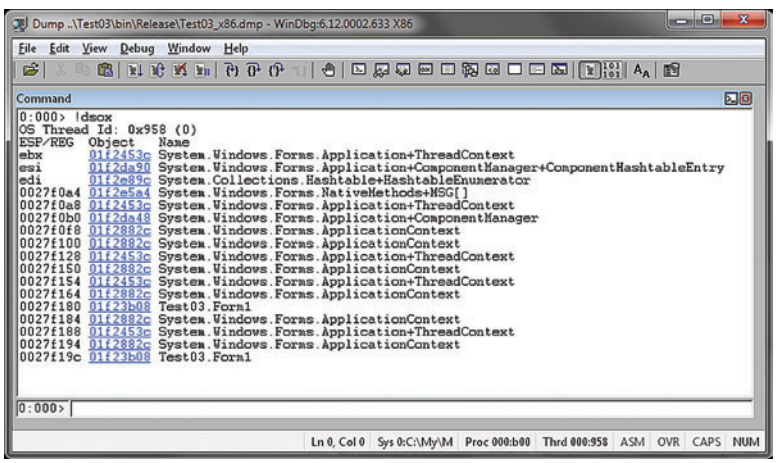


Figure 9 **Test03 x86 !dsox Wrapping !dumpstackobjects**

> The same debugger extension can be loaded multiple times from the same location on disk.

When calling an extension command for parsing, you want to be sure that you're calling the intended command, not a command with the same name on another extension (or another instance).

If you fully qualify a command, the engine simply executes the command specified. But when you don't

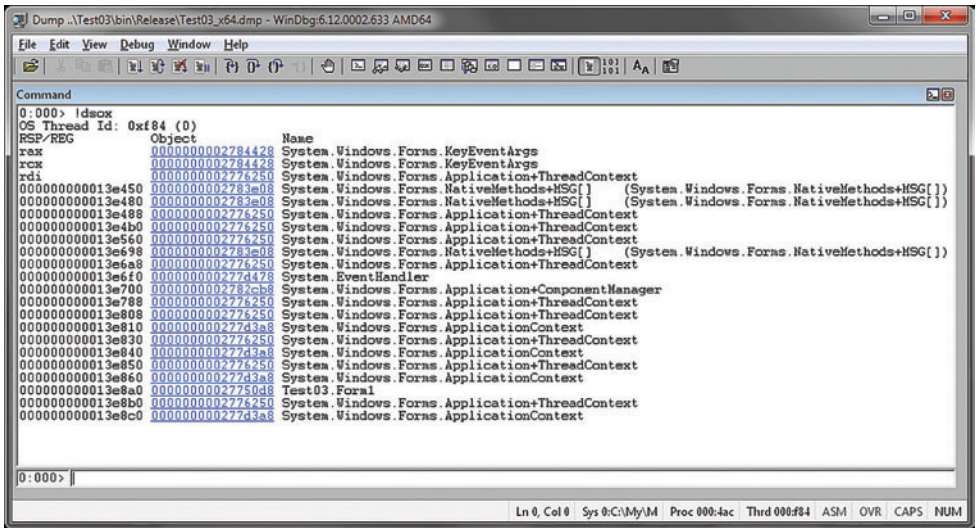


Figure 10 **Test03 x64 !dsox Wrapping !dumpstackobjects**

fully qualify a command, the engine has to do a search. The engine looks at the export table of each extension in the load order and executes the first extension with a matching command name. The .extmatch command displays the current search results for the given command. The .extmatch command supports wildcards of the extension name and the command via the "/e" switch, as shown here:

```
0:000> .extmatch callbackecho
!myext.callbackecho
!myext.dll.callbackecho
0:000> .extmatch /e *myext.dll* *back*
!myext.dll.callbackecho
!myext.dll.callbacktoggle
```

When you load the SOS extension via a .loadby sos.dll mscorwks call, a fully qualified path will be registered for it; that is, it won't be known as "!sos.dll.<command>." In the following example, I ran the Test03 example application on Windows 7 x64 and attached the debugger to it (F6). The SOS extension was loaded from the mscorwks folder (C:\Windows\Microsoft.NET\Framework64\v2.0.50727):

```
0:004> .loadby sos.dll mscorwks
0:004> .chain
Extension DLL search Path:
  C:\debuggers\WINXP;C:\debuggers\winext;...
Extension DLL chain:
  C:\Windows\Microsoft.NET\Framework64\v2.0.50727\sos.dll:
    image 2.0.50727.4952, API 1.0.0, built Thu May 13 05:15:18 2010
    [path: C:\Windows\Microsoft.NET\Framework64\v2.0.50727\sos.dll]
    dbghelp: image 6.12.0002.633, API 6.1.6, built Mon Feb 01 12:15:44 2010
    [path: C:\debuggers\dbghelp.dll]
    ...
```

The only way to safely call an SOS command is to use full qualification. So, when leveraging the SOS extension (or any other extension), there are two facts to determine:

- Is the SOS extension loaded?
- Where is it loaded?

Both can be determined via an extension match (.extmatch /e *sos.dll* <command>). In my LoadSOS function (part of Example08; the code isn't listed here) I first look for the command in the versioned location (\v2.0.50727\sos.dll.<command>). If I can't find the versioned command, I fall back to looking for the command without a version (\sos.dll.<command>). If I can now find it, I still return success, but as S_FALSE (as opposed to S_OK). It's up to the caller of LoadSOS to determine whether it will take an incorrect parsing risk.

If I can't find an SOS version of the command, I assume it isn't loaded (or it's loaded, but not in a way that I'm happy with) and do a private ".loadby." If this doesn't generate any errors, I repeat the search. If I can't load SOS or can't find my command, I return failure (E_FAIL).

Once I have the command, I execute it, rewrite it (for example, ParseDSO) and output the DML version (refer to Example08 in the code download).

In my DML wrappers of SOS commands, I do this check prior to executing a command to be sure that SOS hasn't been unloaded between calls and that the required command exists. **Figures 9** and **10** show my !dsox command in action with x86 and x64 targets.

The commands add a !dumpobj hyperlink to each address listed in !dumpstackobjects; the code for !dsox is in Example08.

Break!

With a little bit of infrastructure code, it's possible to harness the logic of any built-in or extension command. My parsing of the SOS extension is about adding DML, but it could've easily been about the gathering of managed-class information. I didn't have to know anything about the internal structures of the CLR to achieve my goal. I just had to know how to *navigate* to the data I wanted to display.

With a little bit of infrastructure code, it's possible to harness the logic of any built-in or extension command.

Debugger Extensions simplify debug analysis through the automation of data retrieval and the enrichment of display. I hope this series has inspired you to make your own extensions that simplify the debugging you do.

If you're just interested in debugging and want to learn more, you should check out the Advanced Windows Debugging and Troubleshooting blog (blogs.msdn.com/b/ntdebugging)—it has lots of training and case study articles to read. ■

ANDREW RICHARDS *is a Microsoft senior escalation engineer for Windows OEM. He has a passion for support tools and is continually creating debugger extensions and applications that simplify the job of support engineers.*

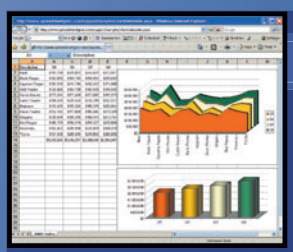
THANKS to the following technical expert for reviewing this article: *Drew Bliss*

20 Minutes to 4 Seconds...

SpreadsheetGear for .NET reduced the time to generate a critical Excel Report "from 20 minutes to 4 seconds" making his team "look like miracle workers."

Luke Melia, Software Development Manager at Oxygen Media in New York

SpreadsheetGear 2010



ASP.NET Excel Reporting

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

Excel Compatible Windows Forms Control

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

Create Dashboards from Excel Charts and Ranges

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

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

www.SpreadsheetGear.com.

SpreadsheetGear

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

Build Multi-Targeted Apps for the Desktop, Prism and Windows Phone 7

Bill Kratochvil

Microsoft provides powerful frameworks for developing scalable, extensible applications. Some recent projects—Prism, the Managed Extensibility Framework (MEF) and Unity—provide multi-targeting capabilities. Multi-targeting permits you to have a *single* code base that can be shared across the desktop, Silverlight and Windows Phone 7 platforms. These free tools have comprehensive documentation with quick starts, labs and plenty of sample code to help get you started. To maximize your return on investment (ROI)—man hours spent researching and learning—it's important to understand that you must approach these tools and documentation with *a new way of thinking*.

Prism, the MEF and Unity use advanced patterns of software development that can make the learning curve difficult for most of us developers—so difficult that it *may appear* to lack an ROI. They require you to not only have a new way of thinking, but to have an understanding of the tools and the patterns involved. Failure in either area will be costly in terms of the learning curve and application stability. Fortunately, if you have a few basic concepts under your belt, you'll not only understand the tools and documentation, you'll be able to open any Prism, MEF or Unity application and quickly navigate through it so you can learn or reuse the best of what's available to create your multi-targeted applications. These concepts include the project linker, dependency injection (DI), sharing code and architectural patterns.

This article discusses:

- The project linker
- Dependency injection
- Sharing code and XML
- Architecture patterns
- Power and versatility

Technologies discussed:

Prism, Managed Extensibility Framework, Windows Phone 7, Unity, Dependency Injection

It's important to understand that you must approach these tools and documentation with *a new way of thinking*.

The Project Linker

It's important to understand that projects can't be shared between platforms. Desktop projects can't reference Silverlight projects and Silverlight projects can't reference desktop projects. Likewise, a Windows Phone 7 project can't reference Silverlight or desktop projects. You aren't sharing projects, you're sharing code. This requires you to create a project for each platform and "link" the files.

Linked files in Visual Studio are identified with a shortcut symbol on the file icon and *do not* physically exist on your hard drive.

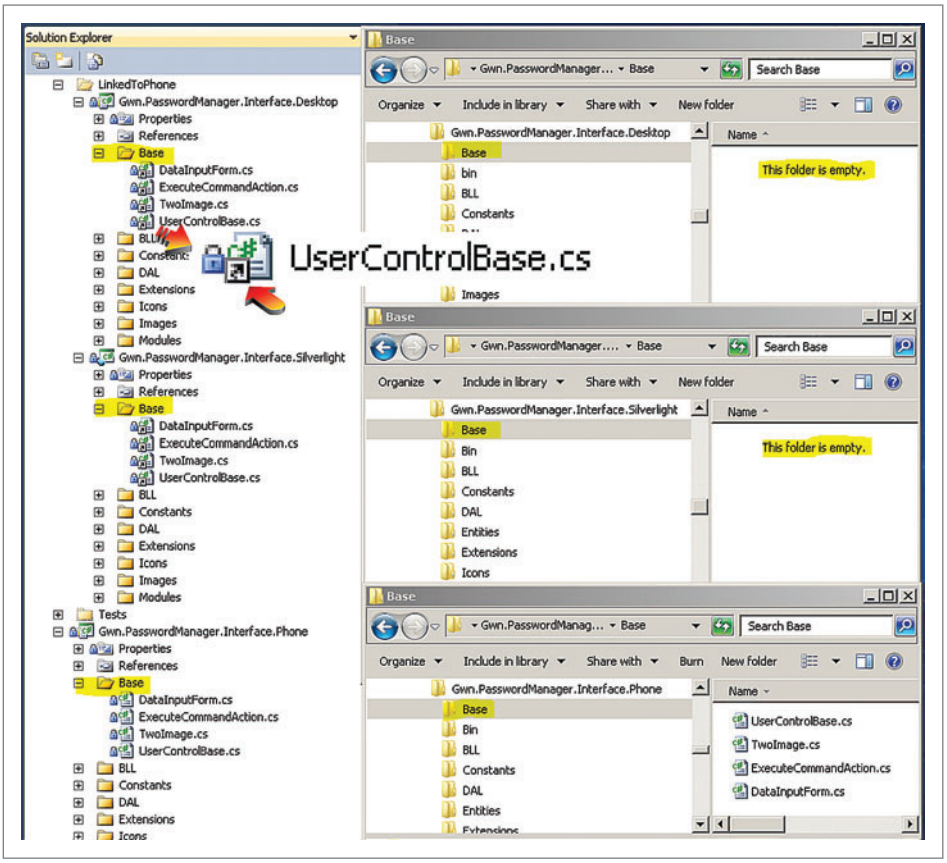


Figure 1 **Only Source Projects Have Code**

Note in **Figure 1** that only the Phone (source) project has code; the Desktop and Silverlight target projects have linked files, so their folders are empty.

The Prism project (compositewpf.codeplex.com) includes a project linker application; without this tool, the management of multi-targeted applications would quickly become a daunting task. It allows you to establish a source project and link to target projects so that any action you perform on the source—for example, add, rename and delete folders and files—will be duplicated in the target projects.

Naming conventions become increasingly important in multi-targeted applications. The linked projects should all share the same name with the exception of a suffix, which represents the platform. For folder structure purposes, it's best to create the project with the suffix—for example, Gwn.Infrastucture.Desktop—and then remove the suffix from the default namespace from the project configuration, as shown in **Figure 2**.

This ensures that your folder structures won't be ambiguous or cause you problems when creating new projects (with the same name) for the other platforms. Removing the suffix from the default namespace is an important step; because we're sharing the code on multiple platforms, the code must not have platform-specific namespaces, or the other platforms will complain during compile time.

The preceding guidance also helps you maintain your sanity during development. If you inadvertently add an assembly reference to a different platform—for example, in **Figure 2** my Desktop application has a reference to a Silverlight assembly—you'll get a confusing error message (because it does exist in the namespace). A quick glance at the project references will reveal the culprit if you used a platform suffix.

Microsoft provides excellent documentation for the project linker at msdn.microsoft.com/library/dd458870. In addition, you can use the Bing search engine to search for "project linker" and find other valuable resources, including a link from myself (aka BillKrat), which has a webcast that walks you through the project linking process. *(Note: The installation process is far easier than noted in documentation if you're using Visual Studio 2010. From the menu options, you can go to Tools | Extension Manager, search for "Project Linker" and then install it from there.)*

Dependency Injection

To understand Prism, the MEF and Unity, you'll require an understanding of DI. Without it, the documentation and samples will seem foreign to you and your learning curve will be harder than it has to be. We'll limit the topic of DI to the Unity container (unity.codeplex.com); once you understand DI, you'll better grasp the MEF (mef.codeplex.com) as well.

Naming conventions become increasingly important in multi-targeted applications.

What's a DI Container? At a high level you can look at a DI container as an over-glorified dictionary class that has the power to create classes. In this dictionary, you can register interfaces and their implementation (or instances)—dictionary.add(IFoo,Foo), for example. When you tell your container to create IFoo—an interface—it will search the dictionary, find it has a type (Foo) and instantiate it. After instantiating Foo, it will see if Foo has any interfaces that need to be resolved, and the chain continues recursively until all children have been successfully created. Once completed, the container returns an instance of Foo.

Constructor and Setter Injection Typically, code statements will instantiate a class so that it can use its resources (**Figure 3**, line 18).

With DI, the classes are resolved (instantiated) externally and instances are injected into the class by the DI container. Unity

supports two types of injection: constructor injection and setter injection. With constructor injection (**Figure 3**, line 25) the container will dynamically create the MyTaskConstructorInjection class, resolve IFoo and pass it in as a constructor parameter. After the Unity container constructs the class and injects parameters, it will search the class for dependency attributes (**Figure 3**, line 33) and resolve the properties. Because setter injection doesn't occur until after constructor injection, you can't reference properties with the dependency attribute in your constructor.

Registering Types In order for the container to resolve interface types, such as IFoo, the implementation must first be registered with the container (see **Figure 4**, lines 49 and 50).

Generally the registration will be made in the application's bootstrapper, module or presenter/controller because these are typically the initialization classes. Once interfaces are registered, the container will know how to resolve an interface when it's encountered in the dependency chain.

Extensible, Scalable Applications For the sake of discussion, let's say that the Foo class is a data access layer (DAL) that your enterprise uses to access your SQL Server database. It's used by hundreds of modules and numerous applications. Management just completed a licensing agreement with Microsoft—for cloud services—and you're notified that you'll have to upgrade your applications to utilize the cloud. During the migration, Foo will continue to be used by some applications (for the SQL Server database), while the others will be moved to the cloud. For those applications migrating to the cloud, you need to be able to switch back to the old database quickly in case the data migration failed for any reason. How long would this take you?

Using DI, it will only take a few lines of code! The pseudocode follows:

```
var  isCloud =  GetCloudConfigurationFromConfigFile();
if(isCloud)
  container.RegisterType<IFoo, FooCloud>();
else
  container.RegisterType<IFoo, Foo>();
```

Because your enterprise applications have integration testing, you can perform test-driven development (TDD) on the new FooCloud DAL. Once TDD is completed, you're ready to update each application's bootstrapper with the preceding code and deploy it with "NO" regression testing required for the applications that continue to use the SQL Server database (Foo class), because the code hasn't changed.

> At a high level you can look at a DI container as an over-glorified dictionary class that has the power to create classes.

The following business logic layer (BLL) could potentially be used by hundreds of presenters/controllers. Because it deals strictly with domain objects, we won't have to touch this code—or any code that consumes this BLL—because we code against a DAL interface (IFoo):

```
public class FinancialDataBll  : IFinancialDataBll
 {
   [Dependency]
     public  IFoo Dal {get;set;}

     public  IEnumerable<FinancialEntity>
       GetFinanicalData(object sender, EventArgs e)
     {
       // Validate event arguments prior to calling data layer
       var returnResults = Dal.GetFinancialData(sender,e);
       // Handle errors with friendly messages as applicable
       return returnResults;
     }
 }
```

Had you tightly coupled your code to the Foo class in **Figure 3**, line 18, then your applications wouldn't be as extensible and the migration process would be considerably more costly in terms of coding and testing.

Layers Effective utilization of DI requires that you properly structure your BLLs and DALs. The BLL should have no knowledge of the DAL components—for example, connection strings,

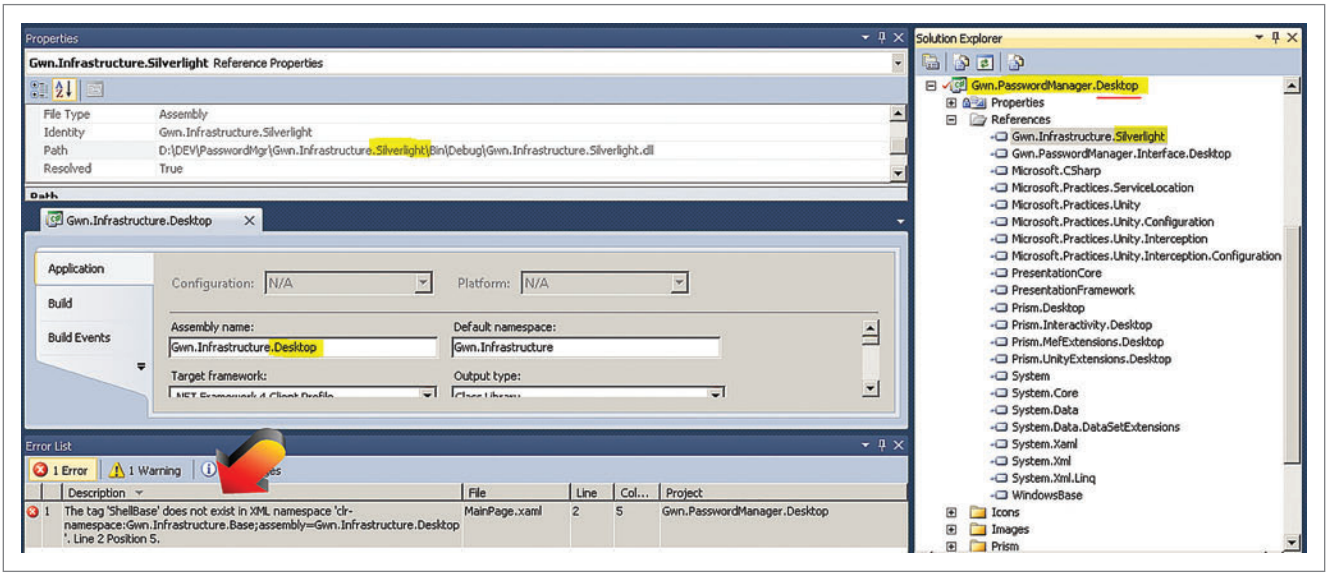
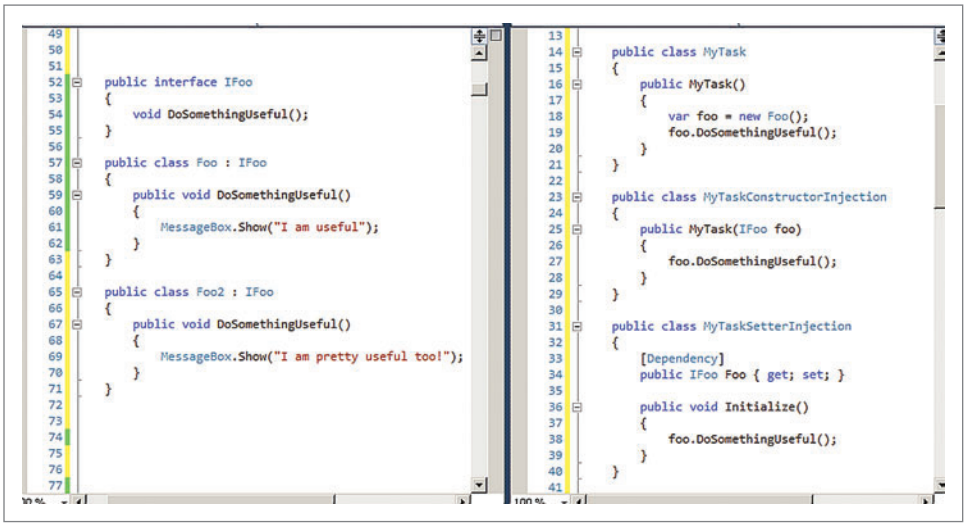


Figure 2 **Remove the Suffix from the Default Namespace After Creating the Project**

SQL statements, file handles as XML files and so on. Likewise, your presentation layer (UI components) should have no knowledge of the DAL; it only knows of the BLL. *(Note: Your BLLs could be swapped out just as easily as the DAL if your presentation layer uses DI to resolve its BLLs; this is invaluable if you have to perform a refactor but only want to affect one application.)*

Event Aggregation Communication between decoupled components can introduce challenges. In the example shown in **Figure 5**, object D contains a drop-down list the user will use to change the currency type—for example, United States dollar to euro.



```
public interface IFoo
{
    void DoSomethingUseful();
}
public class Foo : IFoo
{
    public void DoSomethingUseful()
    {
        MessageBox.Show("I am useful");
    }
}
public class Foo2 : IFoo
{
    public void DoSomethingUseful()
    {
        MessageBox.Show("I am pretty useful too!");
    }
}
```

```
public class MyTask
{
    public MyTask()
    {
        var foo = new Foo();
        foo.DoSomethingUseful();
    }
}
public class MyTaskConstructorInjection
{
    public MyTask(IFoo foo)
    {
        foo.DoSomethingUseful();
    }
}
public class MyTaskSetterInjection
{
    [Dependency]
    public IFoo Foo { get; set; }
    public void Initialize()
    {
        foo.DoSomethingUseful();
    }
}
```

Figure 3 **Constructor and Setter Injection**

The objects B and F are biased by the currently selected currency and have to be recalculated when the currency is changed. One possible solution would be to bubble the event to A (where A subscribes to B, B subscribes to C, and C subscribes to D) and then tunnel the information from A to E, which in turn would tunnel it to F. As the application grows, this process gets more complicated. This complexity would extend to testing, as all affected areas would have to be regression-tested if conditional statements were added anywhere in the path.

To prevent memory leaks for this solution, we would have to ensure we implemented IDisposable on all affected objects and disposed all event subscriptions; this adds another layer of complexity that would have to be managed.

Prism Event aggregation is one of the many areas in which Prism shines. With Prism, you can simply publish an event in D using a syntax comparable to the following:

```
aggregator.GetEvent<MyEvent>().Publish(MyPayload)...
```

Objects B and F would subscribe to this event, so after writing just a few lines of code, you'd be focusing on business logic and wouldn't have to be bogged down in infrastructure plumbing. An example subscription from B and F is shown here:

```
[Dependency]
public ILoggerFacade Logger {get;set;}

private  IEventAggregator _eventAggregator;
[Dependency]
public  IEventAggregator EventAggregator
{
  get { return _eventAggregator; }
  set {
    _eventAggregator=value;
    OnEventAggregatorSet();  // Subscribe to events in this method
  }
}
```

And here's another example:

```
public  MyConstructor(IEventAggregator aggregator){
  aggregator.GetEvent<MyEvent>().Subscribe(HandleMyEvent);
}
public void HandleMyEvent(MyEventArgs e){
  Logger.Log("HandleMyEvent is handling event in Foo", Category.Debug, Priority.None);
 // Do something useful
}
```

Sharing Code and XAML

Before Electronic Health Record (EHR) certification requirements were established (making EHR development cost-prohibitive for small offices), I wrote an EHR application for a small medical group. It started with a Windows Mobile PDA and Web site for the purpose of accurately managing patient care and billing. The PDA had all the business logic, and the Web site provided an interface for the billing staff. Slowly, features of the PDA were migrated to the Web site and the expectation was that all of the business rules and behaviors would be migrated, too. To complicate matters, I was later asked to create a desktop application for their tablets and laptops, and, within a year, the scope of the project changed from a PDA to all three platforms. I found myself having to manage three separate code bases; each had platform-specific variations, which didn't make sharing business logic straightforward.

When the project linker emerged in the Prism project, and the patterns & practices design team shifted gears to support a multi-targeted environment (using the same namespaces across platforms), I was immediately on board because I fully appreciated the ramifications of the direction they were taking. Using multi-targeting, the EHR project could be done in a fraction of the time and at minimal cost to my client for development, upgrades and maintenance.

Code Much thought has to go into the platforms you're going to support. It's important that your single codebase be in the least-supported platform. For example, if you're only going to program in the Silverlight and desktop environments, then you'll set your Silverlight projects as the source and the desktop projects will link to them. If you're going to support Windows Phone 7, Silverlight and the desktop, then the Windows Phone 7 projects need to be the source.

This is important because you don't want to waste your time writing code that won't compile on the other platforms. If, for example, you develop on the Windows Phone 7 platform, you'll quickly see, via IntelliSense, what framework features are available. You can write your code with confidence that it will compile on the other platforms.

It's also recommended that you set a standard for your project's "Conditional compiler symbols" configuration (see **Figure 6**; note that highlighted areas show default settings).

I found this to be an issue because "SILVERLIGHT" is shared in both the Windows Phone 7 and Silverlight projects; to code specifically for Silverlight 4, I had to do the following:

```
#if SILVERLIGHT
#if !WINDOWS_PHONE
  // My Silverlight 4-specific code here
#endif
#endif
```

XAML XAML was surprisingly easy to reuse, particularly between Silverlight and desktop applications; the trick is to not have any assembly references in your XAML. I accomplished this by creating a wrapper class (see **Figure 7**, top-left panel) within the local project that subclasses the actual class in the external assembly.

Notice that my UserControlBase class programmatically loads the platform-specific resource file, which contains the converters, styles and templates that allow me to have minimal XAML declarations that might affect my ability to share it.

> Event aggregation is one of the many areas in which Prism shines.

Architecture Patterns

If you put 10 programmers in a room and assigned them the same development task, it's likely that you would see 10 different ways to successfully complete it. As developers, we have varying levels of architectural and coding experience that could easily be reflected in the code we write (making our enterprise applications hard to follow and maintain). Architectural patterns and standards provide us a common core of experience, which would have the 10 developers coming out of the room with comparable code.

Windows Presentation Foundation (WPF), Prism, the MEF and Unity introduce a development environment unlike any other. The addition of multi-targeting throws in an added level of complexity—particularly if you use it effectively, minimizing platform-specific code. With evolving technology in an agile world, patterns need to be constantly updated to effectively leverage the new technology being offered; it's inevitable that you'll see a multitude and combination of patterns as you review the samples these powerful tools provide.

If you don't recognize patterns, you'll have difficulty understanding the tools, documentation and samples; worse yet, you can use these powerful tools in a way that violates their intended design, which will become a risk to the success of your project. Appendix B in the Prism documentation provides comprehensive information on applicable patterns. I'll summarize the evolution of presentation patterns in the next section.

```
30
31  public class MyTaskConstructorInjection
32  {
33      public MyTask(IFoo foo, ILogger logger)
34      {
35          foo.DoSomethingUseful();
36          logger.Log("Something useful just happend");
37      }
38  }
39
40  public class MyMain
41  {
42      public MyMain()
43      {
44          // Instantiate our container
45          var container = new UnityContainer();
46
47          // Register types required by container
48          container
49              .RegisterType<ILogger, DebugLogger>()
50              .RegisterType<IFoo, Foo>();
51
52          container.Resolve<MyTaskConstructorInjection>();
53      }
54  }
```

Figure 4 **Registering the Implementation with the Container**

Model-View-Controller (MVC) **Application Model** This model emerged as the solution to issues encountered with MVC, more specifically the pollution of domain objects with business logic and UI state, such as text color, a selected item in a list, visibility of the control and so on. It also provided the ability to update the view directly (something not permitted in MVC).

The Application Model was inserted between the View and the Model and contained business logic to manage behavior, state and synchronization with the Model. The View would bind to the Application Model (versus the Model).

MVC Presentation Model The main distinction between the Presentation Model and Application Model lies in its ability to update the View. Presentation Model follows MVC guidelines and doesn't permit the updating of the View directly.

Model-View-ViewModel (MVVM) John Gossman, MVVM creator, states the following:

"On naming, my opinion at this point is the [MVVM] pattern is a WPF-specific version of the Presentation Model pattern. Presentation Model is a lot easier to say and to write, and is more widely known ... but for now, I'm sticking to MVVM terminology in my own thinking in order to capture the WPF-ness and to avoid any interpretation conflicts."

Model-View-Presenter (MVP) MVP was the solution to the limitations of the MVC, Application Model and Presentation Model patterns. With MVP the developer is empowered to control the View and ViewModel. Windows developers have always had the luxury of smart controls to work with, so the distinction between MVC and MVP may not be clear to many of us. With a little research, you'll find that Windows made the MVC controller obsolete because the OS and controls included most of the controller functionality.

The need to evolve from MVC wasn't limited to the obsolete controller; other factors were limitations in the Application Model and Presentation Model as already discussed.

Here's an outline of the components of MVP:

Model The Model contains the data the View will be bound to.

View The View displays the contents of the Model through data binding. This separation is important because a View can be shared by

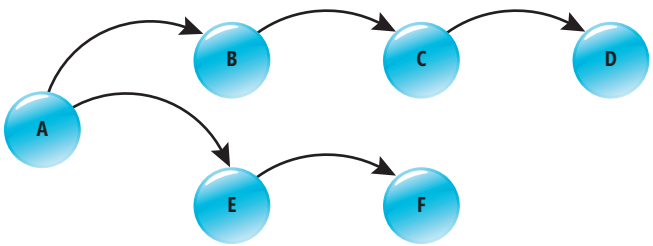


Figure 5 **Communication Between Decoupled Objects**

BEST SELLER

FusionCharts | from $189.05

InfoSoft Global
empowering human thoughts

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

- Liven up your web applications using animated & data-driven charts
- Create AJAX-enabled charts with drill-down capabilities in minutes
- Export charts as images/PDF and data as CSV from charts itself
- Create gauges, dashboards, financial charts and over 550 maps
- Trusted by over 18,000 customers and 375,000 users in 110 countries

BEST SELLER

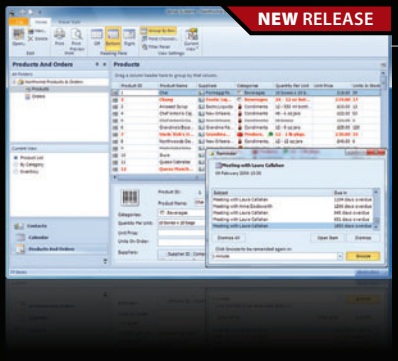
ActiveReports 6 | from $559.20

GrapeCity PowerTools

Latest release of the best selling royalty free .NET report writer.

- Fast and flexible reporting engine
- Flexible event-driven API to completely control the rendering of reports
- Wide range of export and preview formats including Windows Forms viewer, Web viewer, Adobe Flash and PDF
- Royalty-free licensing for Web and Windows applications

NEW RELEASE



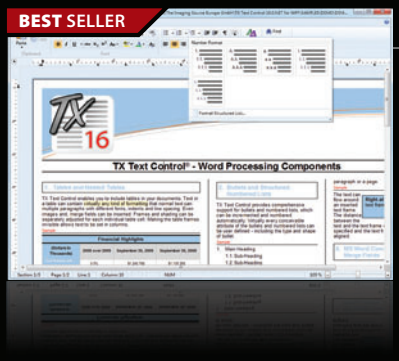
Janus WinForms Controls Suite V4.0 | from $853.44

Janus systems

Add powerful Outlook style interfaces to your .NET applications.

- Includes Ribbon, Grid, Calendar view, Timeline and Shortcut bar
- Now features Office 2010 visual style for all controls
- Visual Studio 2010 and .NET Framework Client Profiles support
- Janus Ribbon adds Backstage Menus and Tab functionality as in Office 2010 applications
- Now features support for multiple cell selection

BEST SELLER



TX Text Control .NET for Windows Forms/WPF | from $564.27

TX TEXT CONTROL
word processing components

Word processing components for Visual Studio .NET.

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

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

GSA Schedule Contract GS-35F-0188R

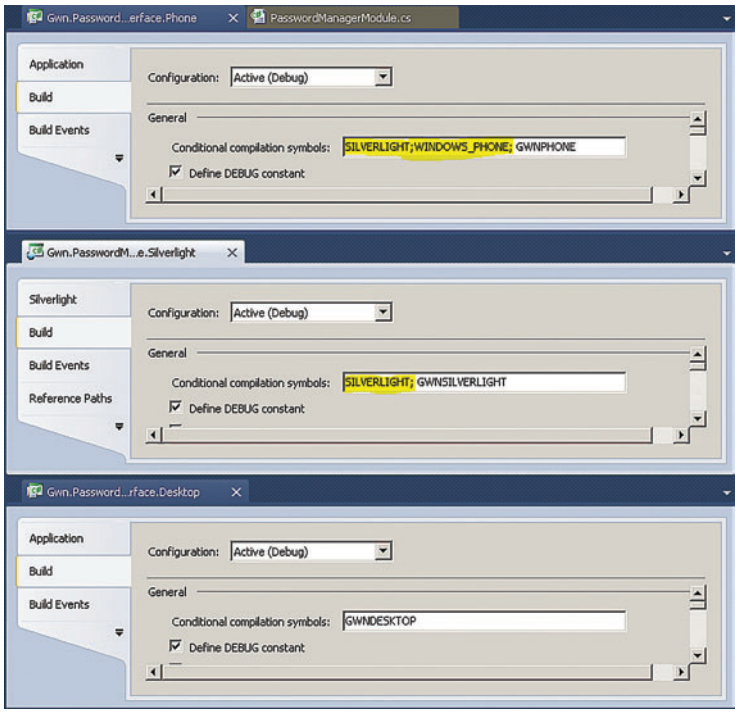


Figure 6 **Set Compiler Symbols for Each Platform**

multiple Presenters. *(Note: In 2006, Martin Fowler split the MVP pattern into two distinctly different patterns: Supervising Controller and Passive View. The main difference between the two is centered on data binding; if data binding is used, then it's referred to as Supervising Controller [following the responsibilities previously outlined]. If there's no data binding, it's referred to as Passive View and the responsibilities shift; it becomes solely the Presenter's responsibility to keep the View synchronized with Model.)*

Presenter The Presenter has knowledge of both the View and the Model, with the responsibility of handling business logic and populating the Model.

Model-View-Presenter, ViewModel (MVPVM) This (unnamed) pattern was found in an early drop of the Prism project; it combined the power of MVP and MVVM:

```
// Load the view into the MainRegion
// after the presenter resolves it
RegionViewRegistry.RegisterViewWithRegion("MainRegion", () =>
  presenter.View);
```

This simple line of code, coupled with lessons learned from the patterns & practices team while using their Prism, Unity, Smart Client Software Factory (SCSF) and Web Client Software Factory (WCSF) projects, would be used to create the pattern that I appropriately call MVPVM. With this pattern, the Presenter is responsible for resolving (instantiating) the ViewModel, View and required BLL components. The Presenter then makes the necessary calls to the applicable BLL methods (see **Figure 8**) to populate the ViewModel with data for the View to display.

Because the business logic resides in the Presenter, the ViewModel can be easily reused by other Presenters. In addition, the Presenter can update the View directly to eliminate complex binding logic (if applicable), particularly when a control reference is required. As such, MVPVM resolves (no pun intended) all of the limitations of Application Model, Presentation Model and MVVM (as outlined by our renowned architects) for our multi-targeted Prism and DI environment.

You'll find the preceding line of code in the SecurityModel of our Password Manager open source project at PasswordMgr.CodePlex.com. This multi-targeting project has desktop, Silverlight and Windows Phone 7 applications *sharing a single codebase*, with the Silverlight and desktop applications *sharing the same XAML*.

(Note: As of this writing, there's no official support for a DI container on the Windows Phone 7 platform. For this project, I downloaded the ContainerModel from the Windows Mobile 6.5 project [mobile.codeplex.com], implemented the IUnityContainer interface [from the Unity project at unity.codeplex.com], pulled over a number of

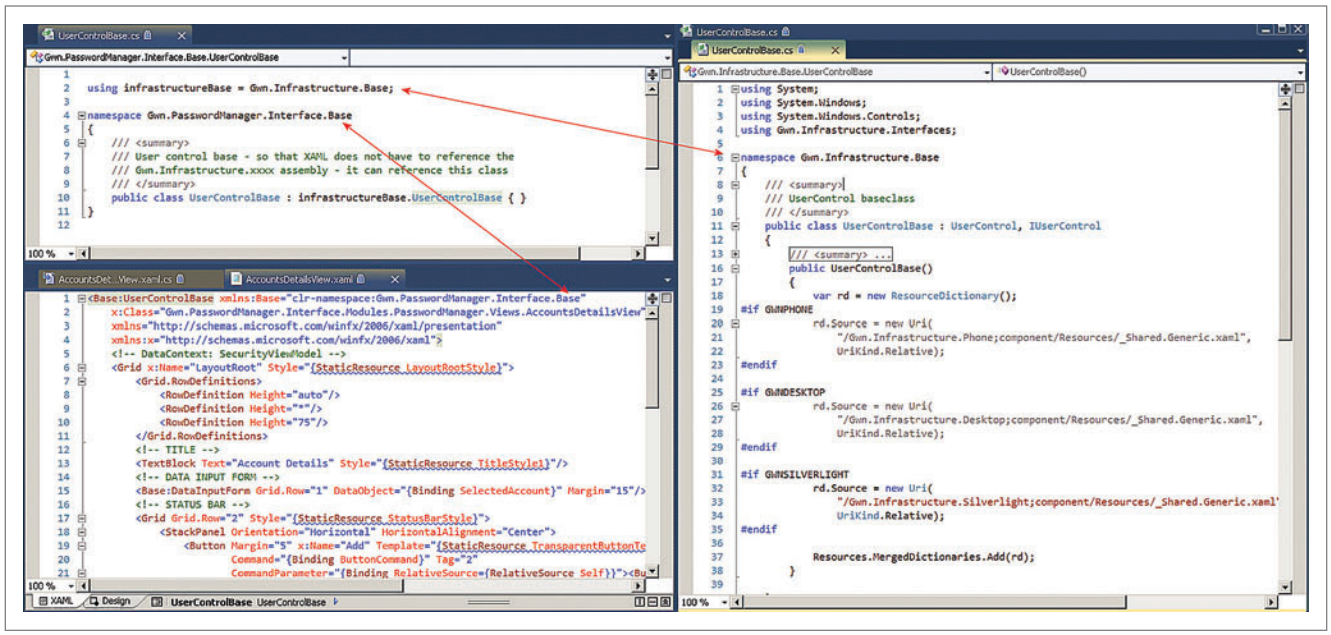


Figure 7 **Create a Wrapper Class for the Current Project**

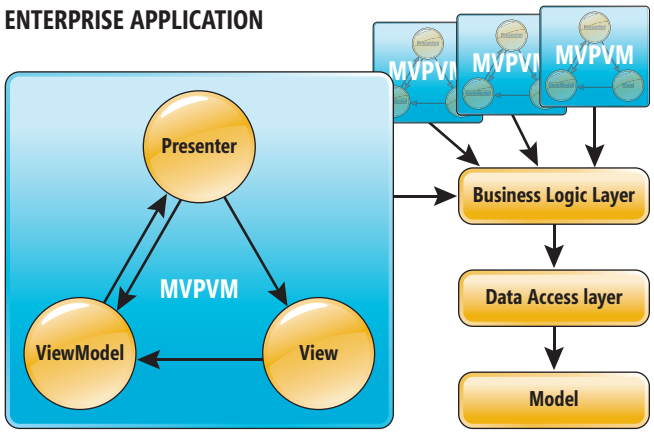


Figure 8 **Model-View-Presenter, ViewModel (MVPVM) and Associated Components**

the Unity unit tests and performed TDD until all unit tests passed. With the newly constructed DI container, which looks and feels like Unity, I was able to port over the Prism project—this Password Manager project has more than 800 unit tests and compiles with zero errors, warnings and messages.)

Model The DAL will have sole responsibility for persisted Model objects. In addition, it has the responsibility of transferring the persisted Model objects to the domain objects (entities that represent the model that are within the scope of all enterprise application layers). *(Note: The BLL has no knowledge of the presentation layer or the Model; it only communicates to the DAL through an interface.)*

View The View observes the ViewModel (data binding) and displays the data on the ViewModel as applicable. A View only has one ViewModel; however, the ViewModel can be shared by multiple Views.

Presenter The Presenter is responsible for instantiating the View and ViewModel, setting the View's data context, wiring up events and handling business logic for any events and behaviors (mouse and button clicks, menu selections and so on).

The Presenter will use (shared) BLLs to handle business logic and perform create, read, update and delete (CRUD) operations on persisted model data. *(Note: BLLs are easy to unit test and can be easily replaced using DI.)*

The Presenter shares the MVP ability to directly modify the View, its controls and the ViewModel.

ViewModel The ViewModel can be reused by numerous Views, so it should always be designed with sharing in mind.

The ViewModel can communicate with the Presenter using either event aggregation (preferred) or through interfaces (for generic functionality). Typically this will be handled by a ViewModel base class (communicating with a Presenter base class). This communication is required because the Presenter handles all business logic, but the View will have its commands and events bound to the ViewModel; these events need to be propagated to the Presenter in a loosely coupled manner for processing.

> Because the business logic resides in the Presenter, the ViewModel can be easily reused by other Presenters.

Power and Versatility

With these basics under your belt, the segment of code shown in **Figure 9** shouldn't appear so foreign to you now.

The power and versatility of the tools the patterns & practices team provides should be apparent, revealing a significant ROI. They abstract the complexity of the infrastructure from you, allowing you to create loosely coupled, scalable and extensible applications—applications that can share the same codebase so that you can accomplish more work with fewer resources in a reliable and stable manner.

Effective utilization of layers (presentation layers, BLLs and DALs) provides a clear separation of concerns, enabling you to reuse your components across multiple modules and applications. This minimizes costs for testing, new development, upgrades and maintenance.

Most importantly, you can create high-velocity, agile teams that can slide through a world of constant change by leveraging their experience and domain knowledge so they can focus their talent on business logic (not infrastructure) in a single codebase, permitting them to construct applications that will work on the Windows Phone 7, Web, tablet or desktop platforms. ■

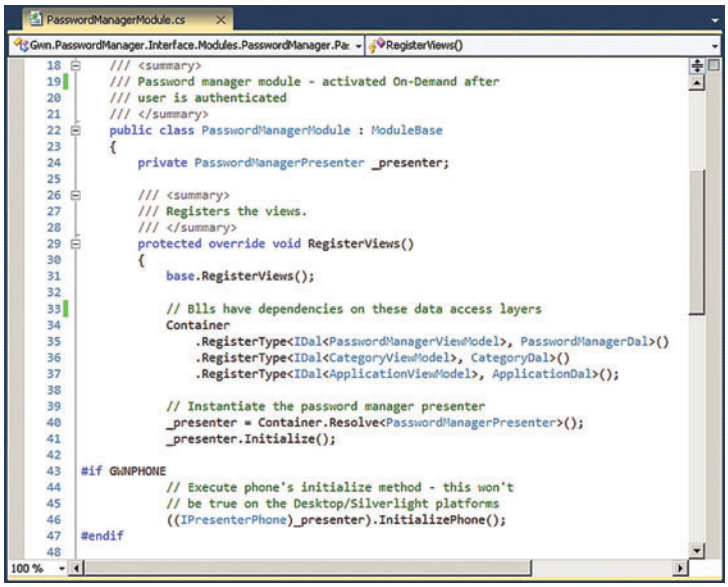


Figure 9 **Module Registrations**

BILL KRATOCHVIL, *an independent contractor, is a lead technologist and architect for an elite team of developers working on a confidential project for a leading company in the medical industry. His own company, Global Webnet LLC, is based in Amarillo, Texas.*

THANKS to the following technical experts for reviewing this article: *Christina Helton and David Kean*

TEAM FOUNDATION SERVER HOSTING IS HERE!

ONLY $20/mo per user

LIMITED TIME SPECIAL OFFER: FIRST 30 DAYS FREE & NO SETUP FEES

Instead of spending your valuable time and resources maintaining a TFS server environment, you have a NEW option - to use the DiscountASP.NET Hosted TFS SaaS solution The DiscountASP.NET Hosted TFS service is a SaaS solution for source code version control and bug tracking. Start saving money and time so you can focus on what you do best – developing killer software!

SIMPLIFY YOUR LIFE CYCLE TODAY!

TFS HOSTING FEATURES:

- Source Code Version Control
- Bug/Issue Tracking
- Web Team Access
- Unlimited Projects
- 5gb of Disk Space
- Visual Studio 2010 Integration
- Visual Studio 2008 Integration
- Web-Based Control Panel
- Month-to-month Billing

New TFS Add-on:
Urban Turtle – scrum tool for agile project management.

FOR MORE INFO: www.DiscountASP.NET/tfs/simplify

discountASP.net
Team Foundation Server Hosting

DevPro 2010 Community Choice GOLD

MEMBERS CHOICE WINNER THE CODE PROJECT 2010

2010 READERS MERIT Visual Studio MAGAZINE

Microsoft GOLD CERTIFIED Partner

Master Large Data Streams with Microsoft StreamInsight

Rob Pierry

Discovering that the production line has gone down, users' media streams are skipping or one of your products has become a "must have" is easy once it has already happened. The real trick is identifying these scenarios as they happen or even predicting them based on past trends.

Successfully predicting scenarios like these requires a near-real-time approach. By the time relevant data is extracted, transformed and loaded into a traditional business intelligence (BI) solution like SQL Server Analysis Services (SSAS), the situation has long since changed. Similarly, any system that relies on a request-response pattern to request updated data from a transactional data store (such as a SQL Server Reporting Services, or SSRS, report) is always operating on stale data near the end of its request-polling interval. The polling interval is usually fixed, so even if a burst of interesting activity occurs, the consuming system won't know until the next interval comes around. Instead, the consuming system should be continuously notified the moment interesting criteria are met.

When detecting emerging trends, intervals of time are key—100 purchases of a specific item in the last five minutes is clearly a bigger indicator of an emerging trend than a steady trickle over the last five months. Traditional systems like SSAS and SSRS require the developer to keep track of the timeliness of data on their own, through a separate dimension in a cube or a time-stamp column in a transactional store. Ideally, tools for identifying emerging scenarios would have the concept of time built-in and provide a rich API for working with it.

Finally, a good indicator of the future comes from analyzing the past. In fact, this is what traditional BI is all about—aggregating and analyzing large volumes of historical data to identify trends. Unfortunately, different tools and query languages are involved when using these systems when compared to more transactional systems. Successfully identifying emerging scenarios requires seamless correlation of past and present data. This kind of tight integration is only possible when the same tools and query language are used for both.

For specific scenarios such as production-line monitoring, highly specific custom tools exist to perform these functions, but these tools are often comparatively expensive and not general purpose.

To prevent the production line from going down or to make sure your products are priced appropriately, the key is to be

This article discusses:

- StreamInsight architectural overview
- Queries and windows
- User-defined aggregates
- Creating custom adapters

Technologies discussed:

SQL Server 2008 R2

Code download available at:

code.msdn.microsoft.com/mag201106StreamInsight

responsive enough to identify and adjust as the situation changes. To easily and quickly identify these scenarios, historical and real-time queries should use the same developer-friendly toolsets and query languages, the system should handle large volumes of data in near-real time (on the order of hundreds of thousands of events per second), and the engine should be flexible enough to handle scenarios across a wide set of problem domains.

Fortunately, this tool exists. It's called Microsoft StreamInsight.

StreamInsight Architectural Overview

StreamInsight is a complex event-processing engine capable of handling hundreds of thousands of events per second with extremely low latency. It can be hosted by any process, such as a Windows Service, or embedded directly in an application. StreamInsight has a simple adapter model for getting data in and out, and queries over both real-time and historical data use the same LINQ syntax accessed just like any other assembly from any Microsoft .NET Framework language. It's licensed as part of SQL Server 2008 R2.

The high-level architecture of StreamInsight is quite simple: events are collected from a variety of sources via input adapters. These events are analyzed and transformed via queries, and the results of the queries are distributed to other systems and people via output adapters. **Figure 1** shows this simple structure.

In the same way that service-oriented architectures are concerned with messages and database systems are concerned with rows, complex event-processing systems like StreamInsight are organized around events. An event is a simple piece of data along with the time at which that data is relevant—like a sensor reading at a particular time of day or a stock ticker price. The data carried by the event is called its payload.

StreamInsight supports three types of events. Point events are events that are instant and have no duration. Interval events are events whose payloads are relevant for a specific period of time. Edge events are similar to interval events, except the duration of the event isn't known upon arrival of the event. Instead, the start time is set and the event effectively has infinite duration until another edge event arrives that sets the end time. For example, a speedometer reading might be a point event because it changes constantly, but the price of milk at the supermarket could be an edge event because it's relevant for a longer period of time. When the retail price of milk changes (say, due to a change in distributor pricing), the duration of the new price isn't known, so an edge event is more appropriate than an interval one. Later, when the distributor updates their pricing again, a new edge event can cap the duration of the previous price change while another edge event will set a new price going forward.

Input and output adapters in StreamInsight are an abstract example of the Adapter design pattern. The StreamInsight engine operates over its own representation of events, but the actual sources of these events can vary wildly, ranging from a proprietary interface to a hardware sensor to status messages generated by applications in the enterprise. Input adapters transform the source events into a stream of events that the engine understands.

The results from StreamInsight queries represent specific business knowledge and can be highly specialized. It's important that these results be routed to the most appropriate place. Output adapters can be used to turn the internal representation of an event into text printed to the console, a message sent via Windows Communication Foundation (WCF) to another system for processing, or even a point on a chart in a Windows Presentation Foundation application. Sample adapters for working with text files, WCF, SQL and more are available at streaminsight.codeplex.com.

StreamInsight Queries by Example

At first glance, StreamInsight queries appear similar to querying rows from a database, but there are critical distinctions. When querying a database, the query is constructed and executed and the results are returned. If the underlying data changes, the output isn't affected because the query has already run. Database query results represent a snapshot of a moment in time, available via the request-response paradigm.

StreamInsight queries are standing queries. As new input events arrive, the query continuously reacts and new output events are created, if necessary.

The query examples in this article are drawn from the sample solution available for download. They start simply, but grow more powerful as new features of the query language are introduced. The queries all use the same class for the payload. Here's the definition of a simple class with properties for Region and Value:

```
public class EventPayload {
  public string Region { get; set; }
  public double Value { get; set; }

  public override string ToString() {
    return string.Format("{0}\t{1:F4}", Region, Value);
  }
}
```

The queries in the sample application make use of an input adapter that randomly generates data and an output adapter that simply writes each event to the console. The adapters in the sample application are simplified for clarity.

To run each query, uncomment the line in the Program.cs file in the sample solution that assigns the query to the local variable named "template."

Here's a basic query that filters the events by the Value property:

```
var filtered =
  from i in inputStream
  where i.Value > 0.5
  select i;
```

This query should look familiar to any developer with experience using LINQ. Because StreamInsight uses LINQ as its query

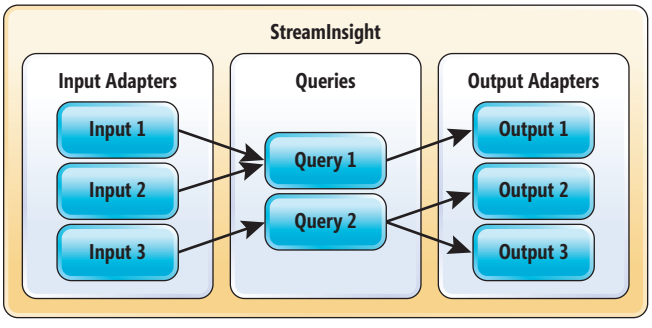


Figure 1 **High-Level Architecture of Microsoft StreamInsight**

language, this query looks just like a LINQ to SQL query hitting a database or an in-memory filtering of an IList. As events arrive from the input adapter, their payloads are inspected, and if the value of the Value property is greater than 0.5, they're passed to the output adapter where they're printed to the console.

When the application runs, notice that events continually arrive in the output. This is effectively a push model. StreamInsight computes new output events from inputs as they arrive, rather than a pull model like a database where the application must periodically poll the data source to see if new data has arrived. This fits nicely with the support of IObservable available in the Microsoft .NET Framework 4, which will be covered later.

Having a push model for continuous data instead of polling is nice, but the real power of StreamInsight becomes apparent when querying over properties relating to time. As events arrive through the input adapter, they're given a timestamp. This timestamp may come from the data source itself (suppose the events represent historical data with an explicit column storing the time) or can be set to the time the event arrived. Time is, in effect, first class in the querying language of StreamInsight.

Queries often look like standard database queries with a time qualifier stuck on the end, such as "every five seconds" or "every three seconds over a five-second span." For example, here's a simple query that finds the average of the Value property every five seconds:

```
var aggregated =
  from i in inputStream
    .TumblingWindow(TimeSpan.FromSeconds(5),
    HoppingWindowOutputPolicy.ClipToWindowEnd)
  select new { Avg = i.Avg(p => p.Value)};
```

Windows of Data

Because the concept of time is a fundamental necessity to complex event-processing systems, it's important to have a simple way to work with the time component of query logic in the system. StreamInsight uses the concept of windows to represent groupings by time. The previous query uses a tumbling window. When the application runs, the query will generate a single output event every five seconds (the size of the window). The output event represents the average over the last five seconds. Just like in LINQ to SQL or LINQ to Objects, aggregation methods like Sum and Average can roll up events grouped by time into single values, or Select can be used to project the output into a different format.

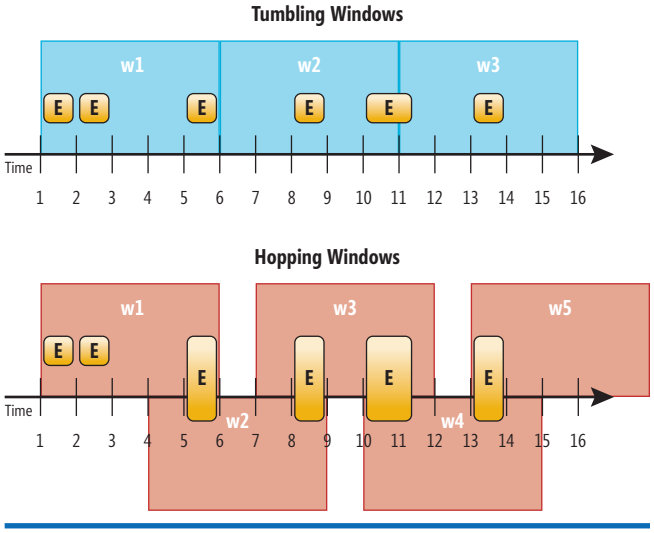


Figure 2 **Tumbling and Hopping Windows**

Tumbling windows are just a special case of another window type: the hopping window. Hopping windows have a size, too, but they also have a hop size that isn't equal to their window size. This means hopping windows can overlap each other.

For example, a hopping window with a window size of five seconds and a hop size of three seconds will produce output every three seconds (the hop size), giving you the average over the last five seconds (the window size). It hops forward three seconds at a time and is five seconds long. **Figure 2** shows an event stream grouped into tumbling and hopping windows.

Notice that the tumbling windows do not overlap, but the hopping windows can if the hop size is smaller than the window size. If the windows overlap, an event may end up in more than one, like the third event, which is in both window 1 and window 2. Edge events (that have duration) may also overlap window boundaries and end up in more than one window, like the second-to-last event in the tumbling window.

Another common window type is the count window. Count windows contain a specific number of events rather than events at a particular point or duration of time. A query to find the average of the last three events that arrived would use a count window. One current limitation of count windows is that the built-in aggregation methods like Sum and Average are not supported. Instead, you must create a user-defined aggregate. This simple process is explained later in the article.

The final window type is the snapshot window. Snapshot windows are easiest to understand in the context of edge events. Every time an event begins or ends, the current window is completed and a new one starts. **Figure 3** shows how edge events are grouped into snapshot windows. Notice how every event boundary triggers a window boundary. E1 begins and so does w1. When E2 begins, w1 is completed and w2 begins. The next edge is E1 ending, which completes w2 and starts w3. The result is three windows: w1 containing E1, w2 containing E1 and E2, and w3 containing E3. Once the events are grouped into the windows, they're stretched so that it appears that the event begins and ends when the window does.

More Complex Queries

Given these available windows and basic query methods like where, group by and order by, a wide variety of queries is possible. Here's a query that groups the input events by region and then uses a hopping window to output the sum of the payload Value for each Region for the last minute:

```
var payloadByRegion =
  from i in inputStream
  group i by i.Region into byRegion
  from c in byRegion.HoppingWindow(
    TimeSpan.FromMinutes(1),
    TimeSpan.FromSeconds(2),
    HoppingWindowOutputPolicy.ClipToWindowEnd)
  select new {
    Region = byRegion.Key,
    Sum = c.Sum(p => p.Value) };
```

These windows use a hop size of two seconds, so the engine sends output events every two seconds.

Because the query operators are defined on the IQueryable interface, composing queries is possible. The following code uses the previous query that finds sums by region to then calculate the region with the highest sum. A snapshot window allows the event stream to be sorted by sum so that the Take method can grab the region with the highest sum:

```
var highestRegion =
  // Uses groupBy query
  (from i in payloadByRegion.SnapshotWindow(
    SnapshotWindowOutputPolicy.Clip)
    from sumByRegion in i
    orderby sumByRegion.Sum descending
    select sumByRegion).Take(1);
```

A common scenario is a query that relates a stream of fast-moving events (like the reading from a sensor) to slower-moving or static reference data (like the fixed location of the sensor). Queries use joins to accomplish this goal.

The StreamInsight join syntax is the same as any other LINQ join, with one important caveat: events will only join together if their durations overlap. If sensor 1 reports a value at time t1, but the reference data about sensor 1's location is only valid for time t2-t3, then the join will not match. The join criteria for duration isn't explicitly written into the query definition; it's a fundamental property of the StreamInsight engine. When working with static data, it's common for the input adapter to effectively treat the data as edge events with infinite duration. This way, all the joins to the fast-moving event streams will succeed.

Correlating multiple event streams via joins is a powerful concept. Assembly lines, oil production facilities or high-volume Web sites don't often fail due to isolated events. One piece of equipment triggering a temperature alarm doesn't usually bring the line down; it's a combination of circumstances such as the temperature being too high over a sustained period of time while a specific tool is in heavy use and the human operators are changing shifts.

Without joins, the isolated events wouldn't have as much business value. Using joins and StreamInsight queries over historical data, users can correlate isolated streams into highly specific monitoring criteria that are then monitored in real time. A standing query can look for the situations that would lead to failure and automatically generate an output event that could be routed to a system that knows how to take the overheating piece of equipment offline instead of waiting until it brings down the whole line.

In a retailing scenario, events relating to sales volume by item over time can feed into pricing systems and customer order histories to ensure optimal pricing per item, or to drive which items to suggest to the user before checkout. Because queries are easily created, modified and composed, you can start out with simple scenarios and refine them over time, yielding increasing value to the business.

User-Defined Aggregates

StreamInsight ships with several of the most common aggregate functions including Count, Sum and Average. When these functions aren't enough (or you need to aggregate over a count window, as mentioned earlier), StreamInsight supports user-defined aggregate functions.

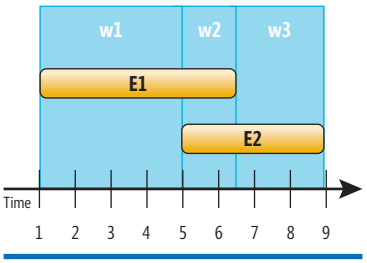


Figure 3 **Snapshot Windows**

The process of creating a user-defined aggregate involves two steps: writing the actual aggregate method, then exposing the method to LINQ via an extension method.

The first step involves inheriting from either CepAggregate<TInput, TOutput> if the aggregate is not time-dependent, or CepTimeSensitiveAggregate<TInput,TOutput> if it is. These abstract classes have a single method to be implemented called GenerateOutput. **Figure 4** shows the implementation of the EveryOtherSum aggregate, which adds up every other event.

The second step involves creating an extension method on CepWindow<TPayload> so that your aggregate can be used in queries. The CepUserDefinedAggregateAttribute is applied to the extension method to tell StreamInsight where to find the implementation of the aggregate (in this case, the class created in the first step). The code for both steps of this process is available in the EveryOtherSum.cs file in the downloadable sample application.

More About Adapters

Queries represent business logic that acts over data provided by adapters. The sample application uses a simple input adapter that generates random data and an output adapter that writes to the console. Both follow a similar pattern, which is also followed by the adapters available on the CodePlex site.

StreamInsight uses a Factory pattern for creating adapters. Given a configuration class, the factory creates an instance of the appropriate adapter. In the sample application, the configuration classes for the input and output adapters are quite simple. The output adapter configuration has a single field to hold a format string to use when writing the output. The input adapter configuration has a field for the time to sleep between generating random events as well as another field called CtiFrequency.

The Cti in CtiFrequency stands for Current Time Increment. StreamInsight uses Cti events to help ensure that events are delivered in the correct order. By default, StreamInsight supports events arriving out of order. The engine will automatically order them appropriately when passing them through the queries. However, there's a limit to this reordering.

Suppose events really could arrive in any order. How would it ever be possible to determine that the earliest event had arrived and could thus be pushed through the query? It wouldn't be, because the next event might have a time earlier than the earliest you've already received. StreamInsight uses Cti events to signal the engine that no more events earlier than what have already been received will arrive. Cti events effectively cue the engine to process the events that have arrived and subsequently ignore or adjust any with timestamps earlier than the current time.

The sample input adapter generates an ordered event stream, so it automatically inserts a Cti event after every generated event to keep things moving along. If you're ever writing an input adapter and your program produces no output, make sure your adapter is inserting Ctis, because without them the engine will wait forever.

StreamInsight ships with a variety of base classes for adapters: typed, untyped, point, interval and edge. Typed adapters always produce events with a well-known payload type—in the sample case, the class RandomPayload. Untyped adapters are useful for event sources that may generate multiple types of events or things like CSV files, where the layout and content of the rows isn't known in advance.

The sample input adapter has a well-known payload type and generates point events, so it inherits from TypedPointInputAdapter<RandomPayload>. The base class has two abstract methods that must be implemented: Start and Resume. In the sample, the Start method enables a timer that fires at the interval specified by the configuration. The timer's Elapsed event runs the ProduceEvent method, which does the main work of the adapter. The body of this method follows a common pattern.

First, the adapter checks whether the engine has stopped since it last ran, and that it's still running. Then a method in the base class is called to create an instance of a point event, its payload is set and the event is queued in the stream. In the sample, the SetRandomEventPayload method stands in for any real adapter logic—for example, reading from a file, talking to a sensor or querying a database.

The input adapter factory is also simple. It implements an interface ITypedInputAdapterFactory<RandomPayloadConfig> because it's a factory for typed adapters. The only trick to this factory is that it also implements the ITypedDeclareAdvanceTimeProperties<RandomPayloadConfig> interface. This interface allows the factory to handle inserting the Ctis, as explained earlier.

The sample application's output adapter follows almost exactly the same pattern as the input. There's a configuration class, a factory and the output adapter itself. The adapter class looks a lot like the input adapter. The main difference is that the adapter removes events from the queue rather than queueing them. Because Cti events are events just like the others, they arrive at the output adapter, too, and are simply ignored.

Observables

Though the adapter model is quite simple, there's an even easier way to get events into and out of the engine. If your application is using the embedded deployment model for StreamInsight, you can use both IEnumerables and IObservables as inputs and outputs for the engine. Given an IEnumerable or an IObservable, you can create an input stream by calling one of the provided extension methods like ToStream, ToPointStream, ToIntervalStream or ToEdgeStream. This creates an event stream that looks exactly the same as one created by an input adapter.

Likewise, given a query, extension methods like ToObservable/Enumerable, ToPointObservable/Enumerable, ToIntervalObservable/Enumerable or ToEdgeObservableEnumerable will route the query output to an IObservable or IEnumerable, respectively. These patterns are especially useful for replaying historical data saved in a database.

Using the Entity Framework or LINQ to SQL, create a database query. Use the ToStream extension method to convert the database results into an event stream and define a StreamInsight query over it. Finally, use ToEnumerable to route the StreamInsight results into something that you can easily foreach over and print out.

Figure 4 **EveryOtherSum Aggregate**

```
public class EveryOtherSum :
  CepAggregate<double, double> {

  public override double GenerateOutput(
    IEnumerable<double> payloads) {

    var sum = default(double);
    var include = true;
    foreach (var d in payloads) {
      if (include) sum += d;
      include = !include;
    }
    return sum;
  }
}
```

Deployment Model and Other Tools

To use the Observable and Enumerable support, StreamInsight must be embedded in your application. StreamInsight does support a standalone model, though. When installing, you're asked whether to create a Windows Service to host the default instance. The service can then host StreamInsight, allowing multiple applications to connect to the same instance and share adapters and queries.

Communicating with a shared server instead of an embedded one simply involves a different static method on the Server class. Instead of calling Create with the instance name, call Connect with an EndpointAddress that points to the shared instance. This deployment strategy is more useful for enterprise scenarios where multiple applications would want to consume shared queries or adapters.

In both cases, it's sometimes necessary to figure out why the output generated by StreamInsight isn't what it should be. The product ships with a tool called the Event Flow Debugger for just this purpose. The use of the tool is beyond the scope of this article, but in general it allows you to connect to instances and trace events from the input through queries and to the output.

A Flexible, Reactive Tool

Flexible deployment options, a familiar programming model and easily creatable adapters make StreamInsight a good choice for a wide variety of scenarios. From a centralized instance querying and correlating thousands of sensor inputs a second, to an embedded instance monitoring current and historical events within a single application, StreamInsight utilizes developer-friendly frameworks like LINQ to enable highly customizable solutions.

Simple-to-create adapters and built-in support for converting between event streams and IEnumerables and IObservables make it easy to quickly get solutions up and running, so the incremental work of creating and refining queries that encapsulate specific business knowledge can begin. As they're refined, these queries provide more and more value, enabling applications and organizations to identify and react to interesting scenarios as they occur, rather than after the window of opportunity has passed. ■

Rob Pierry *is a principal consultant with Captura (capturaonline.com), a consulting company that delivers innovative user experiences backed by scalable technology. He can be reached at rpierry+msdn@gmail.com.*

Thanks to the following technical experts for reviewing this article: *Ramkumar Krishnan, Douglas Laudenschlager and Roman Schindlauer*

WINDOWS FORMS / WPF / ASP.NET / ACTIVEX

WORD PROCESSING COMPONENTS

MILES BEYOND RICH TEXT

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

TX TEXT CONTROL®
word processing components

Microsoft Visual Studio PARTNER

Word Processing Components
for Windows Forms & ASP.NET

WWW.TEXTCONTROL.COM

TX Text Control Sales:
US +1 877 - 462 - 4772 (toll-free)
EU +49 421 - 4270671 - 0

Sterling for Isolated Storage on Windows Phone 7

Jeremy Likness

The launch of Windows Phone 7 provided an estimated 1 million Silverlight developers with the opportunity to become mobile coders practically overnight.

Applications for Windows Phone 7 are written in the same language (C# or Visual Basic) and on a framework that's nearly identical to the browser version of Silverlight 3, which includes the ability to lay out screens using XAML and edit them with Expression Blend. Developing for the phone provides its own unique challenges, however, including special management required when the user switches applications (called "tombstoning") combined with limited support for state management.

Sterling is an open source database project based on isolated storage that will help you manage local objects and resources in your Windows Phone 7 application as well as simplify the tombstoning process. The object-oriented database is designed to be lightweight, fast and easy to use, solving problems such as persistence, cache and state management. It's non-intrusive and works with your existing type definitions without requiring that you alter or map them.

In this article, Windows Phone 7 developers will learn how to leverage the Sterling library to persist and query data locally on the phone with minimal effort, along with a simple strategy for managing state when an application is deactivated during tombstoning.

This article discusses:

- Tombstoning basics
- Saving state on the phone
- Serialization options
- A sample recipe application
- Setting up the database
- Seeding and saving data
- The Main ViewModel: categories and recipes
- Foreign keys
- Querying with indexes
- Practical tombstoning

Technologies discussed:

Silverlight, Sterling, Windows Phone 7

Code download available at:

code.msdn.microsoft.com/mag201106Sterling

Tombstoning Basics

Silverlight in the browser and on the phone runs in a special "security sandbox" that isolates the host system from the application's runtime environment. The phone environment adds a layer of complexity because multiple applications can co-exist on the platform. While the underlying OS of the phone supports multitasking, third-party applications aren't provided with access to this layer. Instead, applications have the opportunity to run "in the foreground" but can be quickly swapped out to make way for other applications, incoming phone calls or hardware features such as the Back button and search. When the application is deactivated, there's a chance

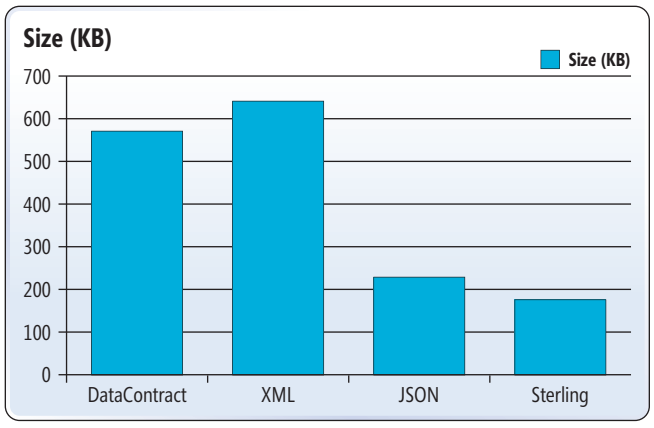


Figure 1 **Size-on-Disk Comparison of Various Serialization Strategies**

it may be "killed," with the opportunity to be revived if and when the user navigates back. This is the process known as tombstoning.

The "Mango" update to Windows Phone 7 will limit tombstone scenarios by providing "fast application switching." Applications will no longer be automatically tombstoned. While this functionality is already present in the Windows Phone 7 Developer Tools, it's important to recognize that it doesn't eliminate the tombstone scenario. Factors such as other running applications and available memory will influence whether the application is tombstoned.

The problem with tombstoning is that when the application is revived, new instances of the pages that are part of the application are created. Therefore, anything the user was working on when the event occurred—such as selecting an item from a pick list or entering text—is lost. It's up to you, the developer, to maintain this state and restore it when the application returns to provide a seamless experience to the user. Imagine the confusion of a typical user who's in the middle of filling out a form and clicks on search to look up a term, only to return and find the application has navigated to a separate, blank page!

Saving State on the Phone

Fortunately, Windows Phone 7 does provide several mechanisms for saving state. To work with applications on the OS, you'll need to become familiar with these methods. The options include SQL CE (with the Mango update), a page state dictionary, the application state dictionary and isolated storage. I'll focus on the last option, isolated storage, but a quick review of the first three will help shed light on why Sterling is useful on the phone.

SQL CE The Mango update will provide SQL CE, a compact version of the popular SQL Server database. The difference between this database and the other options listed here is that SQL is a relational database. It relies on special table formats that must be mapped from your classes and controls. Object-oriented databases can take the existing class structures—even when they include nested classes, lists and other types—and serialize them without any additional mapping or modifications.

Page State Each Page object provides a State property that exposes a dictionary that's defined by key names and related objects. Any key or object may be used, but the object must be serializable, and then only shallow (top level) serialization occurs. In addition, the page state only allows for up to 2MB of data per page (4MB for the entire application) and can only be used after the start of the OnNavigatedTo and before the OnNavigatedFrom methods on the page. This limits the practical use to simple value types. The fact that it can only function within the page navigation methods makes it a poor choice for patterns such as Model-View-ViewModel (MVVM) that synchronize the view state using a separate ViewModel.

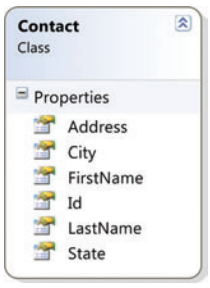


Figure 2: **The Contact Class**

Application State This is also a dictionary. Like the page state, it takes a string key and an object value, and the object must be serializable. The application object on the phone has a Deactivated event called when tombstoning occurs, and an Activated event called when the application returns from the tombstone state. The application state can be accessed anytime between activation and deactivation. It's accessed using the static PhoneApplicationService class (there's a Current property that references a single application-wide instance). There's no documented limit to the size of the application state dictionary, but at some point attempting to store too many items will result in an unhandled COM exception being thrown.

> Sterling is an open source database project based on isolated storage that will help you manage local objects and resources.

Isolated Storage This is by far the most flexible option for maintaining state. Isolated storage isn't unique to the phone, and in fact works almost exactly the same way in Silverlight and in the Microsoft .NET Framework runtime. Isolated storage provides a layer of abstraction from the host file system, so instead of dealing with direct file storage, you interface with an indirect mechanism that provides folders and files in an isolated sandbox. In Windows Phone 7, that sandbox is isolated to the level of your phone application. You may access the storage anywhere from within the application, but not from any other application on the phone.

Isolated storage also has some powerful advantages. While it does provide a settings dictionary similar to the page and application settings described earlier, it also allows you to organize data in folders and as files. In fact, any type of file—XML, binary or text—can be created and accessed in isolated storage. There's no quota for the size of isolated storage on the phone, so you're effectively limited by the amount of memory and storage available on the phone. The only drawback is that the process of writing to storage and retrieving from storage is somewhat slower than the other methods that store the lists in active memory.

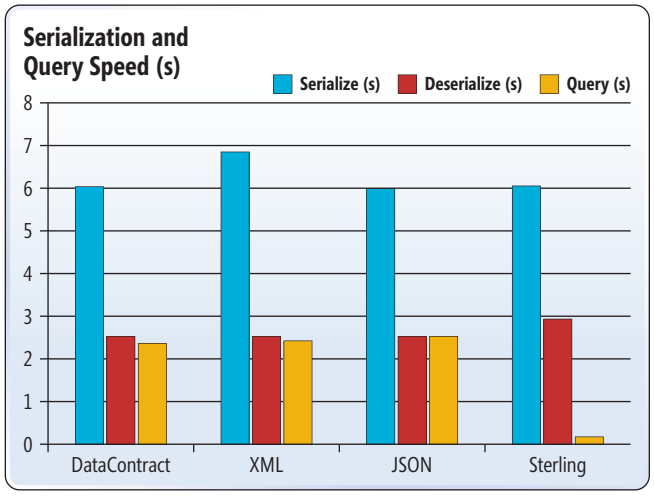


Figure 3 **Comparison of Speed**

Serialization Options

Another benefit of isolated storage is that you can choose multiple serialization strategies. Unlike the settings that allow you to assign a key and an object, the isolated storage mechanism provides a file stream that you can write to using text, XML, JSON or even binary code. This allows for easy and straightforward serialization of many types of objects, including "deep" serialization that can handle a complex object graph and serialize the children and grandchildren and so on of the instance you're working with.

JSON and XML These are the two main serialization strategies available in Silverlight. XML is available using either the DataContractSerializer (which emits XML) or the XMLSerializer. The first is familiar to developers who use Windows Communication Foundation (WCF) and is what the framework uses to hydrate and dehydrate messages sent via Web services. It requires that data is marked using the DataContract and DataMember attributes. This is essentially an "opt-in" approach because you explicitly flag the fields you want serialized. The XmlSerializer serializes all public properties with getters and setters and you can change the behavior by using special XML-specific attributes. For more information on the DataContractSerializer, see bit.ly/fUDPha. For more information on the XmlSerializer, see bit.ly/fCla6q.

JSON stands for JavaScript Object Notation and is popular on the Web because it's a format that's easily translated into JavaScript objects. Some developers prefer this method because it provides readable text in the serialized object in a more compact form than XML. JSON serialization is achieved using a special version of the data contract serializer called the DataContractJsonSerializer. The output it produces takes up significantly less space on disk than XML. For more information on the DataContractJsonSerializer, see bit.ly/8cFyjV.

Binary Silverlight and Windows Phone 7 can also use binary serialization. There's no BinaryFormatter available in Silverlight (this is the class that helps automatically serialize objects to binary) so you must handle the serialization yourself. To use binary serialization, you simply create a binary writer and write to the stream. The writer handles many primitives, so most of the time you can write out the properties on your class to serialize the instance, then create an instance and populate the properties using the reader. For projects with a lot of classes, however, this can become quite cumbersome. This is where Sterling enters the picture.

Sterling Sterling was designed specifically to ease the pain of serializing and deserializing objects in Silverlight. Originally created for Silverlight 4 in the browser, Sterling evolved to a nearly identical codebase that supports Windows Phone 7. Under the covers, Sterling uses binary serialization, which results in a very compact format on disk. Sterling can serialize almost any class and organizes instances using keys that you provide (any property on the instance may be designated as the key). Sterling also provides indexes that can be queried in memory for speed before loading the entire instance from disk. Sterling gives you full control over the underlying serialization stream; you can encrypt, compress or even override the binary stream to serialize types exactly how you want.

> Another benefit of isolated storage is that you can choose multiple serialization strategies.

The advantage Sterling provides is a quick and easy way to serialize objects along with the ability to query keys and indexes with lightning speed using LINQ to Objects. It handles foreign keys and relationships well. All of these benefits come at only a slight speed cost when serializing and deserializing.

Figure 4 **Edit Recipe Screen**

Figure 1 compares various serialization strategies and the relative size on disk.

To generate these figures, a collection of 2,000 contacts was created using random names and addresses. The contact records each contain a full name, address and a unique ID (see **Figure 2**).

These were then saved using the various serializers, including Sterling. The size computed is the total size on disk, including the additional files that Sterling requires to keep track of indexes and keys.

Full serialization to and from disk is only slightly slower for Sterling due to the overhead of walking the object graph and handling indexes and keys. The chart in **Figure 3** compares how quickly each option can save and load all 2,000 contacts.

The final query statistics show where Sterling provides the most leverage in scanning the collection for contacts with names that begin with an "L" and then loading those full contacts. In the example run, the query produced 65 contacts

out of the 2,000. Sterling filtered and loaded them in just 110 milliseconds compared to more than 2 seconds for the other formats.

The Recipe Application

To demonstrate the use of Sterling on Windows Phone 7, I've written a small recipe application that allows you to browse, edit and add new recipes. Each recipe consists of a name, a set of instructions, a category (such as lunch or dinner) and a set of ingredients. Ingredients can have an amount, a measure and a food item. Food can be added on the fly. **Figure 4** shows a sample page from the application.

For this application, Sterling serves three distinct purposes. First, it helps front-load the reference data used by the application, such as categories, measurements, initial foods and sample recipes. Second, it preserves the data users enter when they add their own custom recipes and foods. Finally, it facilitates tombstoning by serializing the state of the application when it's deactivated. The sample application uses the MVVM pattern and demonstrates how to tombstone from within a ViewModel.

The first step, of course, is to add Sterling to the project. This can be done using NuGet (just search for Sterling) or by downloading the binaries and full source from CodePlex at sterling.codeplex.com.

Setting up the Database

The next step is to set up the database. The database configuration defines what types are going to be persisted and what keys and indexes will be used. The Sterling database is a class created by deriving from BaseDatabaseInstance. There are two overloads you must provide: a unique name for the database (the name is unique per application—you can host multiple databases to segregate data or manage different versions) and a list of table definitions. The base class provides helper methods to define tables, keys and indexes.

Keys and Indexes This application defines a table of type FoodModel using the object Id as the key with an index over the FoodName property. This creates an in-memory list of objects that can be quickly queried and filtered.

The following code defines the food "table" with an integer key and a string index:

```
CreateTableDefinition<FoodModel, int>(f => f.Id)
.WithIndex<FoodModel, string, int>(IDX_FOOD_NAME, f=>f.FoodName)
```

The call to create the table definition takes the class type and the key type and is passed a lambda expression used to resolve the key. You can use any unique non-null value on the class as your key. The index extension method requires the table type, the index type and the key type. Here, a constant defines the index name and the lambda expression provides the value that the index will use.

Identity Using Triggers Sterling supports any type of key, so there's no built-in facility to automatically generate new keys. Instead, Sterling allows you to specify how you would like keys to be generated by using triggers. Triggers are registered with the Sterling database and are called before a save, after a save and before a delete. The call before a save allows you to inspect the instance and generate a key if one doesn't already exist.

In the sample application in the code download accompanying this article, all entities use an integer key. Therefore, it's possible to create a generic trigger based on the instance type to generate the

Figure 5 **Base Trigger for Auto-Key Generation**

```
public class IdentityTrigger<T> : BaseSterlingTrigger<T,int>
  where T: class, IBaseModel, new()
{
  private static int _idx = 1;

  public IdentityTrigger(ISterlingDatabaseInstance database)
  {
    // If a record exists, set it to the highest value plus 1
    if (database.Query<T,int>().Any())
    {
      _idx = database.Query<T, int>().Max(key => key.Key) + 1;
    }
  }

  public override bool BeforeSave(T instance)
  {
    if (instance.Id < 1)
    {
      instance.Id = _idx++;
    }

    return true;
  }

  public override void AfterSave(T instance)
  {
    return;
  }

  public override bool BeforeDelete(int key)
  {
    return true;
  }
}
```

Figure 6 **TypeSerializer**

```
public class TypeSerializer : BaseSerializer
{
  /// <summary>
  ///     Return true if this serializer can handle the object,
  ///     that is, if it can be cast to type
    /// </summary>
  /// <param name="targetType">The target</param>
  /// <returns>True if it can be serialized</returns>
  public override bool CanSerialize(Type targetType)
  {
    return typeof (Type).IsAssignableFrom(targetType);
  }

  /// <summary>
  ///     Serialize the object
  /// </summary>
  /// <param name="target">The target</param>
  /// <param name="writer">The writer</param>
  public override void Serialize(object target,
    BinaryWriter writer)
  {
    var type = target as Type;
    if (type == null)
    {
      throw new SterlingSerializerException(
        this, target.GetType());
    }
    writer.Write(type.AssemblyQualifiedName);
  }

  /// <summary>
  ///     Deserialize the object
  /// </summary>
  /// <param name="type">The type of the object</param>
  /// <param name="reader">A reader to deserialize from</param>
  /// <returns>The deserialized object</returns>
  public override object Deserialize(
    Type type, BinaryReader reader)
  {
    return Type.GetType(reader.ReadString());
  }
}
```

Figure 7 **Seeding Data**

```
if (database.Query<CategoryModel, int>().Any()) return;

// Get rid of old data
database.Truncate(typeof(MeasureModel));
database.Truncate(typeof(FoodModel));

var idx = 0;

foreach(var measure in ParseFromResource(FILE_MEASURES,
  line =>
  new MeasureModel
  { Id = ++idx, Abbreviation = line[0], FullMeasure = line[1]} ))
{
  database.Save(measure);
}

// Sample foods auto-generate the id
foreach (var food in
  ParseFromResource(FILE_FOOD, line
    => new FoodModel { FoodName = line[0] })
    .Where(food => !string.IsNullOrEmpty(food.FoodName)))
{
  database.Save(food);
}

var idx1 = 0;

foreach (var category in ParseFromResource(FILE_CATEGORIES,
  line =>
  new CategoryModel { Id = ++idx1, CategoryName = line[0] }))
{
  database.Save(category);
}
```

key. When the trigger is first instantiated, it queries the database for existing keys and finds the highest one. If no records exist, it will set the starting key to 1. Each time an instance is saved that has a key less than or equal to zero, it will assign the next number and increment the key. The code for this base trigger is in **Figure 5**.

Notice that the query can use standard LINQ expressions such as Any and Max. Also, the developer is responsible for providing thread safety within the trigger mechanism.

Using a trigger is easy: You simply register the trigger with the database and pass an instance (this allows you to pass any constructor parameters that are necessary). You can use a similar call to unregister a trigger.

Custom Serializer Sterling supports a variety of types out of the box. When classes and structs are encountered, the public properties and fields of those classes are iterated to serialize the content. Sub-classes and structs are recursively iterated as well. Sterling can't serialize some basic types directly. For example, System.Type is defined as an abstract class that has many possible derived classes. Sterling can't directly serialize or deserialize this type. To support tombstoning, a special class will be created that stores ViewModel properties and uses the type of the ViewModel as the key. In order to handle the type, Sterling lets you create a custom serializer.

To create a custom serializer, derive from the BaseSerializer class and handle the overloads. For the custom TypeSerializer class, any class that derives from System.Type is supported, and the serialization simply writes out the assembly qualified type name. Deserialization uses the static GetType method on the Type class to return the type from the assembly qualified name. The result is shown in **Figure 6**. Note that it explicitly supports any type that derives from (is assignable to) System.Type.

Any custom serializers are registered with the Sterling engine before it's activated.

Seeding and Saving Data

Once the database is defined, it's common to provide "seed data." In the sample application, a list of categories, standard measures and foods are provided to help the user get started—an example recipe is also included. There are several ways to embed the data, but the easiest is to include the data as a resource in the XAP file. The data can then be parsed as a resource stream the first time the application is run and stored in the database.

In order to work with the tombstone process, the Sterling database engine is activated when the application itself is activated, and deactivated when it's tombstoned or exited. This ensures the database keys and indexes are flushed to disk and the database is in a stable state when used again. In the App.xaml.cs file, these events can be connected to the phone lifecycle. To set up the database, only a few lines of code are needed, as shown here:

```
_engine = new SterlingEngine();
_engine.SterlingDatabase.RegisterSerializer<TypeSerializer>();
_engine.Activate();
Database =
  _engine.SterlingDatabase.RegisterDatabase<RecipeDatabase>();
```

The preceeding code snippet demonstrates the steps that create the engine, register the custom serializer and then activate the engine and prepare the database for use. The following code shows how to shut down the engine and database when the application is tombstoned or exited:

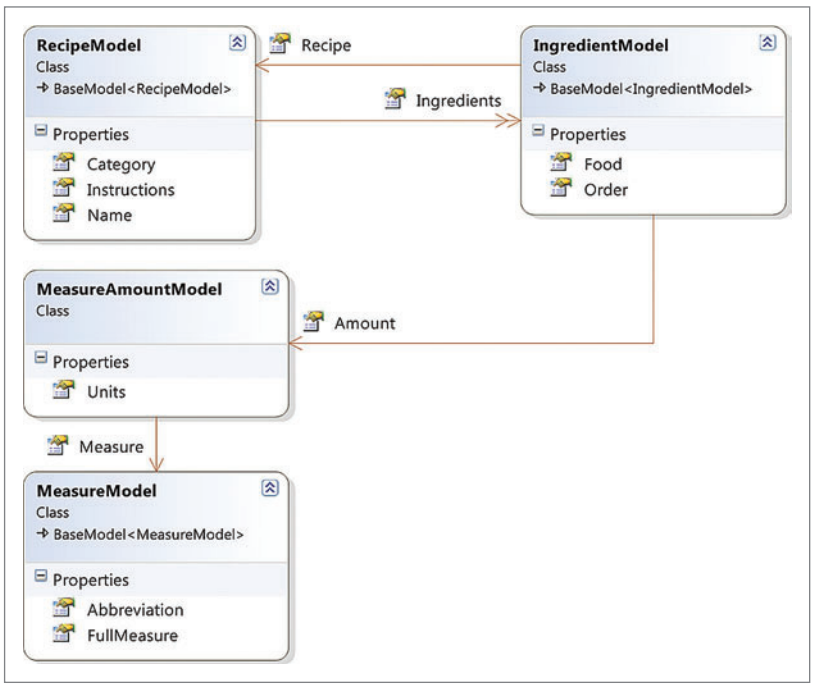


Figure 8 **Recipe Class Hierarchy**

```
Database.Flush();
_engine.Dispose();
Database = null;
_engine = null;
```

Once the database has been activated, it's ready to receive data. The most common way to package data is to include it as an embedded resource in a readable format, such as XML, JSON or CSV. A query to the database can determine whether data already exists, and if it doesn't, the data is loaded. Saving data in Sterling is straightforward: You simply pass the instance to save, and Sterling handles the rest. **Figure 7** shows a query that checks for categories. If no categories exist, the data is read from embedded resource files to seed the database. Note the use of the truncate operation to clear the tables first.

The Main ViewModel: Categories and Recipes

With the seeded database parsed and loaded, the rest of the application can use the existing data to provide lists and prompts for the user, as well as save any information entered by the user. Here's an example of how the seeded data is made available to the application. The following code snippet loads all categories into an observable collection on the main ViewModel:

```
Categories = new ObservableCollection<CategoryModel>();
foreach(var category in App.Database.Query<CategoryModel,int>())
{
  Categories.Add(category.LazyValue.Value);
}
```

The Categories collection can now be bound directly to a Pivot control. A common use of the Pivot control is to provide filtered views over large data sets. The categories indicate the type of meal (whether it's a breakfast meal, lunch meal or other category) and the Pivot displays the relevant recipes when a category is selected. The recipes for each category are exposed by a query that filters based on the currently selected category.

The following XAML snippet shows how the control is directly bound to the collection and selected category:

```
<controls:Pivot
  x:Name="pivotMain"
  Title="Sterling Recipes"
  ItemsSource="{Binding Categories}"
  SelectedItem="{Binding CurrentCategory,Mode=TwoWay}">
```

Editing Ingredients: Foreign Keys

The key to recipes is, of course, their ingredients. The application contains a "master list" of food items and measurements. Each recipe can then have a list of "ingredients" that include the amount, type of measurement and food item. The ingredients button (back in **Figure 4**) takes the user to an ingredients list, where the user can add, delete or edit existing ingredients.

This functionality is possible due to an important feature of Sterling: the support for navigation properties that act like foreign keys. **Figure 8** shows the hierarchy of models used in the application.

Recipes contain lists of ingredients that have a circular reference back to the parent recipe. Ingredients also contain an "amount" model that includes units and measure.

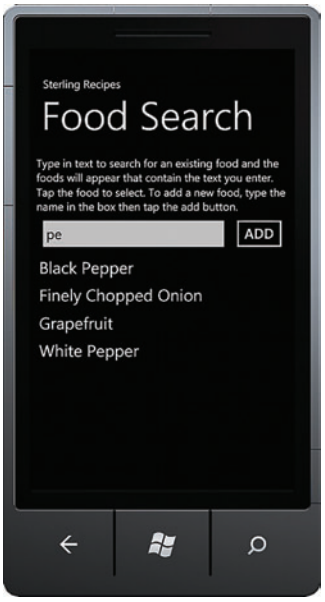


Figure 9 **Food Search for Items that Include the Letters "pe"**

When you save a recipe, Sterling will automatically recognize each ingredient as a separate entry in a separate table definition. Instead of serializing the ingredient with the recipe object, Sterling will serialize the key for the index and then save the ingredient separately. It will also recognize the circular reference with the recipe and stop recursion on the object graph. Including the recipe with the ingredient allows queries directly on the ingredient that can then load the corresponding recipe.

When you modify or add an ingredient, the save operation will automatically save the related tables as well. When loaded, foreign tables are always pulled from disk, ensuring they're always in sync with the latest version.

Food Search: Querying with Indexes

Sterling uses in-memory keys and indexes to facilitate queries and filters. The food search provides an example of filtering and querying. The text box is bound to the ViewModel and will update the text being typed as the user enters it. Users will see the results (foods that contain the text they're typing in the name) instantly as they're typing. This makes it easy for the user to narrow the search and either select an existing food or enter a new one. You can see the food search page in **Figure 9**, with selected items that contain the letters "pe."

> Sterling was designed specifically to ease the pain of serializing and deserializing objects in Silverlight.

Whenever the user types, the property setter on the ViewModel is updated with the search text. This setter in turn raises the property changed event for the food list. The food list executes a new query against the food database and returns the results using LINQ to Objects. **Figure 10** shows the query, which uses an index to filter

Figure 10 **Food Query**

```
public IEnumerable<FoodModel> Food
{
  get
  {
    if (string.IsNullOrEmpty(_foodText))
    {
      return Enumerable.Empty<FoodModel>();
    }
    var foodTextLower = _foodText.ToLower();
    return from f in App.Database.Query<FoodModel,
      string, int>(RecipeDatabase.IDX_FOOD_NAME)
      where f.Index.ToLower().Contains(foodTextLower)
      orderby f.Index
      select new FoodModel { Id = f.Key, FoodName = f.Index };
  }
}
```

Figure 11 **Tombstoning**

```
/// <summary>
///     Tombstone
/// </summary>
public void Deactivate()
{
  var tombstone = new TombstoneModel
    {SyncType = typeof (ITextEditorViewModel)};
  tombstone.State.Add(ExtractPropertyName(()=>Title), Title);
  tombstone.State.Add(ExtractPropertyName(() =>Text), Text);
  App.Database.Save(tombstone);
}

/// <summary>
///     Returned from tombstone
/// </summary>
public void Activate()
{
  var tombstone = App.Database.Load<TombstoneModel>
    (typeof(ITextEditorViewModel));
  if (tombstone == null) return;
  Title = tombstone.TryGet(ExtractPropertyName(() =>
    Title), string.Empty);
  Text = tombstone.TryGet(ExtractPropertyName(() =>
    Text), string.Empty);
}
```

and order the data. To access the query, the database is called with the class type, the index type and the key type, and then is passed the name of the index. Notice that a new food model is created based on the key and index value returned from the index.

Practical Tombstoning

For tombstoning to work, the current state of the application must be saved and restored when the user returns to the application. Sometimes this requirement can be complex because the user may also navigate backward through the application after it has returned from the tombstoned state. Therefore, all pages must be able to retain their values for the experience to remain seamless.

The MVVM pattern takes the responsibility for tombstoning out of the view-centric XAML and codebehind because the state of the view is synchronized with the ViewModel through data binding. Therefore, each ViewModel can be responsible for saving and restoring its own state. The bindings will update the view accordingly. To facilitate tombstoning, a class called TombstoneModel was created.

It's keyed based on a type (which will be the interface for the ViewModel being saved) and contains a dictionary of keys and objects. This provides the ultimate flexibility for storing types or classes as needed to preserve the ViewModel state.

Sterling supports this because regardless of how a property is defined—whether as a generic object, an interface or an abstract base class—the serialization will write out the object as the implemented type. This isn't possible with the built-in serializers. To provide a common interface, an ITombstoneFriendly interface is provided by the lightweight MVVM framework. This interface defines activate and deactivate methods that are called when the bound view is navigated to and from (tombstoning will trigger a "from" navigation, for example).

Tombstoning then becomes as simple as creating the model, setting the type and then setting the values that must be persisted. Returning from the tombstone event involves loading the model, reading the values and restoring the state on the ViewModel. **Figure 11** illustrates these steps in the text editor ViewModel that must preserve the title that was passed in and the text the user has entered.

When the view is closed via normal means (not tombstoning) the record can easily be deleted. When the application is closed, the table is truncated to remove all records, because the state shouldn't be preserved when the application is relaunched.

Lightweight and Flexible

As you can see, Sterling removes the burden of complex serialization strategies from the developer. To persist entities is as simple as defining a class type and a lambda expression that returns a unique key. Lightweight (less than a 100KB DLL as of this writing) and flexible (with hooks for triggers, encryption, compression and custom serialization), it should address most needs related to local, embedded databases, caching and tombstoning. The recipe application demonstrates how Sterling integrates with Windows Phone 7 to easily and effectively address these needs. ■

Jeremy Likness *is a senior consultant and project manager at Wintellect LLC in Atlanta. He's a Microsoft Silverlight MVP and a certified MCTS Silverlight developer. Likness frequently presents topics related to Silverlight line-of-business applications at conferences and users groups. He also regularly updates his Silverlight blog at csharperimage.jeremylikness.com.*

Thanks to the following technical experts for reviewing this article: *Rob Cameron, John Garland and Jeff Prosise*

JAMES MCCAFFREY

Curved Lines for Bing Maps AJAX

In this month's column I present a JavaScript function that can be used to add curved lines to a Bing Maps AJAX map control and discuss the principles used to test the function.

The best way for you to see where I'm headed is to check out the image in **Figure 1**. It shows a Bing Maps AJAX map control of the Denver area in the west-central United States. The custom JavaScript function that's the topic of this article is being used to draw the blue curved line from the city of Boulder—the location of the University of Colorado (which one of my daughters attends)—to the Denver International Airport (where I've been in and out of a lot!). The custom function is named AddBezierCurve. In addition to being a highly useful function if you ever work with Bing Maps, the AddBezierCurve routine provides a great showcase for demonstrating API testing principles and contains some interesting math that you may find useful in other programming scenarios.

In the sections that follow, I'll first give you a brief explanation of creating a Bezier curve, which is the underlying math technique used by AddBezierCurve. Next, I'll walk you through the function line by line so that you'll be able to modify the source code to suit your own needs, if necessary. I'll finish up by describing the general principles and specific techniques you can use to test AddBezierCurve and similar JavaScript functions. This article assumes you have a basic familiarity with JavaScript but doesn't assume you've programmed with the Bing Maps AJAX control library before. I think you'll find the topics presented here interesting and useful additions to your developer and tester skill sets.

Bezier Curves

Bezier curves can be used to draw a curved line between two points. Although Bezier curves have many variations, the simplest form is called a quadratic Bezier curve and requires two endpoints, usually called P0 and P2, and an intermediate point, P1, which determines the shape of the curve. Take a look at the example in the XY graph in **Figure 2**.

Points P0, P1 and P2 are the open red circles on the graph. The two endpoints are P0 = (1,2) and P2 = (13,8). The intermediate point is P1 = (7,10). The Bezier function accepts a parameter, usually called t, which ranges from 0.0 to 1.0. Each value of t produces a point between P0 and P2 that lies on a curve.

In **Figure 2** I used five values for t: 0.0, 0.25, 0.50, 0.75 and 1.00. This produced the five points shown as smaller dots on the graph. I'll present the equations used when I go over the code for the AddBezierCurve function. You can see that the first Bezier point for t = 0.0 is (1,2) = P0, and that the last Bezier point for t = 1.0 is (13,8) = P2. Values of t = 0.0 and t = 1.0 will generate points P0 and P2 in general. If we connect the Bezier points with line segments, we'll approximate a nice curve between P0 and P2. More values of t generate more points, which create a smoother curve approximation.

> Bezier curves can be used to draw a curved line between two points.

Given endpoints P0 and P2, the value of P1 determines the shape of the resulting Bezier curve. In the example in **Figure 2**, I arbitrarily picked a point that's halfway between P0 and P2 on the horizontal axis and slightly above the highest point (P2) on the vertical axis. If I had skewed P1 to the left a bit, the curve would shift to the left and become more symmetric. If I had increased the height of P1, the curve would have been taller and more peaked.

Calling the AddBezierCurve Function

In **Figure 1** you can see that my map is on a Web page named CurveDemo.html (available in the code download). The overall structure of CurveDemo.html is presented in **Figure 3**.

After the HTML <title> tag, I use the <script> element to gain programmatic access to version 6.3 of the Bing Maps AJAX map control library. Notice I'm not using good coding practices, such as including a DOCTYPE declaration, in order to keep the size of my demo code small. At the time I'm writing this article, the current version of the library is version 7. It has improved performance, but it has a completely different code base from earlier versions. Unless you're using AddBezierCurve with an existing Bing Maps AJAX API set, you should probably use version 7. You should be able to refactor the code I present here for Bing Maps AJAX version 7 with little trouble.

I declare a global VEMap object named map and instantiate it to null. Notice that because JavaScript doesn't use explicit type declarations, without a comment it's not obvious that map has type VEMap.

The AddBezierCurve function accepts up to eight parameter values. The first four parameters (start, finish, arcHeight and skew) are required. The next four parameters (color, width, upDown and numSegments)

Code download available at code.msdn.microsoft.com/mag201106TestRun.

are optional and have default values. The start and finish parameters are type VELatLong (latitude and longitude) objects and represent the endpoints P0 and P2 described in the previous section. The arcHeight parameter represents the height of intermediate shape-defining point P1. The skew parameter represents the left-right adjustment of point P1. The color parameter is the VEColor of the curve. The width is a numeric value that's the width of the curve. The upDown parameter is a string that can be "up" or "down" and specifies whether the curve should bend up or down. The numSegments parameter specifies the number of values to use for the Bezier t value, which in turn determines how many line segments make up the curve.

The MakeMap function instantiates the VEMap object using the new keyword and sets the control ID to "myMap" so that the Web browser can use the AJAX response to know where on CurveDemo.html to place the map control. I use the LoadMap method to initially position the map centered just north of Denver—with a zoom level of 10—using the Road view. Next I set the start parameter to the latitude and longitude of Boulder and set the finish parameter to an area just north of the Denver International Airport. I set arcHeight to 0.20. As we'll see in the next section, the arcHeight is interpreted as degrees latitude above the midpoint between P0 (start) and P2 (finish). I set skew to -0.004 to slightly move the hump of the curve 0.004 degrees longitude to the left. I set the color to blue with no alpha transparency, the width of the curved line to 6 and the number of line segments to 200, and then call the AddBezierCurve function with an "up" direction.

In the HTML body tag I use the onload event to call MakeMap, which in turn calls AddBezierCurve.

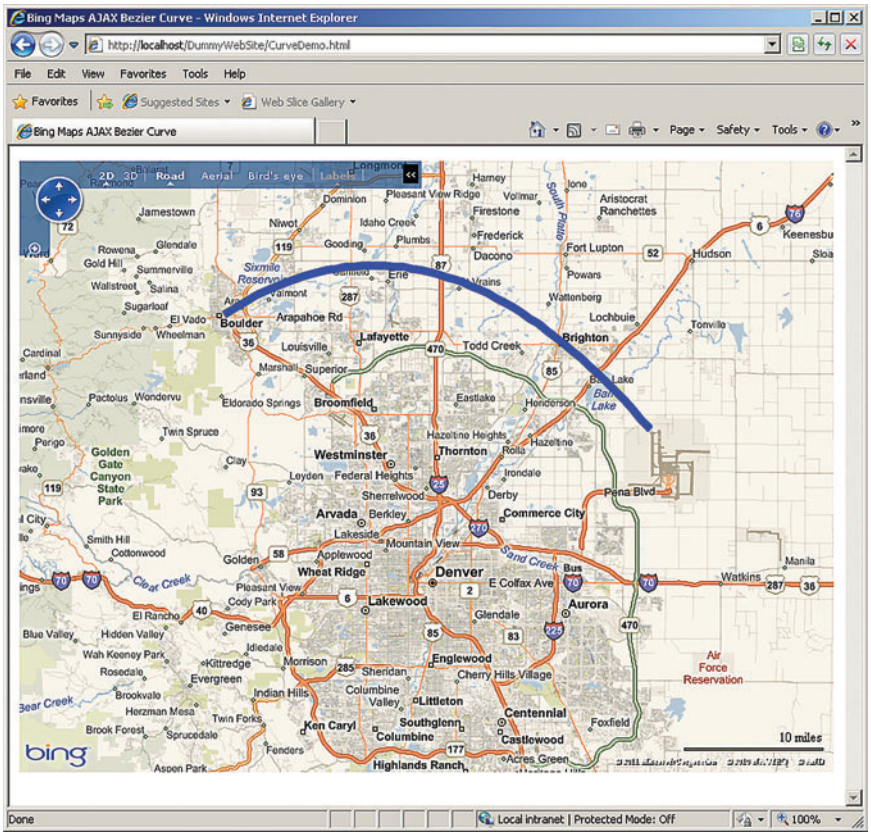


Figure 1 **The Custom Function AddBezierCurve Adds a Curved Line to Bing Maps**

Defining the AddBezierCurve Function

The AddBezierCurve function begins with:

```
function AddBezierCurve(start, finish, arcHeight, skew, color, width,
 upDown, numSegments)
{
  // Preconditions and parameter descriptions here
  if (typeof color == 'undefined') { var color = new VEColor(255,0,0,1.0); }
  if (typeof width == 'undefined') { var width = 2; }
  if (typeof upDown == 'undefined') { var upDown = 'up'; }
  if (typeof numSegments == 'undefined') { var numSegments = 10; }
  ...
```

To save space, I removed the comments that describe the function's assumed preconditions (for example, the existence of an instantiated global VEMap object named map) and descriptions of the eight input parameters. The function code begins by defining default parameters. I use the JavaScript typeof operator to determine if parameters color, width, upDown and numSegments are present or not. The typeof operator returns string "undefined," rather than null, if a variable isn't present. If the color parameter is absent, I create a local-scope VEColor object named color and instantiate it to red (the parameters to VEColor are red, green, blue and transparency). Using the same approach, I create default values for width (2), upDown ("up") and numSegments (10).

The function continues:

```
if (start.Longitude > finish.Longitude) {
 var temp = start;
 start = finish;
 finish = temp;
}

if (numSegments < 2)
 numSegments = 2;
...
```

> Bezier curves are really interesting, and there's a lot of information about them available on the Internet.

I normalize the start and finish VELatLong parameters so the start point is to the left of the finish point. Technically this isn't necessary, but it helps keep the code easier to understand. I perform an error check on the numSegments parameter to ensure that there are at least two line segments for the resulting curve.

Next, I compute the coordinates of a point that lies midway between P0 (start) and P2 (finish) on the line connecting P0 and P2:

```
var midLat = (finish.Latitude + start.Latitude) / 2.0;
var midLon = (finish.Longitude + start.Longitude) / 2.0;
...
```

This point will serve as the starting point to construct the intermediate P1 point using the arcHeight and skew parameters.

Next, I determine P1:

```
if (Math.abs(start.Longitude - finish.Longitude) < 0.0001) {
 if (upDown == 'up')
   midLon -= arcHeight;
 else
   midLon += arcHeight;
 midLat += skew;
}
else { // 'normal' case, not vertical
 if (upDown == 'up')
   midLat += arcHeight;
 else
   midLat -= arcHeight;
 midLon += skew;
}
...
```

I'll explain the first part of the code logic in a moment. The second part of the branching logic is the usual case where the line connecting start and finish isn't vertical. In this case I check the value of the upDown parameter and if it's "up" I add the arcHeight value to the latitude (up-down) value of the midpoint base reference and then add the skew value to the longitude (left-right) of the base reference. If the upDown parameter isn't "up," I assume upDown is "down" and subtract arcHeight from the up-down latitude component of the midpoint base reference. Notice that instead of using an explicit upDown parameter, I could eliminate upDown altogether and just infer that positive values of arcHeight mean up and negative values of arcHeight mean down.

Testing the AddBezierCurve function isn't trivial.

The first part of the branching logic deals with vertical or nearly vertical lines. Here I effectively swap the roles of the arcHeight up-down value and skew the left-right value. Notice that there's no leftRight parameter for the skew; positive values for skew mean to-the-right and negative values mean to-the-left.

With everything ready, I enter the Bezier curve generation algorithm:

```
var tDelta = 1.0 / numSegments;

var lons = new Array(); // 'x' values
for (t = 0.0; t <= 1.0; t += tDelta) {
var firstTerm = (1.0 - t) * (1.0 - t) * start.Longitude;
 var secondTerm = 2.0 * (1.0 - t) * t * midLon;
 var thirdTerm = t * t * finish.Longitude;
 var B = firstTerm + secondTerm + thirdTerm;
 lons.push(B);
}
...
```

First I compute the difference between t values. Recall from the previous section that t ranges from 0.0 to 1.0 inclusive. So if numSegments = 3, then tDelta would be 0.33 and my t values would be 0.00, 0.33, 0.67 and 1.00, yielding four Bezier points and three line segments. Next, I create a new array named lons to hold the x values that are longitudes. A detailed explanation of the Bezier equations is outside the scope of this article; however, notice that there are three terms that depend on the value of t, P0 (start), P1 (midLon) and P2 (finish). Bezier curves are really interesting, and there's a lot of information about them available on the Internet.

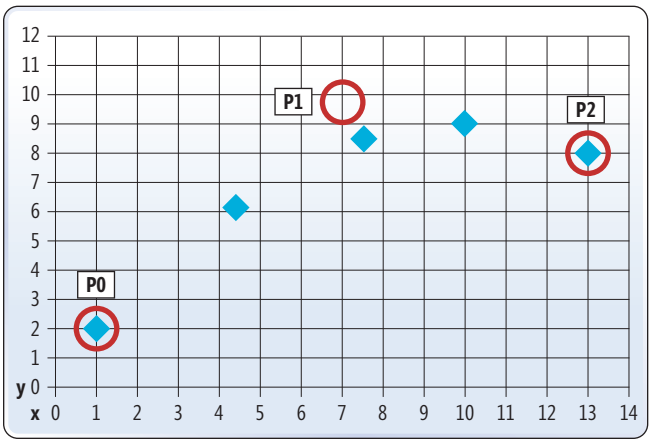


Figure 2 **Constructing Bezier Curves**

Next, I use the same Bezier equations to compute the y (latitude) values into an array named lats:

```
var lats = new Array(); // 'y' values
for (t = 0.0; t <= 1.0; t += tDelta) {
  var firstTerm = (1.0 - t) * (1.0 - t) * start.Latitude;
  var secondTerm = 2.0 * (1.0 - t) * t * midLat;
  var thirdTerm = t * t * finish.Latitude;
  var B = firstTerm + secondTerm + thirdTerm;
  lats.push(B);
}
...
```

Now I finish up the AddBezierCurve function:

```
 var points = new Array();
 for (i = 0; i < lats.length; ++i) {
  points.push(new VELatLong(lats[i], lons[i]));
 }

 var curve = new VEShape(VEShapeType.Polyline, points);
 curve.HideIcon();
 curve.SetLineColor(color);
 curve.SetLineWidth(width);
 map.AddShape(curve);
}
```

I create an array of VELatLong objects named points and add the latitude-longitude pairs from the lats and lons arrays to the points array. Then I instantiate a VEShape of type Polyline, hide the pesky default icon, set the color and width of the Polyline and use the AddShape method to place the Bezier curve onto the global VEMap object named map.

API functions are often well-suited for automated testing using random input.

Testing the AddBezierCurve Function

Testing the AddBezierCurve function isn't trivial. The function has object parameters (start, finish and color), numeric parameters (arcHeight, skew, width and numSegments) and a string parameter (upDown). In fact, some of my colleagues use similar

Figure 3 **The Demo Web Page Structure**

```
<html>
<!-- CurveDemo.html -->
<head>
<title>Bing Maps AJAX Bezier Curve</title>
<script type="text/javascript"
 src="http://ecn.dev.virtualearth.net/mapcontrol/mapcontrol.ashx?v=6.3">
</script>
<script type="text/javascript">

 var map = null; // global VEMap object

 function AddBezierCurve(start, finish, arcHeight, skew, color, width,
  upDown, numSegments)
 {
   // Code here
 }

 function MakeMap()
 {
  map = new VEMap('myMap');
  map.LoadMap(new VELatLong(39.8600, -105.0000), 10, VEMapStyle.Road);
  var start = new VELatLong(40.0200, -105.2700); // Boulder
  var finish = new VELatLong(39.9000, -104.7000); // airport
  var arcHeight = 0.20;
  var skew = -0.004;
  var color = new VEColor(0,0,255,1.0); // blue
  var width = 6;
  var numSegments = 200;

  AddBezierCurve(start, finish, arcHeight, skew, color, width, 'up', numSegments);
 }

</script>
</head>
<body onload="MakeMap();">
<div id='myMap' style="position:relative; width:800px; height:600px;"></div>
</body>
</html>
```

functions as the basis for job interview questions for software-testing positions. This form of testing is often called API testing or module testing. The first thing to check is basic functionality, or, in other words: Does the function do what it's supposed to do in more or less normal situations? Next, a good tester would start looking at the function parameters and determine what would happen in the case of bad or edge-case inputs.

The AddBezierCurve function doesn't perform an initial check on the start and finish VELatLong parameter values. If either or both are null or undefined, the map will render but no curved line will appear. Similarly, the function doesn't check for illegal values of start or finish. VELatLong objects use the World Geodetic System 1984 (WGS 84) coordinate system, where legal latitudes are in the range [-90.0, +90.0] and legal longitudes are in the range [-180.0, +180.0]. Illegal values can cause AddBezierCurve to produce unexpected and incorrect results. Another bad input possibility for the start and finish parameter values are objects of the wrong type—a VEColor object, for example.

Experienced testers would also test numeric parameters arcHeight and skew in a similar way. Interesting boundary condition values for these parameters include 0.0, -1.0 +1.0, and 1.7976931348623157e+308 (the JavaScript maximum number on many systems). A thorough tester would explore the effects of using string or object values for arcHeight and skew.

Testing the color parameter would be similar to the start and finish parameters. However, you would also want to test the default value by omitting the color parameter in the function call. Note that because JavaScript passes parameters by position, if you omitted the color parameter value you should also omit all subsequent parameter values (width, upDown and numSegments). That said, omitting color but then supplying values for one or more of the trailing parameters would have the effect of misaligning parameter values and is something an experienced tester would examine.

Because the width and numSegments parameters represent physical measurements, good testers would certainly try values of 0 and negative values for width and numSegments. Because these parameter values are implied as integers, you'd want to try non-integer numeric values such as 3.5, too.

> If you ever work with maps, you should find the AddBezierCurve function presented here a very useful resource.

If you examine the code for AddBezierCurve, you'll notice that if the value of the upDown parameter is anything other than "up," the function logic interprets the parameter value as "down." This would lead an experienced tester to wonder about the correct behavior of null, empty string and a string consisting of a single space. Additionally, numeric and object values for upDown should be tried. An experienced tester might ask the developer if he had intended the upDown parameter to be case sensitive as it is now; a value of "UP" for upDown would be interpreted as "down."

API functions are often well-suited for automated testing using random input. You could use one of several techniques to programmatically generate random values for start, finish, arcHeight and skew, send those values to the Web page and see if any exceptions are thrown.

Dual Purposes

Wrapping up, this column has dual purposes. The first is to present a practical JavaScript function to draw a Bezier curve on a Bing Maps AJAX map control along with the underlying code logic. If you ever work with maps, you should find the AddBezierCurve function presented here a very useful resource. The second purpose of the column is to present guidelines for testing a nontrivial JavaScript function. We've seen that there are a number of things you should check, including null or missing values, illegal values, boundary values and values with incorrect type. These principles are true for API/module testing functions written in most programming languages. ■

Dr. James McCaffrey *works for Volt Information Sciences Inc., where he manages technical training for software engineers working at the Microsoft Redmond, Wash., campus. He's worked on several Microsoft products, including Internet Explorer and MSN Search. Dr. McCaffrey is the author of ".NET Test Automation Recipes" (Apress, 2006), and can be reached at jammc@microsoft.com.*

Thanks to the following technical experts for reviewing this article: *Paul Koch, Dan Liebling, Anne Loomis Thompson and Shane Williams*

TED NEWARD

Multiparadigmatic .NET, Part 8: Dynamic Programming

In last month's article, we finished off the third of the three metaprogramming facilities supported by the Microsoft .NET Framework languages, that of parametric polymorphism (generics), and talked about how it provided variability in both structural and behavioral manners. In as far as it goes, parametric metaprogramming provides some powerful solutions. But it's not the be-all, end-all answer to every design problem—no single programming paradigm is.

As an example, consider the Money<> class that served as the last example and testbed (see **Figure 1**). Recall, from last time, that the main reason we use the currency as a type parameter is to avoid accidental compiler-permitted conversion of euros to dollars without going through an official conversion rate to do so.

As was pointed out last time, being able to do that conversion is important, though, and that's what the Convert<> routine is intended to do—give us the ability to convert dollars to euros, or to pesos, or to Canadian dollars, or whatever other currency we might need or want to convert to.

But that means some kind of currency-conversion code, which is obviously missing from the implementation in **Figure 1**—right now, we just do a 1-1 conversion, simply changing over the Currency property over to the new C2 currency type, and that's just not going to cut it.

My Money, Your Money, It's All Funny Money

Fixing this means we need some kind of conversion routine to do the calculation of one to the other, and that can take on a lot of different solutions. One approach might be to leverage the inheritance axis again and make USD and EUR into ICurrency types with routines designed to do that exact conversion. Doing so begins with the definition of an ICurrency type and marking USD and EUR as implementors of that interface, as shown in **Figure 2**.

This strategy works great, so far. In fact, the additional type constraint on the type parameter in Money<> is a useful enhancement to make sure we can't have Money<string> or Money<Button>. It looks unusual, but this trick—known as the "marker interface" idiom in Java—serves an interesting and important purpose.

In Java, prior to Java 5 getting its equivalent of custom attributes, we used this trick to put static declarations on types. Just as the .NET Framework uses [Serializable] to indicate that a class can be serialized into a stream of bytes, Java classes implemented (inherited) from the Serializable interface, which had no members.

Much as we'd love to use custom attributes to mark USD and EUR as [Currency], type constraints can't key off of custom attributes, and having that type constraint on C is an important enhancement, so we resort to the marker interface. It's a bit unusual, but if you think about interfaces as a way of making declarative statements about what this type is, rather than just about what it can do, it makes sense.

(While we're at it, we'll add constructors to make it a bit easier to instantiate Money<>.)

But trying to declare a currency conversion in ICurrency runs into an immediate snag: ICurrency has no knowledge of any subtype (concrete currency type), thus we can't really declare a method here that takes Money<USD> and converts it to Money<EUR> through some kind of automatically adjusting conversion calculation. (Some kind of Internet-based lookup or Web service would be the actual implementation here, but for now, let's assume static ratios.) But even if we could, trying to write said methods would be extremely tricky, because we'd need to dispatch based on two types (the currency we're converting from and the currency we're converting to), along with the single parameter (the amount we're converting).

Given that we like keeping the currency as a type, it means that we might take a first stab at writing this method like so:

```
interface ICurrency {
  float Convert<C1, C2>(float from);
}
```

Then, it might seem like we could write something like this as a way of specializing the Convert method in derived types:

```
class USD : ICurrency {
  public float Convert<USD, EUR>(float from) {
    return (from * 1.2f); }
  public float Convert<EUR, USD>(float from) {
    return (from / 1.2f); }
}
```

Figure 1 **Money< >**

```
class USD { }
class EUR { }
class Money<C> {
  public float Quantity { get; set; }
  public C Currency { get; set; }

  public static Money<C> operator
    +(Money<C> lhs, Money<C> rhs) {
    return new Money<C>() {
      Quantity = lhs.Quantity + rhs.Quantity,
      Currency = lhs.Currency };
  }

  public Money<C2> Convert<C2>() where C2 : new() {
    return new Money<C2>() { Quantity = this.Quantity,
      Currency = new C2() };
  }
}
```

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

New in Team 2.11

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

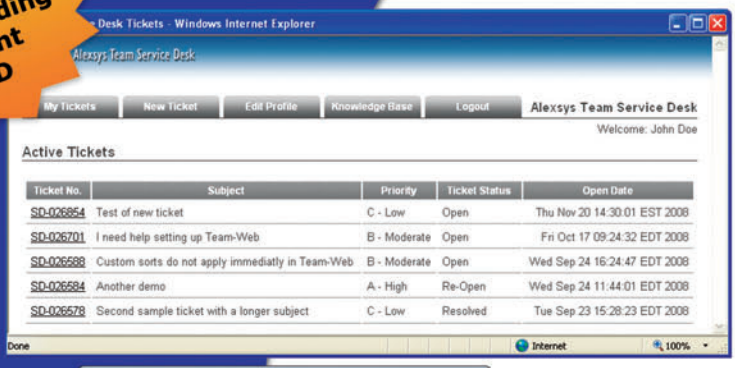
Service Desk Features

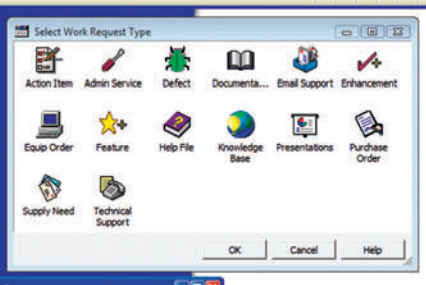
- Fully Secure
- Unlimited Users Self Registered or Active Directory
- Integrated into Your Web Site
- Fast/AJAX Dynamic Content
- Unlimited Service Desks
- Visual Service Desk Builder

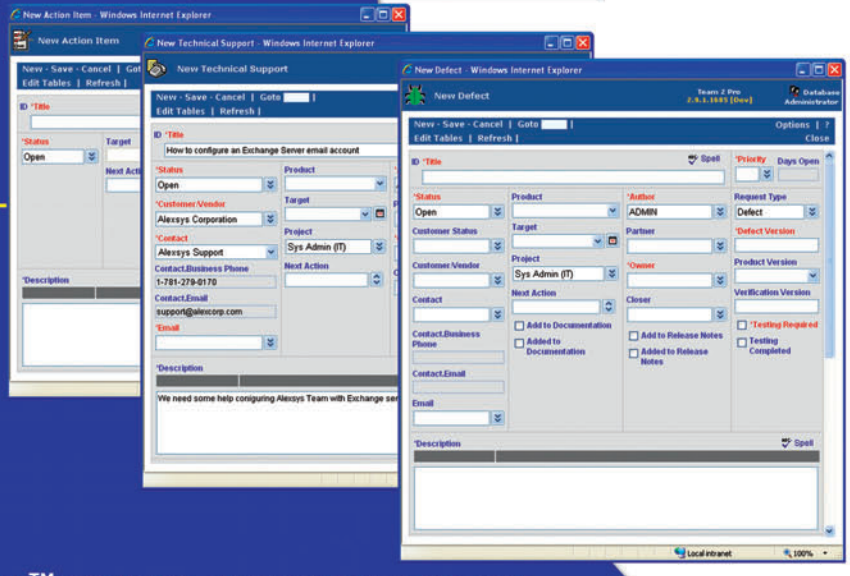
Team 2 Features

- Windows and Web Clients
- Multiple Work Request Forms
- Customizable Database
- Point and Click Workflows
- Role Based Security
- Clear Text Database
- Project Trees
- Time Recording
- Notifications and Escalations
- Outlook Integration

ZDNet Editors' Pick







Free Trial and Single User FreePack™ available at www.alexcorp.com.
FreePack™ includes a free single user Team Pro and Team-Web license.
Need more help? Give us a call at 1-888-880-ALEX (2539).

Team 2 works with its own standard database, while Team Pro works with Microsoft SQL, MySQL, and Oracle Servers.
Team 2 works with Windows 7/2008/2003/Vista/XP.
Team-Web works with Internet Explorer, Firefox, Netscape, Safari, and Chrome.

Figure 2 **ICurrency**

```
interface ICurrency { }
class USD : ICurrency { }
class EUR : ICurrency { }
class Money<C> where C : ICurrency {
  public float Quantity { get; set; }
  public C Currency { get; set; }

  public static Money<C> operator+(Money<C> lhs,
    Money<C> rhs) {

    return new Money<C>() {
      Quantity = lhs.Quantity + rhs.Quantity,
      Currency = lhs.Currency };
  }

  public Money<C2> Convert<C2>() where C2 : new() {
    return new Money<C2>() { Quantity = this.Quantity,
      Currency = new C2() };
  }
}
```

Alas, this would be wrong. The compiler interprets USD and EUR as type parameters just like C1 and C2.

Next, we might try something like this:

```
class USD : ICurrency {
  public float Convert<C1,C2>(float from) {
    if (C1 is USD && C2 is EUR) {
    }
  }
}
```

But again, the compiler complains: C1 is a "type parameter" but is used like a "variable." In other words, we can't use C1 as if it were a type itself. It's just a placeholder. Yikes—this is going nowhere fast.

One potential solution is to resort to simply passing the types as Reflection-based Type parameters, which creates something like the code shown in **Figure 3**.

And it works, in that the code compiles and runs, but numerous traps lay in wait: the conversion code has to be duplicated between both the USD and EUR classes, and when new currencies are added, such as British pounds (GBP), not only will a new GBP class be needed—as would be expected—but both USD and EUR will also need to be modified to include GBP. This is going to get really messy before long.

What's in a Name?

In traditional object-oriented programming (OOP) languages, developers have been able to dispatch based on a single type by use of virtual methods. The compiler sends the request to the appropriate method implementation depending on the actual type behind the reference on which the method is invoked. (This is the classic ToString scenario, for example.)

In this situation, however, we want to dispatch based on two types (C1 and C2)—what's sometimes called double dispatch. Traditional OOP has no great solution for it other than the Visitor design pattern and, frankly, to many developers that's not a great solution at all. It requires the creation of a single-purpose hierarchy of classes, of sorts. As new types are introduced, methods start exploding all over the hierarchy to accommodate each newcomer.

But taking a step back affords us a chance to look at the problem anew. While the type-safety was necessary to ensure that Money<USD> and Money<EUR> instances couldn't be comingled, we don't really need the types USD and EUR for much beyond their places as type parameters. In other words, it's not their types that we care about for currency-conversion purposes, but simply their names. And their names permit another form of variability, sometimes referred to as name-bound or dynamic programming.

Dynamic Languages vs. Dynamic Programming

At first blush, it may seem like there's an intrinsic relationship between dynamic languages and dynamic programming—and to some degree there is, but only in that dynamic languages take the concept of name-bound execution to its highest degree. Rather than ascertain at compile time whether target methods or classes exist, dynamic languages like Ruby, Python or JavaScript simply assume that they exist and look them up at the last moment possible.

It turns out, of course, that the .NET Framework affords the savvy designer the same kind of flexibility in binding using Reflection. You can create a static class that contains the names of the currencies, then invoke it using Reflection, as shown in **Figure 4**.

Adding in a new currency, such as British pounds, means simply creating the empty GBP class (implementing ICurrency), and adding the necessary conversion routines to Conversions.

Of course, C# 4 (and just about every version of Visual Basic before this) provides built-in facilities to make this easier, assuming we know the name at compile time. C# provides the dynamic type and Visual Basic has had Option Strict Off and Option Explicit Off for decades.

Figure 3 **Using Reflection-Based Type Parameters**

```
interface ICurrency {
  float Convert(Type src, Type dest, float from);
}

class USD : ICurrency {
  public float Convert(Type src, Type dest, float from) {
    if (src.Name == "USD" && dest.Name == "EUR")
      return from / 1.2f;
    else if (src.Name == "EUR" && dest.Name == "USD")
      return from * 1.2f;
    else
      throw new Exception("Illegal currency conversion");
  }
}

class EUR : ICurrency {
  public float Convert(Type src, Type dest, float from) {
    if (src.Name == "USD" && dest.Name == "EUR")
      return from / 1.2f;
    else if (src.Name == "EUR" && dest.Name == "USD")
      return from * 1.2f;
    else
      throw new Exception("Illegal currency conversion");
  }
}

class Money<C> where C : ICurrency, new() {
  public Money() { Currency = new C(); }
  public Money(float amt) : this() { Quantity = amt; }

  public float Quantity { get; set; }
  public C Currency { get; set; }

  public static Money<C> operator +(Money<C> lhs, Money<C> rhs) {
    return new Money<C>(lhs.Quantity + rhs.Quantity);
  }

  public Money<C2> Convert<C2>() where C2 : ICurrency, new() {
    return new Money<C2>(
      Currency.Convert(typeof(C), typeof(C2), this.Quantity));
  }
}
```

Figure 4 **Dynamic Binding with Reflection**

```
static class Conversions {
  public static Money<EUR> USDToEUR(Money<USD> usd) {
    return new Money<EUR>(usd.Quantity * 1.2f);
  }

  public static Money<USD> EURToUSD(Money<EUR> eur) {
    return new Money<USD>(eur.Quantity / 1.2f);
  }
}

class Money<C> where C : ICurrency, new() {
  public Money() { Currency = new C(); }
  public Money(float amt) : this() { Quantity = amt; }

  public float Quantity { get; set; }
  public C Currency { get; set; }

  public static Money<C> operator +(Money<C> lhs, Money<C> rhs) {
    return new Money<C>(lhs.Quantity + rhs.Quantity);
  }

  public Money<C2> Convert<C2>() where C2 : ICurrency, new() {
    MethodBase converter = typeof(Conversions).GetMethod(
      typeof(C).Name + "To" + typeof(C2).Name);
    return (Money<C2>)converter.Invoke(null, new object[] { this });
  }
}
```

In fact, as Apple Objective-C shows, dynamic programming isn't necessarily limited to interpreted languages. Objective-C is a compiled language that uses dynamic programming all over the place in its frameworks, particularly for event-handling binding. Clients that wish to receive events simply provide the event-handling method, named correctly. When the sender wants to inform the client of something interesting, it looks up the method by name and invokes the method if it's present. (For those who remember back that far, this is also exactly how Smalltalk works.)

Of course, name-bound resolution has its faults, too, most of which come up in error-handling. What should happen when a method or class that you expect to be present isn't? Some languages (such as Smalltalk and the Apple implementation of Objective-C) hold that simply nothing should happen. Others (Ruby, for example) suggest that an error or exception should be thrown.

Much of the right answer will depend on the domain itself. In the Money<> example, if it's reasonable to expect that certain currencies cannot be converted, then a missing conversion routine should trigger some kind of message to the user. If all currencies within the system should be convertible, though, then obviously there's some kind of developer mistake here and it should be caught during unit tests. In fact, it's a fair statement to suggest that no dynamic programming solution should ever be released to an unsuspecting public without a significant set of unit tests successfully passing first.

Creating Commonality

Name-bound variability represents a powerful mechanism for commonality/variability analysis, but dynamic programming hardly ends there. Using the full-fidelity metadata capabilities present within the CLR, it becomes possible to start looking at creating commonality through other criteria beyond name: method return types, method parameter types and so on. In fact, it could be argued that attributive metaprogramming is really just an offshoot of dynamic programming based on custom attributes. What's more, name-bound variability doesn't have to be tied to the entire name. Early builds of the NUnit unit testing framework assumed a test method was any method that began with the characters "test."

In my next column, we'll examine the last of the paradigms in common .NET Framework languages: that of functional programming, and how it provides yet another way to view commonality/variability analysis, which happens to be almost diametrically opposite to that viewed by traditional object-ists. ■

TED NEWARD *is a Principal with Neward & Associates, an independent firm specializing in enterprise .NET Framework and Java platform systems. He has written more than 100 articles, is a C# MVP, INETA speaker, and has authored and coauthored a dozen books, including "Professional F# 2.0" (Wrox, 2010). He consults and mentors regularly—reach him at ted@tedneward.com, or read his blog at blogs.tedneward.com.*

THANKS to the following technical expert for reviewing this article: *Mircea Trofin*

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

OCTOBER 17–21
REDMOND, WASHINGTON

WE'LL BE HERE. COME JOIN US!

GET READY FOR FIVE ACTION-PACKED DAYS ON THE MICROSOFT CAMPUS!

You'll learn from the Microsoft development team and industry experts during in-depth sessions on:

- **VISUAL STUDIO 2010 / .NET**
- **DATA MANAGEMENT**
  Windows, RIA, Web and Cloud computing applications.
- **MOBILE DEVELOPMENT**
  Apps for Windows Phone 7 (WP7), Android and iOS (iPhone/iPad/iPod Touch) devices.
- **DEVELOPING SERVICES**
  WCF (Windows Communication Foundation), REST, SOAP, OData, Windows Server AppFabric, and other key technologies.
- **CLOUD COMPUTING**
- **PROGRAMMING PRACTICES**
  Methodologies such as SCRUM and Kanban; and tools such as Visual Studio testing and Team Foundation Server (TFS).
- **LIGHTSWITCH**
- **WEB / HTML 5**
  Web tools and technologies such as ASP.NET MVC 3, WebMatrix, and Razor.
- **SILVERLIGHT / WPF**

Microsoft Visual Studio · Microsoft .NET · Microsoft Visual Studio LightSwitch · Windows Azure · Windows Phone · Microsoft Silverlight

And on top of all this great education, you'll have EXCLUSIVE access to check out Microsoft's corporate headquarters!

REGISTER NOW AT
WWW.VSLIVE.COM/REDMOND

SAVE $300 WHEN YOU SIGN UP
BEFORE AUGUST 10 USE CODE ADJUN

PLATINUM SPONSORS:

VERSANT

esri

GOLD SPONSOR:

DevExpress™
Download • Compare • Decide!

MEDIA SPONSOR:

THE CODE PROJECT
WWW.CODEPROJECT.COM

SUPPORTED BY:

Microsoft

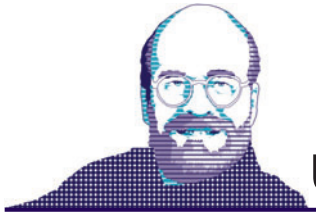
msdn

Microsoft® Visual Studio®

Visual Studio MAGAZINE

PRODUCED BY:

1105 MEDIA INC



CHARLES PETZOLD

Principles of Pagination

For a couple decades now, programmers specializing in computer graphics have known that the most difficult tasks do not involve bitmaps or vector graphics, but good old text.

Text is tough for several reasons, most of which relate to the fact that it's usually desirable for the text to be readable. Also, the actual height of a piece of text is only casually related to its font size, and character widths vary depending on the character. Characters are often combined into words (which must be held together). Words are combined into paragraphs, which must be separated into multiple lines. Paragraphs are combined into documents, which must be either scrolled or separated into multiple pages.

In the last issue I discussed the printing support in Silverlight 4, and now I'd like to discuss printing text. The most important difference between displaying text on the screen and printing text on the printer can be summed up in a simple truth suitable for framing: The printer page has no scrollbars.

A program that needs to print more text than can fit on a page must separate the text into multiple pages. This is a non-trivial programming task known as pagination. I find it quite interesting that pagination has actually become more important in recent years, even as printing has become less important. Here's another simple truth you can frame and put up on your wall: pagination—it's not just for printers anymore.

Pick up any e-book reader—or any small device that lets you read periodicals, books or even desktop book-reading software—and you'll find documents that have been organized into pages. Sometimes these pages are preformatted and fixed (as with PDF and XPS file formats), but in many cases pages can by dynamically reflowed (such as with EPUB or proprietary e-book formats). For documents that can be reflowed, something as simple as changing the font size can require a whole section of the document to be repaginated dynamically while the user is waiting, probably impatiently.

Paginating on the fly—and doing it quickly—turns a non-trivial programming job into one that can be exceptionally challenging. But let's not frighten ourselves too much. I'll build up to the hard stuff over time, and for now will start off very simply.

Stacking TextBlocks

Silverlight provides several classes that display text:

- The Glyphs element is perhaps the one class with which most Silverlight programmers are least familiar. The font used in Glyphs must be specified either with a URL or a Stream object, which makes the element most useful in fixed-page documents or document packages that rely heavily on specific fonts. I won't be discussing the Glyphs element.
- The Paragraph class is new with Silverlight 4 and mimics a prominent class in the document support of the Windows Presentation Foundation (WPF). But Paragraph is used mostly in conjunction with RichTextBox, and it's not supported in Silverlight for Windows Phone 7.
- And then there's TextBlock, which is often used in a simple way by setting the Text property—but it can also combine text of different formats with its Inlines property. TextBlock also has the crucial ability to wrap text into multiple lines when the text exceeds the allowable width.

TextBlock has the virtue of being familiar to Silverlight programmers and suitable for our needs, so that's what I'll be using.

The SimplestPagination project (available with the downloadable code for this article) is designed to print plain-text documents. The program treats each line of text as a paragraph that might need to be wrapped into multiple lines. However, the program implicitly assumes that these paragraphs are not very long. This assumption comes from the limitation of the program to break paragraphs across pages. (That's the Simplest part of the SimplestPagination name.) If a paragraph is too long to fit on a page, the whole paragraph is moved to the next page, and if the paragraph is too large for a single page, then it's truncated.

You can run the SimplestPagination program at bit.ly/elqWgU. It has just two buttons: Load and Print. The Load button displays an OpenFileDialog that lets you pick a file from local storage. Print paginates it and prints it.

The OpenFileDialog returns an object of FileInfo. The OpenText method of FileInfo returns a StreamReader, which has a ReadLine method for reading whole lines of text. **Figure 1** shows the PrintPage handler.

As usual, the printed page is a visual tree. The root of this particular visual tree is the Border element, which is given a Padding property to obtain 48-unit (half-inch) margins as indicated in the desiredMargins field. The PageMargins property of the event arguments provides the dimensions of the unprintable margins of the page, so the Padding property needs to specify additional space to bring the total up to 48.

A StackPanel is then made a child of the Border, and TextBlock elements are added to the StackPanel. After each one, the Measure

Code download available at code.msdn.microsoft.com/mag201106UIFrontiers.

Figure 1 **The PrintPage Handler of SimplestPagination**

```
void OnPrintDocumentPrintPage(
  object sender, PrintPageEventArgs args) {

  Border border = new Border {
    Padding = new Thickness(
      Math.Max(0, desiredMargin.Left - args.PageMargins.Left),
      Math.Max(0, desiredMargin.Top - args.PageMargins.Top),
      Math.Max(0, desiredMargin.Right - args.PageMargins.Right),
      Math.Max(0, desiredMargin.Bottom - args.PageMargins.Bottom))
  };

  StackPanel stkpnl = new StackPanel();
  border.Child = stkpnl;
  string line = leftOverLine;

  while ((leftOverLine != null) ||
    ((line = streamReader.ReadLine()) != null)) {

    leftOverLine = null;

    // Check for blank lines; print them with a space
    if (line.Length == 0)
      line = " ";

    TextBlock txtblk = new TextBlock {
      Text = line,
      TextWrapping = TextWrapping.Wrap
    };

    stkpnl.Children.Add(txtblk);
    border.Measure(new Size(args.PrintableArea.Width,
      Double.PositiveInfinity));

    // Check if the page is now too tall
    if (border.DesiredSize.Height > args.PrintableArea.Height &&
      stkpnl.Children.Count > 1) {

      // If so, remove the TextBlock and save the text for later
      stkpnl.Children.Remove(txtblk);
      leftOverLine = line;
      break;
    }
  }

  if (leftOverLine == null)
    leftOverLine = streamReader.ReadLine();

  args.PageVisual = border;
  args.HasMorePages = leftOverLine != null;
}
```

method of the Border is called with a horizontal constraint of the printable width of the page, and a vertical constraint of infinity. The DesiredSize property then reveals how big the Border needs to be. If the height exceeds the height of the PrintableArea, then the TextBlock must be removed from the StackPanel (but not if it's the only one).

The leftOverLine field stores the text that didn't get printed on the page. I also use it to signal that the document is complete by calling ReadLine on the StreamReader one last time. (Obviously if StreamReader had a PeekLine method, this field wouldn't be required.)

The downloadable code contains a Documents folder with a file named EmmaFirstChapter.txt. This is the first chapter of Jane Austen's novel, "Emma," specially prepared for this program: All the paragraphs are single lines, and they're separated by blank lines. With the default Silverlight font, it's about four pages in length. The printed pages aren't easy to read, but that's only because the lines are too wide for the font size.

This file also reveals a little problem with the program: It could be that one of the blank lines is the first paragraph on a page. If that's the case, it shouldn't be printed. But that's just additional logic.

For printing text that has actual paragraphs, you could use blank lines between paragraphs, or you might prefer to have more control by setting the Margin property of TextBlock. It's also possible to have a first-line indent by changing the statement that assigns the Text property of the TextBlock from this:

```
Text = line,

to this:
Text = "     " + line,
```

But neither of these techniques would work well when printing source code.

Splitting the TextBlock

After experimenting with the SimplestPagination program, you'll probably conclude that its biggest flaw is the inability to break paragraphs across pages.

One approach to fixing this problem is illustrated in the BetterPagination program, which you can run at bit.ly/ekpdZb. This program is much like SimplestPagination except in cases when a TextBlock is added to the StackPanel, which causes the total height to exceed the page. In SimplestPagination, this code simply removed the entire TextBlock from the StackPanel:

```
// Check if the page is now too tall
if (border.DesiredSize.Height > args.PrintableArea.Height &&
  stkpnl.Children.Count > 1) {

  // If so, remove the TextBlock and save the text for later
  stkpnl.Children.Remove(txtblk);
  leftOverLine = line;
  break;
}
```

BetterPagination now calls a method named RemoveText:

```
// Check if the page is now too tall
if (border.DesiredSize.Height > args.PrintableArea.Height) {
  // If so, remove some text and save it for later
  leftOverLine = RemoveText(border, txtblk, args.PrintableArea);
  break;
}
```

RemoveText is shown in **Figure 2**. The method simply removes one word at a time from the end of the Text property of the TextBlock and checks if that helps the TextBlock fit on the page. All the removed text is accumulated in a StringBuilder that the PrintPage handler saves as leftOverLine for the next page.

Figure 2 **The RemoveText Method from BetterPagination**

```
string RemoveText(Border border,
  TextBlock txtblk, Size printableArea) {

  StringBuilder leftOverText = new StringBuilder();

  do {
    int index = txtblk.Text.LastIndexOf(' ');

    if (index == -1)
      index = 0;

    leftOverText.Insert(0, txtblk.Text.Substring(index));
    txtblk.Text = txtblk.Text.Substring(0, index);
    border.Measure(new Size(printableArea.Width,
      Double.PositiveInfinity));

    if (index == 0)
      break;
  }
  while (border.DesiredSize.Height > printableArea.Height);

  return leftOverText.ToString().TrimStart(' ');
}
```

It's not pretty, but it works. Keep in mind that if you're dealing with formatted text (different fonts, font sizes, bold and italic), then you'll be working not with the Text property of the TextBlock but with the Inlines property, and that complicates the process immensely.

And yes, there are definitely faster ways to do this, although they certainly are more complex. For example, a binary algorithm can be implemented: Half the words can be removed, and if it fits on the page, half of what was removed can be restored, and if that doesn't fit on the page, then half of what was restored can be removed, and so forth.

However, keep in mind that this is code written for printing. The bottleneck of printing is the printer itself, so while the code might spend a few more seconds testing each and every TextBlock, it's probably not going to be noticeable.

But you might start wondering exactly how much is going on under the covers when you call Measure on the root element. Certainly all the individual TextBlock elements are getting Measure calls, and they're using Silverlight internals to determine how much space that text string actually occupies with the particular font and font size.

You might wonder whether code like this would even be tolerable for paginating a document for a video display on a slow device.

So let's try it.

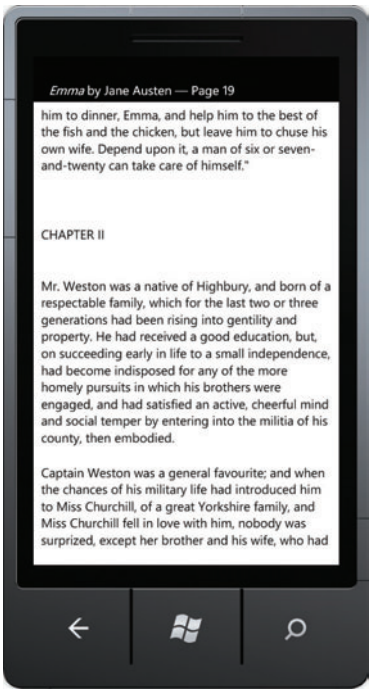


Figure 3 **EmmaReader Running on the Windows Phone 7 Emulator**

Pagination on Windows Phone 7

My goal (which won't be completed in this article) is to build an e-book reader for Windows Phone 7 suitable for reading plain-text book files downloaded from Project Gutenberg (gutenberg.org). As you may know, Project Gutenberg dates from 1971 and was the very first digital library. For many years, it focused on providing public-domain books—very often the classics of English literature—in a plain-text ASCII format. For example, the complete "Emma" by Jane Austen is the file gutenberg.org/files/158/158.txt.

Every book is identified by a positive integer for its file name. As you can see here, "Emma" is 158 and its text version is in file 158.txt. In more recent years Project Gutenberg has provided other formats such as EPUB and HTML, but I'm going to stick with plain text for this project for obvious reasons of simplicity.

The EmmaReader project for Windows Phone 7 includes 158.txt as a resource and lets you read the entire book on the phone. **Figure 3** shows the program running on the Windows Phone 7 emulator. For gesture support, the project requires the Silverlight for Windows Phone Toolkit, downloadable from silverlight.codeplex.com. Tap or flick left to go to the next page; flick right to go to the previous page.

The program has almost no features except those necessary to make it reasonably useable. Obviously I'll be enhancing this program, particularly to allow you to read other books besides "Emma"—perhaps even books of your own choosing! But for nailing down the basics, it's easier to focus on a single book.

If you examine 158.txt, you'll discover the most significant characteristic of plain-text Project Gutenberg files: Each paragraph consists of one or more consecutive 72-character lines delimited by a blank line. To turn this into a format suitable for TextBlock to wrap lines, some pre-processing is required to concatenate these individual consecutive lines into one. This is performed in the PreprocessBook method in EmmaReader. The entire book—including zero-length lines separating paragraphs—is then stored as a field named paragraphs of type List<string>. This version of the program doesn't attempt to divide the book into chapters.

As the book is paginated, every page is identified as an object of type PageInfo with just two integer properties: ParagraphIndex is an index into the paragraphs list and CharacterIndex is an index into the string for that paragraph. These two indices indicate the paragraph and character that begins the page. The two indices for the first page are obviously both zero. As each page is paginated, the indices for the next page are determined.

The program does not attempt to paginate the entire book at once. With the page layout I've defined and the default Silverlight for Windows Phone 7 font, "Emma" sprawls out to 845 pages and requires nine minutes to get there when running on a real device. Obviously the technique I'm using for pagination—requiring Silverlight to perform a Measure pass for each page, and very often many times if a paragraph continues from one page to the next—takes a toll. I'll be exploring some faster techniques in later columns.

But the program doesn't need to paginate the entire book at once. As you start reading at the beginning of a book and proceed page by page, the program only needs to paginate one page at a time.

Features and Needs

I originally thought that this first version of EmmaReader would have no features at all except those necessary to read the book from beginning to end. But that would be cruel. For example, suppose you're reading the book, you've gotten to page 100 or so, and you turn off the screen to put the phone in your pocket. At that time, the program is tombstoned, which means that it's essentially terminated. When you turn the screen back on, the program starts up afresh and you're back at page one. You'd then have to tap 99 pages to continue reading where you left off!

For that reason, the program saves the current page number in isolated storage when the program is tombstoned or terminated. You'll always jump back to the page you left. (If you experiment with this feature when running the program under Visual Studio, either on the emulator or an actual phone, be sure to terminate the program by pressing the Back button, not by stopping debugging in Visual Studio. Stopping debugging doesn't allow the program to terminate correctly and access isolated storage.)

Figure 4 **The Paginate Method in EmmaReader**

```
void Paginate(ref int paragraphIndex, ref int characterIndex) {
  stackPanel.Children.Clear();

  while (paragraphIndex < paragraphs.Count) {
    // Skip if a blank line is the first paragraph on a page
    if (stackPanel.Children.Count == 0 &&
      characterIndex == 0 &&
      paragraphs[paragraphIndex].Length == 0) {
        paragraphIndex++;
        continue;
    }

    TextBlock txtblk = new TextBlock {
      Text =
        paragraphs[paragraphIndex].Substring(characterIndex),
      TextWrapping = TextWrapping.Wrap,
      Foreground = blackBrush
    };

    // Check for a blank line between paragraphs
    if (txtblk.Text.Length == 0)
      txtblk.Text = " ";

    stackPanel.Children.Add(txtblk);
    stackPanel.Measure(new Size(pageHost.ActualWidth,
      Double.PositiveInfinity));

    // Check if the StackPanel fits in the available height
    if (stackPanel.DesiredSize.Height > pageHost.ActualHeight) {
      // Strip words off the end until it fits
      do {
        int index = txtblk.Text.LastIndexOf(' ');

        if (index == -1)
          index = 0;

        txtblk.Text = txtblk.Text.Substring(0, index);
        stackPanel.Measure(new Size(pageHost.ActualWidth,
          Double.PositiveInfinity));

        if (index == 0)
          break;
      }
      while (stackPanel.DesiredSize.Height > pageHost.ActualHeight);

      characterIndex += txtblk.Text.Length;

      // Skip over the space
      if (txtblk.Text.Length > 0)
        characterIndex++;

      break;
    }
    paragraphIndex++;
    characterIndex = 0;
  }

  // Flag the page beyond the last
  if (paragraphIndex == paragraphs.Count)
    paragraphIndex = -1;
}
```

Saving the page number in isolated storage isn't actually enough. If only the page number was saved, then the program would have to paginate the first 99 pages in order to display the hundredth. The program needs at least the PageInfo object for that page.

But that single PageInfo object isn't enough, either. Suppose the program reloads, it uses the PageInfo object to display page 100, and then you decide to flick your finger right to go to the previous page. The program doesn't have the PageInfo object for page 99, so it needs to repaginate the first 98 pages.

For that reason, as you progressively read the book and each page is paginated, the program maintains a list of type List<PageInfo> with all the PageInfo objects that it has determined so far. This entire list is saved to isolated storage. If you experiment with the program's source code—for example, changing the layout, or the font size or replacing the entire book with another—keep in mind that any change that affects pagination will invalidate this list of PageInfo objects. You'll want to delete the program from the phone (or the emulator) by holding your finger on the program name on the start list, and selecting Uninstall. This is currently the only way to erase the stored data from isolated storage.

Here's the content Grid in MainPage.xaml:

```
<Grid x:Name="ContentPanel"
  Grid.Row="1" Background="White">
  <toolkit:GestureService.GestureListener>
  <toolkit:GestureListener
    Tap="OnGestureListenerTap"
    Flick="OnGestureListenerFlick" />
  </toolkit:GestureService.GestureListener>

  <Border Name="pageHost" Margin="12,6">
    <StackPanel Name="stackPanel" />
  </Border>
</Grid>
```

During pagination, the program obtains the ActualWidth and ActualHeight of the Border element and uses that in the same way that the PrintableArea property is used in the printing programs. The TextBlock elements for each paragraph (and the blank lines between the paragraphs) are added to the StackPanel.

The Paginate method is shown in **Figure 4**. As you can see, it's very similar to the methods used in the printing programs except that it's accessing a List of string objects based on paragraphIndex and characterIndex. The method also updates these values for the next page.

As you can see in **Figure 3**, the program displays a page number. But notice that it does not display a number of pages, because this can't be determined until the entire book is paginated. If you're familiar with commercial e-book readers, you're probably aware that the display of page numbers and number of pages is a big issue.

One feature that users find necessary in e-book readers is the ability to change the font or font size. However, from the program's perspective, this has deadly consequences: All the pagination information accumulated so far has to be discarded, and the book needs to be repaginated to the current page, which is not even the same page it was before.

Another nice feature in e-book readers is the ability to navigate to the beginnings of chapters. Separating a book into chapters actually helps the program deal with pagination. Each chapter begins on a new page, so the pages in each chapter can be paginated independently of the other chapters. Jumping to the beginning of a new chapter is trivial. (However, if the user then flicks right to the last page of the previous chapter, the entire previous chapter must be re-paginated!)

You'll probably also agree that this program needs a better page transition. Having the new page just pop into place is unsatisfactory because it doesn't provide adequate feedback that the page has actually turned, or that only one page has turned instead of multiple pages. The program definitely needs some work. Meanwhile, enjoy the novel. ■

CHARLES PETZOLD *is a longtime contributing editor to* MSDN Magazine. *His recent book, "Programming Windows Phone 7" (Microsoft Press, 2010), is available as a free download at bit.ly/cpebookpdf.*

THANKS to the following technical expert for reviewing this article: *Jesse Liberty*

DAVID PLATT

Will Microsoft Learn DEC's Lesson?

Last month, I told you how DEC, once the world's second-largest computer company, spiraled down into death by not recognizing the market changes happening around it. That same, self-inflicted death spiral could happen to Microsoft if it doesn't start recognizing current market trends better than it has.

An old proverb, allegedly Russian, says, "When all you have is a hammer, every problem looks like a nail." To DEC, everything looked like a VAX. To Microsoft, everything looks like a PC.

When Microsoft first entered the palmtop market a decade ago, what did it call its product? Yep, a Pocket PC. Its proudest features were Pocket Word and Pocket Excel. I still have one that I use as a paperweight. My cat Simba sometimes knocks it off my desktop just for fun.

Why do customers buy palmtop devices? Instant access—it's in your pocket when you need it. In return, you sacrifice display size and ease of input. A user won't limp along doing spreadsheets on her palmtop device for very long, because it's much harder there than on her PC. She'll use the palmtop for purposes that require instant access, like getting directions when her husband is lost and won't admit it. It's an entirely different beast, not a smaller PC.

> To DEC, everything looked like a VAX. To Microsoft, everything looks like a PC.

Microsoft took a decade to realize how palmtop devices differ from PCs and produce a decent one, the Windows Phone 7. Its voice integration with Bing is superb—just say what you're looking for ("Cape Ann Veterinary"), and it not only finds and displays the site, but also offers to call the business or navigate you to it. Now *that* makes life easier—not tapping in Excel spreadsheet entries with a stylus, one painful character at a time.

But even today, what comes on every Windows Phone 7 device? Office Mobile, with Word, Excel and PowerPoint. Just in case Notepad isn't enough for my grocery list. I'd rather use the space for more music or games or higher-res pictures, but I can't remove Office—Microsoft glued it in place. We're Microsoft, we're a PC company, *Thou shalt carry Office*.

Microsoft is now falling farther behind in the tablet market. It brought out its first tablet PC about a decade ago—an overpriced, underpowered PC with touchscreen support so you could use your finger instead of a mouse. The silence was deafening. Dell offers only two models of Windows tablets today, both notebook PCs with touchscreens slapped on. The HP Slate is also a full Windows 7 PC, minus a keyboard. They're a tiny niche market at best.

Apple sold 15 million iPads in the first nine months, while the Microsoft tablet has flailed around for a decade. Why? Because Apple realizes what Microsoft does not: the true laptop form factor is an entirely different device, not a smaller PC. It requires a completely different design approach, much closer to a phone than to a PC. I'm watching my 8-year-old daughter play with my father's iPad as I write these words. She loves it, far more than her PC at home. I wouldn't write a novel on one, but my daughter prefers it for kaleidoscope art and pony races. "Toys," sneered one current Microsoft employee. That's exactly what Ken Olsen used to say about PCs.

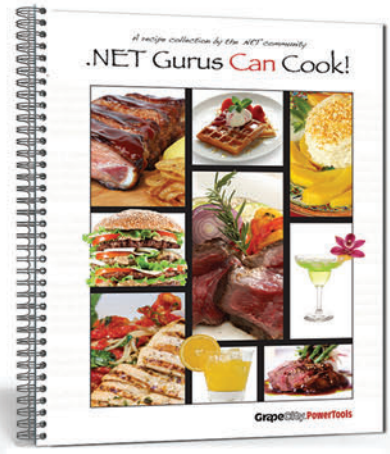
Like French generals building the Maginot line after WWI, Microsoft keeps fighting the last war over again. And like the French, the company will get slaughtered if it doesn't break its self-reinforcing positive feedback loop and start realizing how the world has changed. Parts of Windows Phone 7 show the brilliance to which Microsoft developers can rise when their managers point them at the correct problems. If Microsoft doesn't open its eyes, Apple and Google will do to Microsoft what Microsoft and others did to DEC. If I'm still around to write Bill Gates' obituary, I wonder how much of it I'll be able to do by cutting and pasting from Ken Olsen's. ■

DAVID S. PLATT *teaches Programming .NET at Harvard University Extension School and at companies all over the world. He's the author of 11 programming books, including "Why Software Sucks" (Addison-Wesley Professional, 2006) and "Introducing Microsoft .NET" (Microsoft Press, 2002). Microsoft named him a Software Legend in 2002. He wonders whether he should tape down two of his daughter's fingers so she learns how to count in octal. You can contact him at rollthunder.com.*

.NET Gurus Can Cook!

A recipe collection by the .NET community

You've seen them in **Speak** in KeyNotes. You've read their **Articles**. You've searched their **Sites** and perused their **Magazines**. You've met them at **.NET User's Groups**.



Now see them Cook!

STARRING: HENRY ALLAIN, SCOTT ALLEN, TED BAHR, GLENN BLOCK, RICHARD CAMPBELL, ROB CONERY, HOWARD DIERKING, JIM DUFFY, MARK DUNN, KEITH ELDER, DINO ESPOSITO, MARY JO FOLEY, CARL FRANKLIN, RUSSELL FUSTINO, JON GALLOWAY, GIORGIO GARCIA-AGREDA, GLEN GORDON, PHIL HAACK, JEFF HADFIELD, SCOTT HANSELMAN, JOE HEALY, MATT HINZE, TIM HUCKABY, CHRIS KINSMAN, JULIE LERMAN, DAVID LYMAN, MATT MOROLLO, JASON OLSON, FRITZ ONION, JEFFREY PALERMO, JOHN PAPA, HARRY PIERSON, AYENDE RAHIEN, DAVID RUBENSTEIN, SHERVIN SHAKIBI, STEVE SMITH, DAVE WARD, JOHN WATERS, CHRIS WOODRUFF AND A WHOLE LOT MORE!

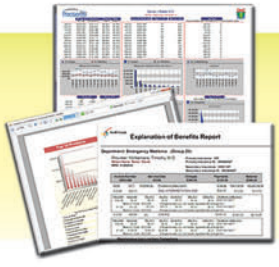
"AWESOME, PURE GENIUS"
★★★★★

Download your cookbook of recipes:
www.GCPowerTools.com/CookBook

Brought to you by:

ACTIVEREPORTS

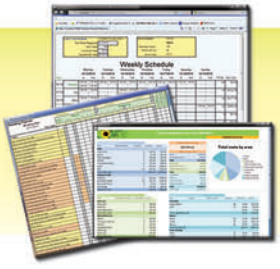
The de facto standard reporting tool for Microsoft Visual Studio



- ✓ Fast and flexible reporting engine
- ✓ Flexible event-driven API to completely control the rendering of reports
- ✓ Wide range of export and preview formats including Windows Forms viewer, Web viewer, Adobe Flash and PDF
- ✓ Royalty-free licensing for Web and Windows applications

SPREAD

The award-winning Microsoft Excel compatible spreadsheet component for Microsoft Visual Studio



- ✓ Programmable, embedded spreadsheet platform
- ✓ Full import/export support for Excel documents
- ✓ Built-in support for Excel functions
- ✓ Flexible printing and PDF export options
- ✓ Build-in chart with hundreds of chart styles
- ✓ Royalty-free licensing

Download your free trials today and see what you've been missing!

start building smarter applications

GrapeCity PowerTools

www.GCPowerTools.com

Dundas Dashboard

The Dashboard Framework for Developers like you

V2.5 Now Available

Dundas Dashboard was built with developers and IT staff in mind. Whether it's our open API, simple web integration or powerful scripting capabilities, technologists have all the tools and options they need for getting their dashboard projects up and running quickly and easily.

Powered by Microsoft® Silverlight™

Dundas Data Visualization

www.dundas.com

(416) 467-5100 • (800) 463-1492