

msdn[®] magazine

CONCURRENCY

Simplify Asynchronous Programming with Tasks

Igor Ostrovsky 24

Throttling Concurrency in the CLR 4.0 ThreadPool

Erika Fuentes 32

Actor-Based Programming with the Asynchronous Agents Library

Michael Chu and Krishnan Varadarajan 38

PLUS:

Migrate Your ASP.NET 1.1 Apps to Visual Studio 2010

Jonathan Waldman 50

Create a Silverlight 4 Web Part for SharePoint 2010

Paul Stubbs 64

Making MapPoint 2010 and SQL Server Spatial Work Together

Eric Frost and Richard Marsden 70

COLUMNS

EDITOR'S NOTE

U.S. Schools Not Getting It Done
Keith Ward page 4

CUTTING EDGE

Better Web Forms with the MVP Pattern
Dino Esposito page 6

GOING PLACES

IronRuby on Windows Phone 7
Shay Friedman page 16

TEST RUN

Request-Response Testing Using IronPython
James McCaffrey page 79

SECURITY BRIEFS

The MSF-Agile+SDL Process Template for TFS 2010
Bryan Sullivan page 84

UI FRONTIERS

Touch and Response
Charles Petzold page 92

DON'T GET ME STARTED

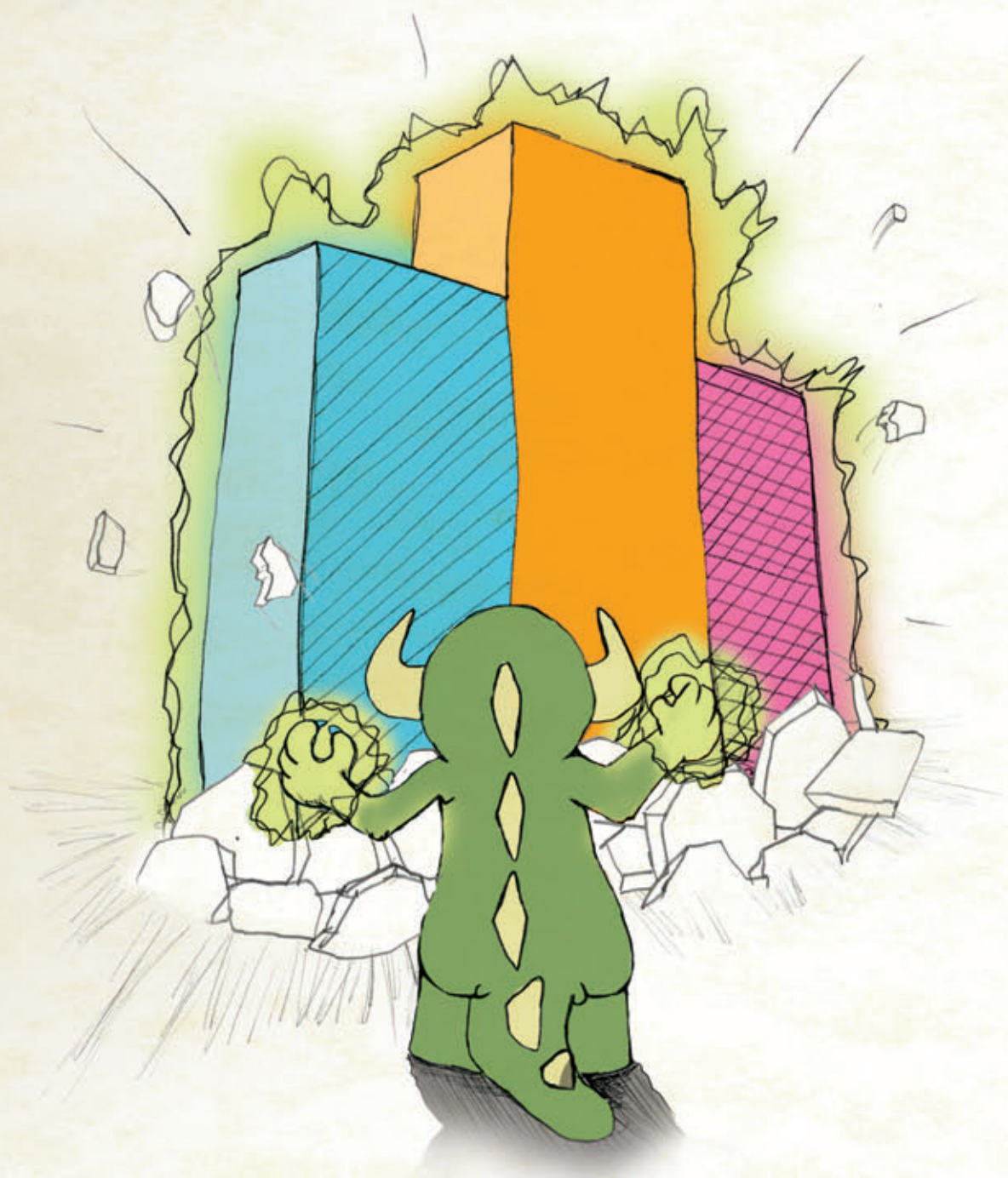
Weasel Words
David Platt page 96



This month at
msdn.microsoft.com/magazine:

THE WORKING PROGRAMMER

Multiparadigmatic .NET, Part 1
Ted Neward



BREAK OUT OF THE BOX

Sure, Visual Studio 2010 has a lot of great functionality—we're excited that it's only making our User Interface components even better! We're here to help you go beyond what Visual Studio 2010 gives you so you can create Killer Apps quickly, easily and without breaking a sweat! Go to infragistics.com/beyondthebox today to expand your toolbox with the fastest, best-performing and most powerful UI controls available. You'll be surprised by your own strength!

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



dtSearch®

Instantly Search Terabytes of Text



- ◆ 25+ full-text and fielded data search options

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

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

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

- ◆ Fully-functional evaluations available

Content extraction only licenses also available

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

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

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

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

1-800-IT-FINDS • www.dtSearch.com

The Smart Choice for Text Retrieval® since 1991



SEPTEMBER 2010 VOLUME 25 NUMBER 9

LUCINDA ROWLEY Director

DIEGO DAGUM Editorial Director/mmeditor@microsoft.com

KERI GRASSL Site Manager

KEITH WARD Editor in Chief/mmeditor@microsoft.com

TERRENCE DORSEY Technical Editor

DAVID RAMEL Features Editor

WENDY GONCHAR Managing Editor

MARTI LONGWORTH Associate Managing Editor

SCOTT SHULTZ Creative Director

JOSHUA GOULD Art Director

ALAN TAO Senior Graphic Designer

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

Redmond Media Group

Henry Allain President, Redmond Media Group

Matt Morollo Vice President, Publishing

Doug Barney Vice President, Editorial Director

Michele Imgrund Director, Marketing

Tracy Cook Online Marketing Director

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

Matt Morollo VP Publishing

Chris Kourtoglou Regional Sales Manager

William Smith National Accounts Director

Danna Vedder Microsoft Account Manager

Jenny Hernandez-Asandas Director Print Production

Serena Barnes Production Coordinator/msdnadproduction@1105media.com

1105 MEDIA

Neal Vitale President & Chief Executive Officer

Richard Vitale Senior Vice President & Chief Financial Officer

Michael J. Valenti Executive Vice President

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

Christopher M. Coates Vice President, Finance & Administration

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

Carmel McDonagh Vice President, Attendee Marketing

David F. Myers Vice President, Event Operations

Jeffrey S. Klein Chairman of the Board

MSDN Magazine (ISSN 1528-4859) is published monthly by 1105 Media, Inc., 9201 Oakdale Avenue, Ste. 101, Chatsworth, CA 91311. 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, PO. Box 3167, Carol Stream, IL 60132, email MSDNmag@1105service.com or call (847) 763-9560. POSTMASTER: Send address changes to MSDN Magazine, PO. Box 2166, Skokie, IL 60076. Canada Publications Mail Agreement No: 40612608. Return Undeliverable Canadian Addresses to Circulation Dept. or IMS/NJ. Attn: Returns, 310 Paterson Plank Road, Carlstadt, NJ 07072.

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

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

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

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

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

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

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

Microsoft



Printed in the USA

Your best source for software development tools!

programmer's paradise®



LEADTOOLS Recognition SDK by LEAD Technologies

Develop desktop and server document imaging and ECM applications that require high-speed multi-threaded forms recognition and processing, OCR, ICR, OMR, and barcode technology.

- Supports text, OMR, image, and 1D/2D barcode fields
- Recognize machine print and constrained handwritten text
- Auto-registration and clean-up to improve recognition results
- Latin character set support is included. Arabic and Asian support is available

Paradise #
I05 26301A01

\$3,214.99

programmers.com/LEAD



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

by Pragma Systems

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

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

Certified for Windows 7/2008R2

Paradise #
P35 04201A01

\$550.99

programmers.com/pragma



ActiveReports 6 by GrapeCity

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

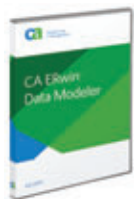
- Fast and Flexible reporting engine
- Data Visualization and Layout Controls such as Chart, Barcode and Table Cross Section Controls
- Wide range of Export and Preview formats including Windows Forms Viewer, Web Viewer, Adobe Flash and PDF
- Royalty-Free Licensing for Web and Windows applications

NEW VERSION 6!

Professional Ed.
Paradise #
D03 04301A01

\$1,310.99

programmers.com/grapacity



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

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

Paradise #
P26 04201E01

\$3,931.99

programmers.com/ca

Multi-Edit⁺

by Multi Edit Software

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

Pre-Order Your Copy and Save!

NEW RELEASE!



1-49 Users
Paradise #
A30Z10101A01

\$223.20

programmers.com/multiedit

InstallShield Professional for Windows

by Flexera Software

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

FLEXERA SOFTWARE™
InstallShield®



Upd from any
Active IS Pro +
IS Pro Silver Mtn
Paradise #
I21 02401B01

\$1,399.00

programmers.com/flexera

Quick View Plus 11

View, Copy and Print Virtually Any File

by Avantstar

With support for over 300 different file formats, Quick View Plus 11 Standard Edition lets you view nearly all the files and e-mail attachments you need instantly without purchasing numerous software programs. Quick View Plus Standard Edition maintains the formatting of the files you view. See and print files as they were originally created and meant to be seen, complete with fonts, text formatting and graphics.

Compatible Operating Systems:
Windows 7 (32-bit version), Vista (32-bit version), XP and 2000



NEW RELEASE!

Standard Edition
Qty 51-99 Users
Paradise #
AV5 01351A01

CALL

programmers.com/avantstar

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 85101A02

\$446.99

programmers.com/vSphere

BUILD ON VMWARE ESXi AND VSPHERE

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

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

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



TX Text Control 15.1

Word Processing Components

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

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

NEW RELEASE!



Professional Edition
Paradise #
T79 02101A02

\$1,220.99

Download a demo today.

programmers.com/theimagingsource

STOP OVERBUYING SOFTWARE TODAY!

Eliminate Wasteful Software
License Spend:

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

**FREE 30-DAY
PROOF OF CONCEPT**

Learn more:
programmers.com/eliminate-wasteful-license-spend



866-719-1528

Prices subject to change. Not responsible for typographical errors.

programmersparadise.com



U.S. Schools Not Getting It Done

In a recent column, I asked for your feedback on the issue of whether U.S. schools are properly preparing students for real-world software development, and the topic touched a nerve, as you came through in spades. What follows is a representative sampling of responses. I've withheld information about some writers, at their request.

Brian Fulford, Vice President of Information Technology, Database Solutions Inc.: *As the exec in charge of IT at a small software company, I've been seeing the same signs of a lack of readiness as I interview potential candidates. Our shop does a lot of t-SQL programming, so I administer a practical exam for all applicants to gauge their competency in t-SQL. Not only do undergraduates not understand the basics of relational databases, but also many applicants cannot complete the exam—and I'm talking simple selects with inner joins. I think there's too much theory being taught to CS students and not enough practical application in a variety of programming languages.*

Peter Lanoie, Clifton Park, NY: *As I encountered people in the workplace doing the same job as me, I found that some who were educated in more traditional [computer science] programs really couldn't program. Sure, they understood more of the theory than I did, but we weren't building DB engines or operating systems, we were making ASP Web sites ... Practical skills are an important part of a future programmer's technical education; core problem-solving skills are as, if not more, important.*

Brad B.: *I'm starting my fourth year using online classes from the University of Phoenix. Previous to starting those classes, I achieved an associate degree many years ago. I had a total of 10 weeks of classes that covered C programming. Those 10 weeks covered nothing more than basic logic; structs or other useful bits weren't covered there. With the almost full year of classes with UoP, I have yet to take another course that involved writing code. The closest that a class has come was one course that covered pseudocode. Checking my fourth-year classes, I will have a five-week course on SQL, two five-week courses on Java,*

and two five-week courses on .NET. This is for an IT degree with an emphasis on software engineering!

Anonymous by Request: *As a professor, I think the quality of education in the computer science field is not where it needs to be. When I first started teaching, I taught Intro to CS using C++, [and] on average about 10 percent to 30 percent of the class would fail; the final project in the class was comparable to a project I had to complete in my second or third week of my Intro class when I was an undergrad. Obviously, a high failure rate doesn't sit well with the powers that be, so the class was dumbed down.*

As a senior app developer for a supplemental insurance company, my job duties include interviewing prospective employees and mentoring junior developers. So far, I have met a few students who seem to know their stuff, but I've also had some frustrating interviews. For example, here's a basic question I'd ask: "What can you tell me about a database cursor?" Response: "Do you mean the little flashy thing on the screen?"

David Luxford, Pittsfield Township, MI.: *Until the last 10 years, most colleges had no idea what they were doing when it came to educating those bound for a computer job. At my first college, we were expected to program in C in upper-division courses, but had to teach ourselves. There were no classes on writing Windows GUIs, the NTFS filesystem, DirectX or driver development. Our local community college was better, but its program was only two years. There's a significant disconnect between the curriculum of software engineering programs and the actual skills you use. Using technology even only 2 years old leaves a graduate up to six years behind when he graduates. There is no education on QA, configuration management, virtual machines, installs, patching or deployment.*

Do you want to get in on the conversation? Write to me at mmeditor@microsoft.com.

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

© 2010 Microsoft Corporation. All rights reserved.

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

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

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

Introducing **OnTime 2010** **NEW!** Scrum Planning Board that Changes Everything!



FREE!
1-User License



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

Project Management • Bug Tracking • Scrum

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



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



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



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



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



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



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



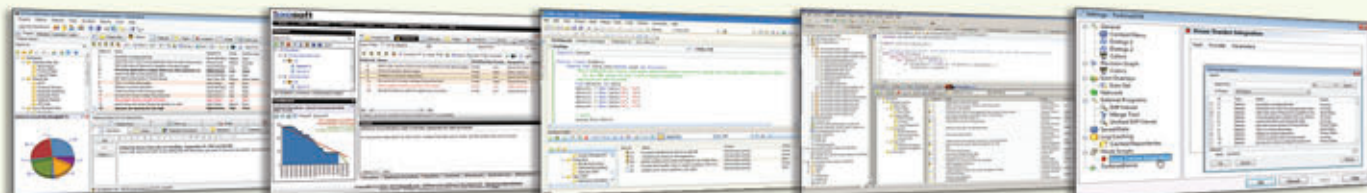
Notifications
Ensure team members are in sync with automatic email notifications and alerts.



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



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



Windows

Web

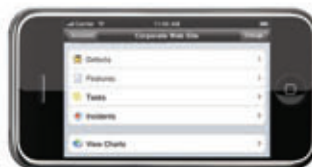
Visual Studio

Eclipse

Sub-Version

axosoft

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



800.653.0024
www.axosoft.com



Better Web Forms with the MVP Pattern

The advent of the Model-View-Controller (MVC) pattern is an important milestone in software development. It showed that designing applications with separation of concerns in mind improved both the development process and the finished application. It also offered a reproducible approach for putting that pattern into practice.

MVC is not perfect, though, so several variations of it appeared over the years.

Because it was devised in the 1980s, one problem that's surfaced is that MVC does not directly accommodate development for the Web. Adapting MVC to the Web took a few more years and led to the development of more specific MVC patterns such as Model2. (Model2 is the actual flavor of MVC implemented by Castle Mono-Rail and ASP.NET MVC.)

In a more general context, the Model-View-Presenter (MVP) pattern is an evolution of MVC that separates view and model neatly by placing the controller in between as a mediator. **Figure 1** illustrates the behavior of an application designed with the MVP pattern.

In this article, I'll first present a possible (and relatively standard) implementation of the MVP pattern for ASP.NET Web Forms and then discuss the application of the pattern, its benefits to the team, and compare it to ASP.NET MVC and Model-View-ViewModel (MVVM) as it has been implemented in Windows Presentation Foundation (WPF) and Silverlight.

MVP at a Glance

MVP is a derivative of the original MVC pattern developed at Taligent (now part of IBM) in the 1990s. The paper available for download at wildcrest.com/Potel/Portfolio/mvp.pdf offers a nice introduction to MVP and the ideas behind it.

The creators of MVP neatly separated the model (the data being worked on in the view) from the view/controller pair. They also renamed the controller as *presenter* to reinforce the idea that in the pattern, the role of the controller is that of a mediator between the user and the application. The presenter is the component that "presents" the UI to the user and accepts commands from the user. The presenter contains most of the presentation logic and knows how to deal with the view and the rest of the system, including back-end services and data layers.

A key innovation in MVP is the fact that the details of the view are abstracted to an

interface (or base class). The presenter talks to an abstraction of the view, which makes the presenter itself a highly reusable and highly testable class. This enables two interesting scenarios.

First, the presentation logic is independent from the UI technology being used. Subsequently, the same controller could be reused in Windows and Web presentation layers. In the end, the presenter is coded against an interface and it can talk to any object that exposes that interface—whether a Windows Forms object, an ASP.NET Page object, or a WPF Window object.

Second, the same presenter could work with different views of the same application. This is an important achievement with regard to Software as a Service (SaaS) scenarios where an application is hosted on a Web server and offered as a service to customers, each requiring its own customized UI.

It goes without saying that both benefits are not necessarily applicable in all situations. Whether these benefit you mostly depends on the application and navigation logic you expect to employ in your Windows and Web front end. However, when the logic is the same, you can reuse it via the MVP model.

MVP in Action

When implementing the MVP pattern, the first step is defining the abstraction for each required view. Each page in an ASP.NET application and each form in a Windows (or WPF/Silverlight) application will have its own interface to talk to the rest of the presentation layer. The interface identifies the data model that the view supports. Each logically equivalent view will have the same interface regardless of the platform.

The view abstraction incorporates the model the view recognizes and works with and can extend it with some ad hoc methods and

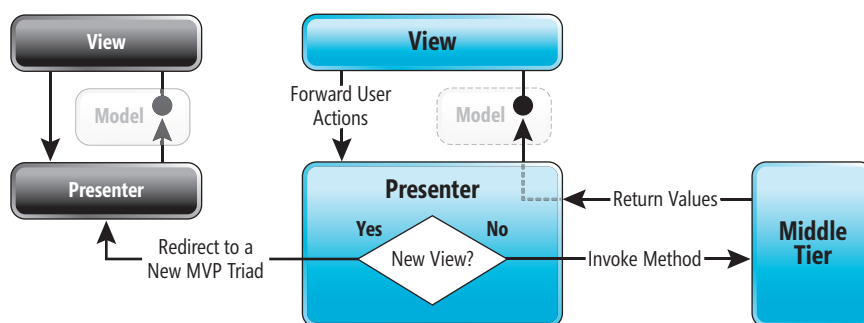


Figure 1 Using the MVP Pattern

Esri® Developer Network

Integrate Mapping and GIS into Your Applications



Give your users an effective way to visualize and analyze their data so they can make more informed decisions and solve business problems.

By subscribing to the Esri® Developer Network (EDNSM), you have access to the complete Esri geographic information system (GIS) software suite for developing and testing applications on every platform. Whether you're a desktop, mobile, server, or Web developer, EDN provides the tools you need to quickly and cost-effectively integrate mapping and GIS into your applications.

Subscribe to EDN and leverage the power of GIS to get more from your data. Visit www.esri.com/edn.



Figure 2 An Example of a View Abstraction

```
public interface IMemoFormView {
    String Title { get; set; }
    String Summary { get; set; }
    String Location { get; set; }
    String Tags { get; set; }
    DateTime BeginWithin { get; set; }
    DateTime DueBy { get; set; }
    String Message { get; set; }

    Int32 GetSelectedPriorityValue();
    void FillPriorityList(Int32 selectedIndex);
    Boolean Confirm(String message, String title);
    void SetErrorMessage(String controlName);
}
```

events useful to favor a smooth interaction between the presenter and the view. **Figure 2** shows a possible abstraction for the view rendered in **Figure 3** that's being used by a simple to-do list application.

In **Figure 3** you also see how members in the interface match visual elements in the form.

The fundamental point is that any interaction between the presenter and the UI must happen through the contract of the view. Any button clicking, any selection, any typing must be forwarded to the presenter and handled there. If the presenter needs to query for some data in the view, or to pass data down to the view, there should be a method in the interface to account for that.

Implementing the View Contract

The interface that represents the view must be implemented by the class that represents the view itself. As mentioned, the view class is the page in ASP.NET, the form in Windows Forms, the Window in WPF and the user control in Silverlight. **Figure 4** shows an example for Windows Forms.

As you can see, properties are implemented as wrappers for some properties on visual controls. For example, the Title property is a wrapper for the Text property of a TextBox control. Similarly, the DueBy property wraps the Value property of a DatePicker control. More importantly, the interface shields the presenter

class from the details of the UI for a given platform. The same presenter class created to interact with the IMemoFormView interface can deal with any object that implements the interface, blissfully ignoring the details of the programming interface of underlying controls.

How would you deal with UI elements that require a collection of data, such as a drop-down list? Should you use data binding (as in **Figure 4**) or should you opt for a simpler approach that keeps the view passive and devoid of any presentation logic?

That's up to you. In response to these kinds of questions, the MVP pattern has been split in two separate patterns—Supervising Controller and Passive View—whose primary difference is just the amount of code in the view. Using data binding for populating the UI (see **Figure 4**) adds some presentation logic to the view and would make it gain the flavor of a supervising controller.

The more logic you have in the view, the more you should care about testing. And testing a piece of UI is a task that can't be easily automated. Going for a supervising controller or opting for a thinner and dumber view is merely a judgment call.

How would you deal with
UI elements that require a
collection of data?

The Presenter Class

The controls in the view capture any user gesture and trigger an event to the view, such as a button-click or a selected-index change. The view contains simple event handlers that dispatch the call to the presenter that's in charge the view. When the view is loaded for the first time, it creates an instance of its presenter class and saves that internally as a private member. **Figure 5** shows the typical constructor of a Windows Form.

The presenter class typically receives a reference to the view through its constructor. The view holds a reference to the presenter and the presenter holds a reference to the view. However, the presenter knows the view only through the contract. The presenter works by segregating any view object it receives to its contracted view interface. **Figure 6** shows the basics of a presenter class.

The constructor receives and caches the reference to the view and initializes the view using the public interface represented by the contract. The context object you see used in the code of **Figure 6** is any input data the presenter needs to receive from the caller in order to initialize the view. This information is not necessary in all cases, but it turns out to be necessary when you use the form to edit some data or when you have dialog boxes to display some information.

Figure 3 Binding Members of the Interface to Visual Elements

version 17

LEADTOOLS®

The World Leader in Imaging Development SDKs



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

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

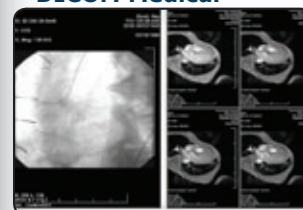
LEADTOOLS provides developers easy access to decades of expertise in color, grayscale, document, medical, vector and multimedia imaging development. Install LEADTOOLS to eliminate months of research and programming time while maintaining high levels of quality, performance and functionality.

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

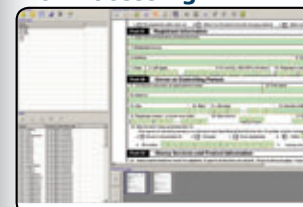
Document



DICOM Medical



Form Recognition & Processing



Barcode



Multimedia



Vector



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

Figure 4 A Possible Implementation of the View Class

```
public partial class MemoForm : Form, IMemoFormView {
    public string Title {
        get { return memoForm_Text.Text; }
        set { memoForm_Text.Text = value; }
        ...
    }

    public DateTime DueBy {
        get { return memoForm_DueBy.Value; }
        set { memoForm_DueBy.Value = value; }
    }

    public int GetSelectedPriorityValue() {
        var priority =
            memoForm_Priority.SelectedItem as PriorityItem;
        if (priority == null)
            return PriorityItem.Default;
        return priority.Value;
    }
}

public void FillPriorityList(int selectedIndex) {
    memoForm_Priority.DataSource =
        PriorityItem.GetStandardList();
    memoForm_Priority.ValueMember = "Value";
    memoForm_Priority.DisplayMember = "Text";
    memoForm_Priority.SelectedIndex = selectedIndex;
}

public void SetErrorMessage(string controlName) {
    var control = this.GetControlFromId(controlName);
    if (control == null)
        throw new NullReferenceException(
            "Unexpected null reference for a form control.");

    memoForm_ErrorManager.SetError(control,
        ErrorMessages.RequiredField);
}

...
}
```

Initializing the view is as simple as assigning values to the members of a class, except that now any assignment results in an update to the UI.

The presenter class also contains a number of methods that execute in response to any requests from the UI. Any clicking or user action is bound to a method on the presenter class:

```
private void memoForm_OK_Click(
    object sender, EventArgs e) {
    presenter.Ok();
}
```

The presenter method uses the view reference to access input values and updates the UI in the same way.

Navigation in MVP

The presenter is also responsible for navigation within the application. In particular, the presenter is responsible for enabling (or disabling) sub-views and command navigation to the next view.

A sub-view is essentially a subset of the view. It's typically a panel that can be expanded or collapsed according to the context or perhaps a child window—either modal or modeless. The presenter controls the visibility of sub-views through members (mostly Boolean members) on the view interface.

What about transferring control to another view (and presenter)? You create a static class that represents the application controller—

that is, the central console that holds all the logic to determine the next view. **Figure 7** shows the diagram of the application controller.

The application controller class represents the shell that presenters invoke to navigate elsewhere. This class will have a `NavigateTo` method that implements the workflow that determines the next view or that simply moves to the specified view. The workflow can be anything—as complex as a real workflow or simply a sequence of IF statements. The logic of the workflow can be statically coded in the application controller or imported from an external and pluggable component (see **Figure 8**).

The actual navigation logic in the workflow component will use a platform-specific solution to switch to a different view. For Windows Forms it will use methods to open and display forms; in ASP.NET it will use the `Redirect` method on the `Response` object.

Figure 6 A Sample Presenter Class

```
public class MemoFormPresenter {
    private readonly IMemoFormView view;

    public MemoFormPresenter(IMemoFormView theView) {
        view = theView;
        context = AppContext.Navigator.Argument
            as MemoFormContext;
        if (_context == null)
            return;
    }

    public void Initialize() {
        InitializeInternal();
    }

    private void InitializeInternal() {
        int priorityIndex = _context.Memo.Priority;
        if (priorityIndex >= 1 && priorityIndex <= 5)
            priorityIndex--;
        else
            priorityIndex = 2;

        if (_context.Memo.BeginDate.HasValue)
            _view.BeginWithin = _context.Memo.BeginDate.Value;
        if (_context.Memo.EndDate.HasValue)
            _view.DueBy = _context.Memo.EndDate.Value;
        _view.FillPriorityList(priorityIndex);
        _view.Title = _context.Memo.Title;
        _view.Summary = _context.Memo.Summary;
        _view.Tags = _context.Memo.Tags;
        _view.MemoLocation = _context.Memo.Location;
    }

    ...
}
```

Figure 5 Creating an MVP Form

```
public partial class Form1 :
    Form, ICustomerDetailsView {

    private MemoFormPresenter presenter;

    public Form1() {
        // Framework initialization stuff
        InitializeComponent();
        // Instantiate the presenter
        presenter = new MemoFormPresenter(this);
        // Attach event handlers
        ...
    }

    private void Form1_Load(
        object sender, EventArgs e) {

        presenter.Initialize();
    }

    ...
}
```


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

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

The Altova MissionKit includes multiple tools for software architects:

UModel® – UML tool for software modeling

- Support for all UML diagram types, plus 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

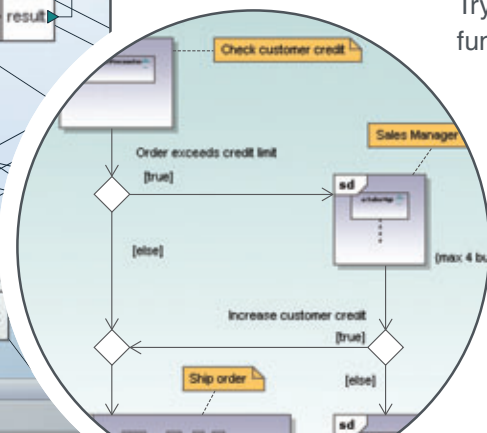
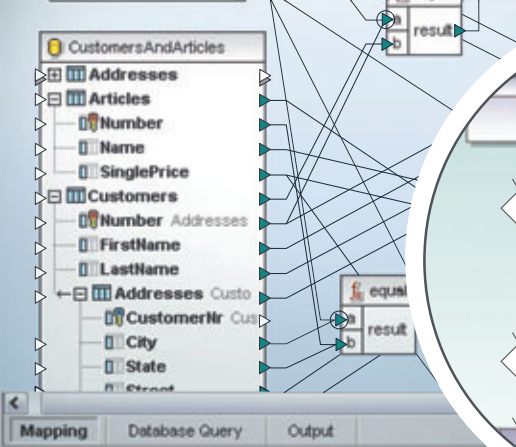
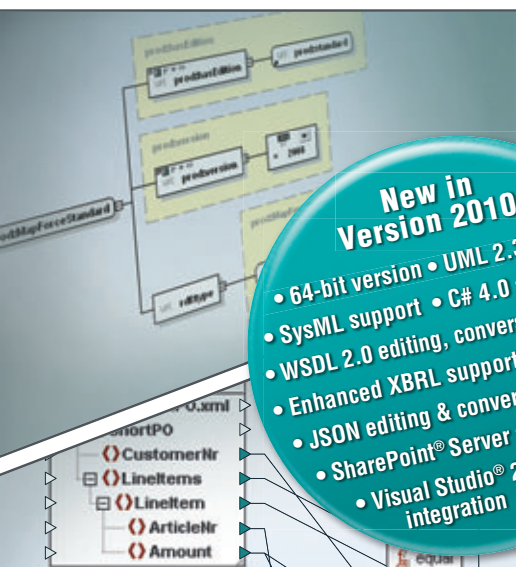
Plus up to five additional tools...

 Download a 30 day free trial!

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

**New in
Version 2010:**

- 64-bit version • UML 2.3 support
- SysML support • C# 4.0 support
- WSDL 2.0 editing, conversion, mapping
- Enhanced XBRL support
- JSON editing & conversion
- SharePoint® Server support
- Visual Studio® 2010 integration



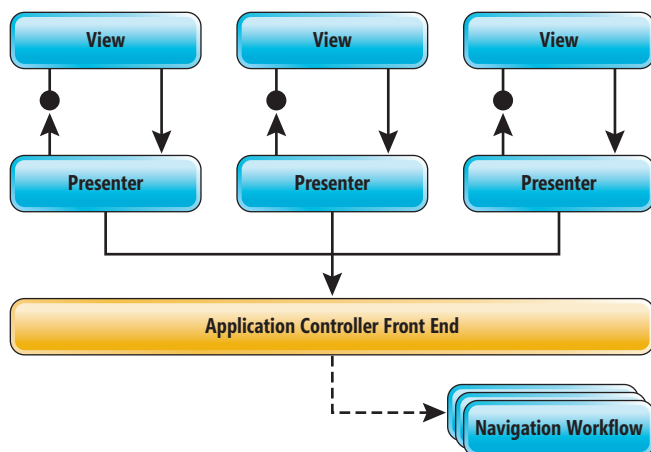


Figure 7 The Application Controller

MVP and ASP.NET MVC

ASP.NET MVC is based on a flavor of the MVC pattern that has a few things in common with MVP. The controller in MVC is a mediator between the view and the back end. The controller doesn't hold a reference to the view, but fills up a model object and passes that to the view using the services of an intermediate component—the view engine.

In a way, the view is abstracted through the model, whose structure reflects the characteristics of the view and its UI. Navigation is managed by the controller, which selects the next view from the context of each action. It does that using some built-in logic. Should the logic be particularly complex for a given controller method—frankly, not something that will happen every day—you can always introduce a workflow component that determines the next view to select.

What about Web Forms? Web Forms lends itself well to host an MVP implementation. However, it should be clear that all you

Figure 8 Implementation of an Application Controller

```

public static class ApplicationController {
    private static INavigationWorkflow instance;
    private static object navigationArgument;

    public static void Register(
        INavigationWorkflow service) {
        if (service == null)
            throw new ArgumentNullException();
        instance = service;
    }

    public static void NavigateTo(string view) {
        if (instance == null)
            throw new InvalidOperationException();
        instance.NavigateTo(view);
    }

    public static void NavigateTo(
        string view, object argument) {
        if (instance == null)
            throw new InvalidOperationException();
        navigationArgument = argument;
        NavigateTo(view);
    }

    public static object Argument {
        get { return navigationArgument; }
    }
}
  
```

get is adding layers in the context of the postback event. Anything before the postback cannot be incorporated, nor can anything that happens after the postback event. A full MVP implementation that expands to cover the full lifecycle is not possible in Web Forms, but even adding MVP around the postback is a good thing and will significantly increase the level of testability of Web Forms pages.

MVP and MVVM

What about, instead, MVP and MVVM in the context of WPF and Silverlight applications? MVVM is a variation of MVP also known as Presentation Model. The idea is that the view model is incorporated in the presenter class and the presenter class exposes public members that the view will read and write. This happens through two-way data binding. At the end of the day, you can call MVVM as a special flavor of MVP particularly suited to rich UIs and to frameworks (like WPF) that promote this ability.

The presenter is also responsible for navigation within the application.

In MVVM, the view is data-bound to properties on the presenter class (the view model). Anything the user does updates these properties in the presenter. Any requests from the user (commands in WPF) are handled via a method on the presenter class. Any results the presenter method calculates are stored in the view model and made available via data binding to the view. In WPF and Silverlight, there's nothing that prevents you from using a manual implementation of the MVP pattern. However, it turns out that tools such as Blend will make it simpler yet effective to use MVVM via data binding.

Postback

MVP provides guidance on how to manage heaps of views and, quite obviously, comes at a cost: the cost of increased complexity in the application code. As you can imagine, these costs are easier to absorb in large applications than in simple programs. MVP, therefore, is not just for any application. Based on a contract that represents the view, MVP allows for designers and developers to work in parallel, which is always a good thing in any development scenario. MVP keeps the presenter class as a standalone and isolated from the view. In Web Forms, MVP represents the only reasonable way to add testability at least to the code that executes the postback. ■

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

THANKS to the following technical experts for reviewing this article:
Don Smith and Josh Smith

Now v4.0!

Why is Amyuni PDF so interesting?



Proven

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

High-Performance

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

Rapid Integration

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

Expertise

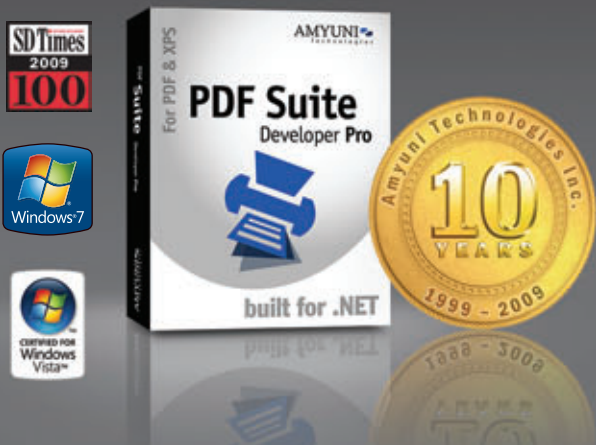
Produce accurate and stable PDF documents using reliable tools built by experts with over ten years of experience.

OEM Licenses

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

Customization

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



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

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

www.amyuni.com

All trademarks are property of their respective owners. © 1999-2009 AMYUNI Technologies. All rights reserved.

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

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

AMYUNI 
Technologies

Studio Enterprise has 8,310,799 lines of code and would cost

\$710,467,541

and take 10 Years to develop*

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

Do you develop for the web?

Our ASP.NET
AJAX &
Silverlight controls
would cost
\$159,009,712



Do you have mobile versions of your ASP.NET apps?

Building
our Mobile Web
controls would
cost
\$3,217,145



Only build desktop apps?

Our
WinForms
& WPF controls
would cost
\$233,189,875



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



Welcome to

DEVTOPIA

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

67
WinForms Controls

59
Silverlight Controls

46
ASP.NET AJAX Controls

40
WPF Controls

34
ActiveX Controls

29
iPhone & Mobile Controls

275
CONTROLS in
Studio Enterprise

Learning Made Easy.
Documentation Pages per Studio

3,186 pgs
iPhone/Mobile

7,758 pgs
ASP.NET
AJAX

8,111 pgs
WPF

7,981 pgs
WinForms

43,539 pgs
Studio Enterprise

Get developing right away.
2,791
WinForms

Code snippets per Studio
847
Silverlight

513
WPF

1,340
ActiveX

706
ASP.NET
AJAX

691
iPhone
& Mobile

Studio Enterprise Code Snippets
6,888

Check out the demo with source code

STUDIO
ENTERPRISE 2010 ^{New!} v2

DOWNLOAD TODAY @ COMPONENTONE.COM/DEVTOPIA



IronRuby on Windows Phone 7

A few years ago, I was a 100 percent .NET guy. I didn't have a clue about the rest of the development world, and I was quite happy in my own bubble. Then, kind of by mistake, I learned Ruby, and the experience was jaw-dropping. The way things get done using the language's built-in features was striking to me.

Still, you can take the person out of the .NET world, but you can't take the .NET world out of the person. So as soon as I heard Microsoft was developing an implementation of the Ruby language—called IronRuby—on top of the Microsoft .NET Framework, I got pretty excited and dove right into it.

With IronRuby, the .NET world and the Ruby world are now connected. This enables endless new possibilities, and the benefits of such a connection are nothing short of phenomenal.

In this article, I'm going to tell you about one of the benefits that's important to both .NET Framework and Ruby developers—you can use IronRuby on Windows Phone 7.

What Is IronRuby?

In 2006, Microsoft announced the development of IronRuby. It took more than three years to develop, and in April the IronRuby team announced the first stable version of IronRuby: version 1.0.

With IronRuby, the .NET world and the Ruby world are now connected.

IronRuby supports the entire feature set of the Ruby language with a unique addition: integration between Ruby code and .NET Framework code. This integration is fairly seamless and requires little more than loading a .NET Framework assembly to the Ruby context. For example, this IronRuby code loads the System.Windows.Forms assembly and takes advantage of its classes:

```
require 'System.Windows.Forms'

include System::Windows::Forms

form = Form.new
form.height = 200
form.width = 400
form.text = "IronRuby Window"
form.show_dialog
```

This integration is possible thanks to the Dynamic Language Runtime (DLR), a layer added to the .NET Framework infrastructure to provide common services to dynamic languages written on

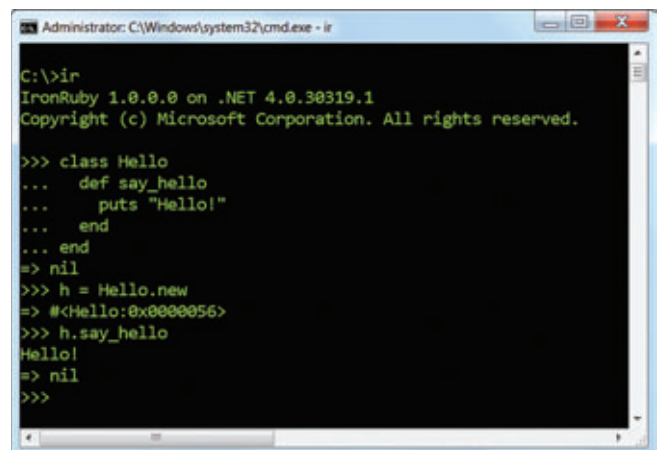


Figure 1 Using the IronRuby Console

top of the framework. The DLR is written on top of the CLR and makes it much easier to implement dynamic languages on top of .NET. This is one of the main reasons for the rise of .NET Framework dynamic languages we've seen lately, including IronRuby, IronPython, IronJS, Nua, ClojureCLR and others.

Key Features of IronRuby

Ruby is a dynamic language and so is IronRuby. This means there's no compiler at hand, and most of the operations done during compilation and build time in static languages are done during run time. This behavior provides a variety of features that are difficult or impossible to achieve in most current static languages.

Interoperability with .NET Framework Objects The Ruby language has various implementations: MRI (which is the original one), JRuby, Rubinius, MacRub, IronRuby and others. What makes IronRuby stand out from the crowd is its ability to conveniently interact with .NET Framework objects. That interoperability goes both ways—.NET Framework objects are available from IronRuby code and IronRuby objects are available from .NET Framework code.

Dynamic Typing IronRuby variable types are calculated during run time, so there's no need to specify the types in your code. However, that doesn't mean that IronRuby doesn't have types. It does, and every type has its own rules, just like types in static languages.

This column is based on a prerelease version of Windows Phone 7. All information is subject to change.

Code download available at code.msdn.microsoft.com/mag201009GoPlaces.

FULL
Visual Studio
2010
support

RadControls for Silverlight

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

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

www.telerik.com/Silverlight

 **telerik**
deliver more than expected

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

Figure 2 An Example of Duck Typing

```
class Human
  def say_hi
    puts "Hi!"
  end
end
class Duck
  def say_hi
    puts "Quack!"
  end
end

def introduce(obj)
  obj.say_hi
end

human = Human.new
duck = Duck.new

introduce(human) # prints "Hi!"
introduce(duck) # prints "Quack!"
```

This code sample demonstrates the dynamic typing mechanism in a few simple steps:

```
# Declaring a numeric variable
a = 1

# The variable is of a numeric type
# and therefore numeric operations are available
a = a * 2 + 8 / 4

# The next line will raise an exception
# because it is not possible to add a string to a number
a = a + "hello"

# However, the next line is entirely legit and will result
# in changing the variable type to String
a = "Hello"
```

The Interactive Console Similar to the Windows command prompt, the interactive console is an application that retrieves IronRuby code and immediately executes it. The execution flow is also known as Read-Evaluate-Print-Loop (REPL). You can define variables, methods and even classes, load IronRuby files or .NET Framework assemblies and use them instantly. For example, **Figure 1** shows a simple console session that creates a class and immediately uses it.

Duck Typing IronRuby is an object-oriented language. It supports classes, inheritance, encapsulation and access control, like you'd expect from an object-oriented language. However, it doesn't support interfaces or abstract classes, like many static languages do.

This isn't a flaw in the language design, though. With dynamic typing, declaring code contracts such as interfaces or abstract classes

Figure 3 Adding a Method to a Class After It Has Been Declared

```
# Creating a class with no methods
class Demo
end

# Creating an instance of class Demo
d = Demo.new

# Opening the class and adding a new method - hello_world
class Demo
  def hello_world
    puts "hello world"
  end
end

# Using the newly added method on the class instance
d.hello_world # prints "hello world"
```

becomes redundant. The only thing that matters about an object is whether it defines a specific method or not, and there's no need to mark it when it does. This is known as duck typing—if it quacks like a duck and it swims like a duck, it's a duck, and there's no need to stamp it to consider it as a duck.

For example, the code sample in **Figure 2** contains two classes with a method named *say_hi* and another general method named *introduce* that retrieves an object and executes its *say_hi* method. (Notice the absence of interfaces or other marking mechanisms.)

Metaprogramming IronRuby comes with powerful metaprogramming capabilities. Metaprogramming is a way to add, change and even remove methods during run time. For example, it's possible to add methods to a class, write methods that define other methods or remove method definitions from an existing class. **Figure 3** adds a method to a class that's reflected to all current and future instances of that class.

Moreover, there are special methods that can be used to catch calls to undefined methods or constants. Using these methods makes it easy to support dynamic method names such as *find_by_[column name]* where *[column name]* can be replaced with any value such as *find_by_name*, *find_by_city* or *find_by_zipcode*.

RubyGems The Ruby language, as powerful as it is, wouldn't have become such a huge success without the external libraries that can be installed and used with it.

The main method of installing Ruby libraries is via the RubyGems system. It's a package manager that helps distribute and install Ruby libraries, which are called *gems*. There are thousands of free gems available, covering almost every programming aspect and task, including testing frameworks, tax calculation libraries, Web development frameworks and more.

You should be aware that some RubyGems depend on C libraries. These gems can't run on the current version of IronRuby unless the C libraries are ported to plain Ruby or to C#.

The Community One of the best things about IronRuby is that you get access to the Ruby community. This includes valuable content in dozens of forums, mailing lists, chat rooms and blogs provided by people who are willing to help with any question. Don't hesitate to take advantage of these resources; they're extremely useful.

IronRuby and Silverlight

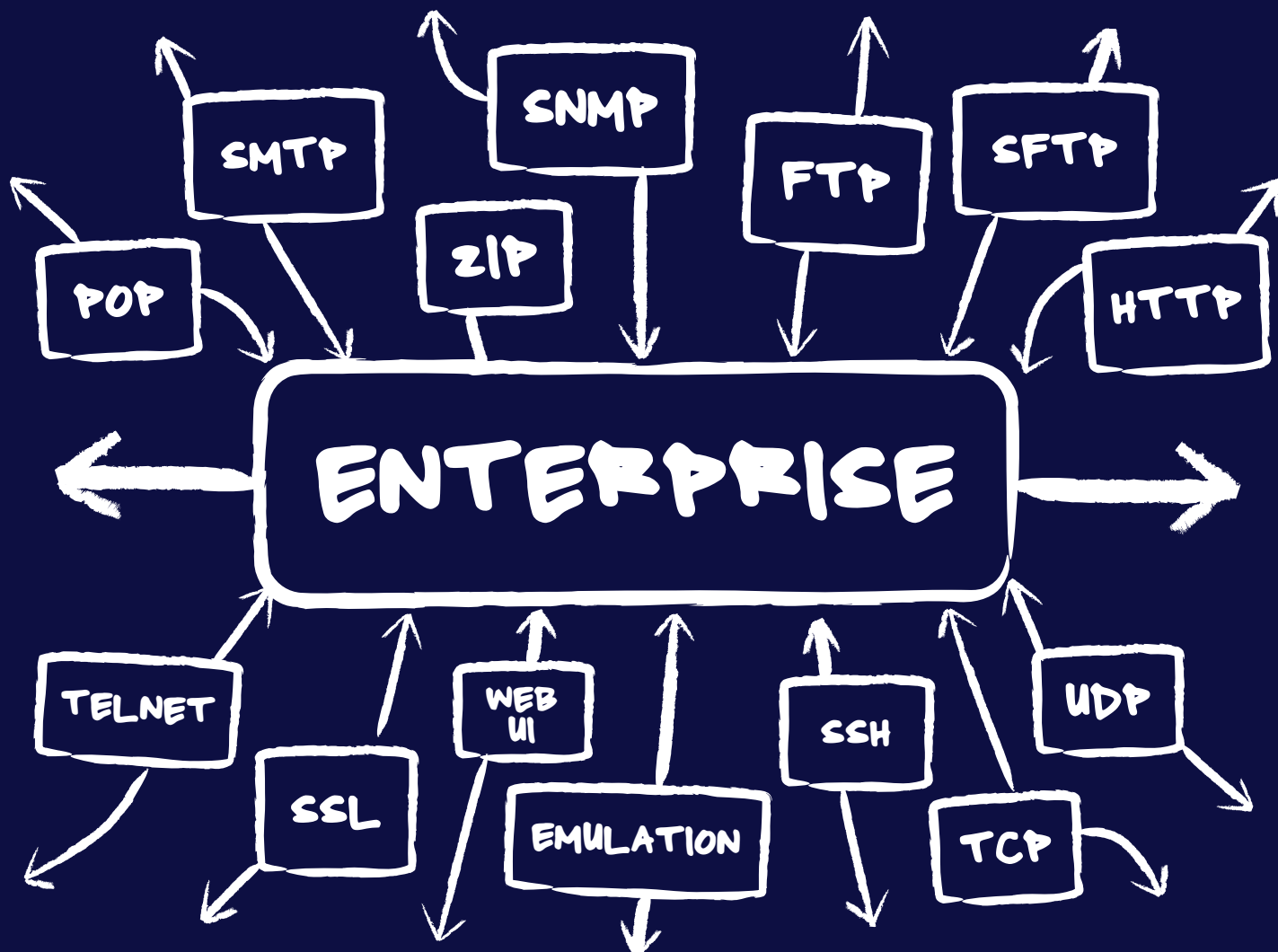
Silverlight 2 introduced a new and important feature: support for DLR languages. As a result, developers can use IronRuby with Silverlight applications, from incorporating it in the application to writing entire Silverlight applications with it.

But wait, Silverlight is running on Windows Phone 7, right? Exactly.

Windows Phone 7

The next Microsoft mobile platform, Windows Phone 7, is expected by some to become a game-changer in the smartphone industry. Apart from the standard multi-touch capabilities and a shiny new UI, the best news about Windows Phone 7 from a developer's perspective is that Silverlight is its development platform.

It's a smart move by Microsoft to make use of a well-established technology, thus enabling a large number of developers to create mobile applications with an easy, almost unnoticeable, learning curve.



Internet Connectivity for the Enterprise

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



PowerSNMP for ActiveX and .NET

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

PowerWEB for ASP.NET

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

PowerTCP for ActiveX and .NET

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

SSH
UDP
TCP
SSL

FTP
SFTP
HTTP
POP

SMTP
IMAP
S/MIME
Ping

DNS
Rlogin
Rsh
Rexec

Telnet
VT Emulation
ZIP Compression
more...

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

Download a fully functional product trial today!

 **DART.com**

Figure 4 IronRuby Code File to Run on Windows Phone 7

```
# Include namespaces for ease of use
include System::Windows::Media
include System::Windows::Controls

# Set the titles
Phone.find_name("ApplicationTitle").text = "MSDN Magazine"
Phone.find_name("PageTitle").text = "IronRuby& WP7"

# Create a new text block
textBlock = TextBlock.new
textBlock.text = "IronRuby is running on Windows Phone 7!"
textBlock.foreground = SolidColorBrush.new(Colors.Green)
textBlock.font_size = 48
textBlock.text_wrapping = System::Windows::TextWrapping.Wrap

# Add the text block to the page
Phone.find_name("ContentGrid").children.add(textBlock)
```

Because DLR languages are capable of running within the Silverlight environment, you can take advantage of IronRuby and use it to write Windows Phone 7 applications.

However, there are some limitations you should be aware of. Windows Phone 7 comes with the .NET Compact Framework, which is a subset of the .NET Framework. The Compact Framework is designed for mobile and embedded applications and contains approximately 30 percent of the full .NET Framework. Consequently, numerous classes are missing, and this affects how IronRuby works.

The feature that makes IronRuby stand out from the crowd is the ability to conveniently interact with .NET Framework objects.

The main missing feature that affects IronRuby is the Reflection.Emit namespace. IronRuby uses this feature to compile code on the fly to make applications run faster. However, it's only a performance optimization and not a component necessary for running simple scripts and applications.

Another limitation concerns the way new Windows Phone 7 applications are created. Such applications can be created only from Visual Studio and only in C#. This requirement forces developers to write code in C# that initiates the IronRuby code.

The last important limitation is that RubyGems won't work on Windows Phone 7. Hence, to use a gem, you have to include its code files within the application files and use them as any other IronRuby code files.

Building a Simple IronRuby Application on Windows Phone 7

To start an IronRuby-driven Windows Phone 7 application, you first need to install the Windows Phone 7 Developer Tools, which can be downloaded from developer.windowsphone.com.

After the tools are installed, open Visual Studio and go to File | New | Project. In the New Project dialog select the "Silverlight for

Windows Phone" category and then choose the "Windows Phone Application" project template. Name it and continue.

As soon as the new project opens, you'll notice that a simple XAML file has been created for you. Note that XAML is required for Silverlight in general and isn't language-dependent. Therefore, even though the application code will be written in IronRuby, you must use XAML to create the UI. In this simple application, the default XAML file is enough, so no changes need to be made here.

The interesting part of this simple application is the code. Before we dive into that, however, we need to add references to the IronRuby and DLR assemblies. These assemblies aren't the regular ones; we need the Windows Phone 7-ready assemblies, which you can retrieve from ironruby.codeplex.com/releases/view/43540#DownloadId=133276. You'll find the needed assemblies inside the silverlight/bin folder in the downloaded package.

Next, we need to write the IronRuby code. Add a new text file to the application and name it MainPage.rb. In addition, to ease the deployment to the phone, open the properties of this file and change the "Build Action" property to "Embedded Resource."

Then paste the code from **Figure 4** into the file.

The IronRuby code in **Figure 4** is pretty straightforward; we set the titles, create a text block with some text and add it to the page. Note that you can use everything in the Ruby language (not done here), such as classes, metaprogramming and libraries, with the aforementioned limitations of running within the Windows Phone environment.

Now all that's left is to actually execute the IronRuby code. To do so when the application loads, the code from **Figure 5** should be added to the MainPage class constructor, which is located inside the MainPage.xaml.cs file.

The code in **Figure 5** is fairly short and gracefully demonstrates how easy it is to run IronRuby code from C# code.

In addition, make sure to add these using statements to the class:

```
using System.Reflection;
using System.IO;
using Microsoft.Scripting.Hosting;
using IronRuby;
```

The third line of code in **Figure 5** loads the System.Windows.Media assembly into the IronRuby context, which enables the code to interoperate with this assembly's classes and enums.

Figure 5 Adding Code to Execute IronRuby Code from the Class Constructor

```
// Allow both portrait and landscape orientations
SupportedOrientations = SupportedPageOrientation.PortraitOrLandscape;

// Create an IronRuby engine and prevent compilation
ScriptEngine engine = Ruby.CreateEngine();

// Load the System.Windows.Media assembly to the IronRuby context
engine.Runtime.LoadAssembly(typeof(Color).Assembly);

// Add a global constant named Phone, which will allow access to this class
engine.Runtime.Globals.SetVariable("Phone", this);

// Read the IronRuby code
Assembly execAssembly = Assembly.GetExecutingAssembly();
Stream codeFile =
    execAssembly.GetManifestResourceStream("SampleWPApp.MainPage.rb");
string code = new StreamReader(codeFile).ReadToEnd();

// Execute the IronRuby code
engine.Execute(code);
```



Figure 6 An IronRuby-Driven Application Running on Windows Phone 7

The next line allows the IronRuby code to access the current Silverlight page. This line exposes the current instance (*this*) to the IronRuby code via a constant named *Phone*.

The rest of the code reads the IronRuby code from the embedded file (note that the application namespace should be added to the file name, so *MainPage.rb* becomes *SampleWPApp.MainPage.rb*) and then executes it using the engine instance.

The IronRuby and Windows Phone 7 platforms are both new and they're getting better all the time.

And that's it. We've created an application that, once loaded, runs IronRuby, which, in turn, changes the titles and adds a text block to the Silverlight page. All that's left is to run the application, and the result is shown in **Figure 6**.

Getting Better All the Time

Even though the workflow isn't perfect when using IronRuby on Windows Phone 7, and you need to keep the various limitations in mind, this is only the beginning. The IronRuby and Windows Phone 7 platforms are both new and they're getting better all the time.


This combination opens up many possibilities, to both .NET Framework developers and Ruby developers. Now, .NET developers can take advantage of the incredible power of the Ruby language when writing Windows Phone 7 applications, such as incorporating an IronRuby console into their apps or providing extensibility capabilities. And Ruby developers, on the other end, can—for the first time—write mobile applications using their language.

This is, without a doubt, the dawn of a brave new world with a lot of opportunities and possibilities. And it's all in the palm of your hands. ■

SHAY FRIEDMAN is a Microsoft Visual C#/IronRuby MVP and the author of "IronRuby Unleashed" (Sams, 2010). He's working as a dynamic languages leader in Sela Group where he consults and conducts courses around the world. Read his blog at IronShay.com.

THANKS to the following technical expert for reviewing this article:
Tomas Matousek

msdnmagazine.com



Discover **SCALABLE** Performance


SCALEOUT STATESERVER®


Simply Powerful Distributed Caching

As a software architect, you know the importance of combining simplicity with power. ScaleOut StateServer's distributed cache gives you both.

Its powerful architecture simplifies both configuration and management so that you can quickly - and easily - tap into blazing performance, scalability, and 24x7 availability.

Give your server farm and grid applications the performance boost they need. Download your **FREE** trial copy of ScaleOut StateServer today!





SCALEOUT SOFTWARE

www.scaleoutsoftware.com/eval | (503) 643-3422



XCEED DataGrid for WPF

Serious recognition

Microsoft® Visual Studio® Team System 2010

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

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

IBM® U2 SystemBuilder™

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

*Vincent Smith,
U2 Tools Product Manager at IBM*

**NEW FOR
SILVERLIGHT!**



XCEED
DataGrid
for Silverlight

The only Silverlight datagrid that:

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

Datagrids, transformed

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



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

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

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

Try it live on xceed.com

XCEED
MULTI-TALENTED COMPONENTS



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

Simplify Asynchronous Programming with Tasks

Igor Ostrovsky

Asynchronous programming is a collection of techniques for implementing expensive operations that run concurrently with the rest of the program. One domain where asynchronous programming often comes up is in the context of programs with a graphical UI: It's generally unacceptable to freeze the UI while an expensive operation completes. Also, asynchronous operations are important for server applications that need to handle multiple client requests concurrently.

Representative examples of asynchronous operations that come up in practice include sending a request to a server and waiting for a response, reading data from the hard disk and running an expensive computation such as a spell check.

Consider the example of an application with a UI. The app could be built with Windows Presentation Foundation (WPF) or Windows Forms. In such an application, most of your code executes on the UI thread because it executes the event handlers

for events that originate from the UI controls. When the user clicks a button, the UI thread will pick up the message, and execute your Click event handler.

Now, imagine that in the Click event handler, your application sends a request to a server and waits for a response:

```
// !!! Bad code !!!
void Button_Click(object sender, RoutedEventArgs e) {
    WebClient client = new WebClient();
    client.DownloadFile("http://www.microsoft.com", "index.html");
}
```

There's a major problem in this code: downloading a Web site can take several seconds or longer. In turn, the call to Button_Click can take several seconds to return. That means the UI thread will be blocked for several seconds and the UI will be frozen. A frozen interface makes for a poor user experience and is almost always unacceptable.

To keep the application UI responsive until the server responds, it's important that the download isn't a synchronous operation on the UI thread.

Let's try to fix the problem of the frozen UI. One possible (but suboptimal) solution is to communicate with the server on a different thread so the UI thread remains unblocked. Here's an example that uses a thread-pool thread to talk to the server:

```
// Suboptimal code
void Button_Click(object sender, RoutedEventArgs e) {
    ThreadPool.QueueUserWorkItem(_ => {
        WebClient client = new WebClient();
        client.DownloadFile(
            "http://www.microsoft.com", "index.html");
    });
}
```

This article discusses:

- Problems with threaded operations
- Event pattern
- IAsyncResult pattern
- Task pattern

Technologies discussed:

Microsoft .NET Framework 4

This code sample fixes the problem of the first version: now the `Button_Click` event does not block the UI thread, but the thread-based solution has three significant problems. Let's take a closer look at these problems.

Problem 1: Wasted Thread-Pool Threads

The fix I just demonstrated uses a thread from the thread pool to send a request to the server and waits until the server responds.

The thread-pool thread is going to sit around blocked until the server responds. The thread cannot be returned to the pool until the call to `WebClient.DownloadFile` completes. Blocking a thread-pool thread is much better than blocking the UI thread because the UI will not freeze, but it does waste one thread from the thread pool.

If your application occasionally blocks a thread-pool thread for a while, the performance penalty may be negligible. But if your application does it a lot, its responsiveness can degrade due to pressure on the thread pool. The thread pool will attempt to cope by creating more threads, but that comes at a noticeable performance cost.

All other patterns of asynchronous programming presented in this article fix the problem of wasted thread-pool threads.

Problem 2: Returning the Result

There's another difficulty with using threads for asynchronous programming: returning a value from the operation that executed on the helper thread gets a little messy.

In the initial example, the `DownloadFile` method writes the downloaded Web page into a local file, and so it has a void return value. Consider a different version of the problem—instead of writing the downloaded Web page into a file, you want to assign the received HTML into the `Text` property of a `TextBox` (named `HtmlTextBox`).

A naïve—and wrong—way to implement this would be as follows:

```
// !!! Broken code !!!
void Button_Click(object sender, RoutedEventArgs e) {
    ThreadPool.QueueUserWorkItem(_ => {
        WebClient client = new WebClient();
        string html = client.DownloadString(
            "http://www.microsoft.com", "index.html");
        HtmlTextBox.Text = html;
    });
}
```

The problem is that a UI control—`HtmlTextBox`—is getting modified from a thread-pool thread. That's an error because only the UI thread is allowed to modify the UI. This restriction is present in both WPF and Windows Forms, for very good reasons.

To fix this issue, you can capture the synchronization context on the UI thread and then post a message to it on the thread-pool thread:

```
void Button_Click(object sender, RoutedEventArgs e) {
    SynchronizationContext ctx = SynchronizationContext.Current;
    ThreadPool.QueueUserWorkItem(_ => {
        WebClient client = new WebClient();
        string html = client.DownloadString(
            "http://www.microsoft.com");
        ctx.Post(state => {
            HtmlTextBox.Text = (string)state;
        }, html);
    });
}
```

It's important to recognize that the problem of returning a value from a helper thread is not limited to applications with UIs. In general, returning a value from one thread to another is a tricky issue that requires usage of synchronization primitives.

Figure 1 Methods for an Event-Based Pattern

```
public class AsyncExample {
    // Synchronous methods.
    public int Method1(string param);
    public void Method2(double param);

    // Asynchronous methods.
    public void Method1Async(string param);
    public void Method1Async(string param, object userState);
    public event EventHandler Method1Completed;

    public void Method2Async(double param);
    public void Method2Async(double param, object userState);
    public event EventHandler Method2Completed;

    public void CancelAsync(object userState);

    public bool IsBusy { get; }

    // Class implementation not shown.
    ...
}
```

Problem 3: Composing Asynchronous Operations

Explicitly working with threads also makes it difficult to compose asynchronous operations. For example, to download multiple Web pages in parallel, the synchronization code gets even more difficult to write and more error-prone.

Such an implementation would maintain a counter of asynchronous operations that are still executing. The counter would have to be modified in a thread-safe manner, say by using `Interlocked.Decrement`. Once the counter reaches zero, the code that processes the downloads would execute. All of this results in a non-trivial amount of code that's easy to get wrong.

Needless to say, a more complicated composition pattern would become even more difficult to implement correctly using the thread-based pattern.

Event-Based Pattern

One common pattern for asynchronous programming with the Microsoft .NET Framework is the event-based model. The event model exposes a method to start the asynchronous operation and raises an event when the operation completes.

The event pattern is a convention for exposing asynchronous operations, but it's not an explicit contract, such as via an interface. The class implementer can decide how faithfully to follow the pattern. **Figure 1** shows an example of methods exposed by a correct implementation of the event-based asynchronous programming pattern.

`WebClient` is one class in the .NET Framework that implements asynchronous operations via the event-based pattern. To provide an asynchronous variant of the `DownloadString` method, `WebClient` exposes the `DownloadStringAsync` and `CancelAsync` methods, and the `DownloadStringCompleted` event. This is how our sample would be implemented in an asynchronous way:

```
void Button_Click(object sender, RoutedEventArgs e) {
    WebClient client = new WebClient();
    client.DownloadStringCompleted += eventArgs => {
        HtmlTextBox.Text = eventArgs.Result;
    };
    client.DownloadStringAsync("http://www.microsoft.com");
}
```

This implementation resolves Problem 1 of the inefficient thread-based solution: unnecessary blocking of threads. The call to `Down-`

Figure 2 Examples of the Event-Based Asynchronous Pattern in .NET Classes

Class	Operation
System.Activities.WorkflowInvoker	InvokeAsync
System.ComponentModel.BackgroundWorker	RunWorkerAsync
System.Net.Mail.SmtpClient	SendAsync
System.Net.NetworkInformation.Ping	SendAsync
System.Net.WebClient	DownloadStringAsync

Figure 3 Examples of IAsyncResult in .NET Classes

Class	Operation
System.Action	BeginInvoke
System.IO.Stream	BeginRead
System.Net.Dns	BeginGetHostAddresses
System.Net.HttpWebRequest	BeginGetResponse
System.Net.Sockets.Socket	BeginSend
System.Text.RegularExpressions.MatchEvaluator	BeginInvoke
System.Data.SqlClient.SqlCommand	BeginExecuteReader
System.Web.DefaultHttpHandler	BeginProcessRequest

loadStringAsync returns immediately and does not block either the UI thread or a thread-pool thread. The download executes in the background and once it's finished, the DownloadStringCompleted event will be executed on the appropriate thread.

Note that the DownloadStringCompleted event handler executes on the appropriate thread, without the need for the SynchronizationContext code I needed in the thread-based solution. Behind the scenes, WebClient automatically captures the SynchronizationContext and then posts the callback to the context. Classes that implement the event-based pattern will generally ensure that the Completed handler executes on the appropriate thread.

The event-based asynchronous programming pattern is efficient from the perspective of not blocking more threads than is necessary, and it's one of the two patterns broadly used across the .NET Framework. However, the event-based pattern has several limitations:

- The pattern is informal and by convention only—classes can deviate from it.
- Multiple asynchronous operations can be quite difficult to compose, such as handling asynchronous operations launched in parallel, or handling a sequence of asynchronous operations.
- You cannot poll and check whether the asynchronous operation is done.
- Great care must be taken when utilizing these types. For example, if one instance is used to handle multiple asynchronous operations, a registered event handler must be coded to handle only the one asynchronous operation it's targeting, even if it's invoked multiple times.
- Event handlers will always be invoked on the SynchronizationContext captured when the asynchronous operation was launched, even if executing on the UI thread is unnecessary, leading to additional performance costs.
- It can be difficult to implement well and requires defining multiple types (for example, event handlers or event arguments).

Figure 2 lists several examples of .NET Framework 4 classes that implement the event-based asynchronous pattern.

IAsyncResult Pattern

Another convention for implementing asynchronous operations in .NET is the IAsyncResult pattern. Compared to the event-based model, IAsyncResult is a more advanced solution to asynchronous programming.

In the IAsyncResult pattern, an asynchronous operation is exposed using Begin and End methods. You call the Begin method to initiate the asynchronous operation, and pass in a delegate that will be called when the operation completes. From the callback, you call the End method, which returns the result of the asynchronous operation. Alternatively, instead of providing a callback, you can poll whether the operation has completed, or synchronously wait on it.

As an example, consider the Dns.GetHostAddresses method that accepts a hostname and returns an array of IP addresses that the hostname resolves to. The signature of the synchronous version of the method looks like this:

```
public static IPAddress[] GetHostAddresses(
    string hostNameOrAddress)
The asynchronous version of the method is exposed as follows:
public static IAsyncResult BeginGetHostAddresses(
    string hostNameOrAddress,
    AsyncCallback requestCallback,
    Object state)
```

```
public static IPAddress[] EndGetHostAddresses(
    IAsyncResult asyncResult)
```

Here's an example that uses the BeginGetHostAddresses and EndGetHostAddresses methods to asynchronously query DNS for the address www.microsoft.com:

```
static void Main() {
    Dns.BeginGetHostAddresses(
        "www.microsoft.com",
        result => {
            IPAddress[] addresses = Dns.EndGetHostAddresses(result);
            Console.WriteLine(addresses[0]);
        },
        null);
    Console.ReadKey();
}
```

Figure 3 lists several .NET classes that implement an asynchronous operation using the event-based pattern. By comparing **Figures 2** and **3**, you'll notice that some classes implement the

Figure 4 Using TaskCompletionSource

```
static void Main() {
    // Construct a TaskCompletionSource and get its
    // associated Task
    TaskCompletionSource<int> tcs =
        new TaskCompletionSource<int>();
    Task<int> task = tcs.Task;

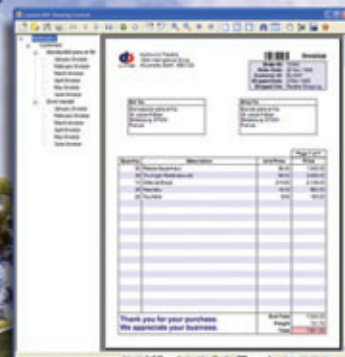
    // Asynchronously, call SetResult on TaskCompletionSource
    ThreadPool.QueueUserWorkItem( _ => {
        Thread.Sleep(1000); // Do something
        tcs.SetResult(123);
    });

    Console.WriteLine(
        "The operation is executing asynchronously...");
    task.Wait();

    // And get the result that was placed into the task by
    // the TaskCompletionSource
    Console.WriteLine("The task computed: {0}", task.Result);
}
```




Dependable Developer Components



DynamicPDF Viewer

Our new, customizable DynamicPDF Viewer allows you to display PDF documents within any WinForm application. No longer rely on an external viewer for displaying your PDF documents. DynamicPDF Viewer utilizes the proven reliable and efficient Foxit PDF viewing engine and maximizes performance and compatibility with our other DynamicPDF products.



DynamicPDF Converter

Our DynamicPDF Converter library can efficiently convert over 30 document types (including HTML and all common Office file formats) to PDF. Events can be used to manage the action taken on a successful or failed conversion. It is highly intuitive and flexible and integrates well with our other DynamicPDF products.



DynamicPDF Rasterizer

Our DynamicPDF Rasterizer library can quickly convert PDF documents to over 10 common image formats including multi-page TIFF. Rasterizing form field values as well as annotations is fully supported. PDFs can also be rasterized to a System.Drawing.Bitmap class for further manipulation.



**Try our three
new products
FREE today!**

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

To learn more about these or any of our other popular tools: **DynamicPDF Generator, DynamicPDF Merger, DynamicPDF ReportWriter, DynamicPDF Suite, DynamicPDF WebCache or Firemail**, visit us online.

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

ceTe software
INFINITE POSSIBILITIES

www.cete.com
info@cete.com

800.631.5006
+1 410.772.8620

Figure 5 Running Operations in Parallel

```
static void Main() {
    string[] urls = new[] { "www.microsoft.com", "www.msdn.com" };
    Task<IPAddress[]>[] tasks = new Task<IPAddress[]>[urls.Length];

    for(int i=0; i<urls.Length; i++) {
        tasks[i] = Task<IPAddress[]>.Factory.FromAsync(
            Dns.BeginGetHostAddresses,
            Dns.EndGetHostAddresses,
            urls[i], null);
    }

    Task.WaitAll(tasks);

    Console.WriteLine(
        "microsoft.com resolves to {0} IP addresses. msdn.com resolves to {1}",
        tasks[0].Result.Length,
        tasks[1].Result.Length);
}
```

event-based pattern, some implement the *IAsyncResult* pattern, and some implement both.

From a historic perspective, the *IAsyncResult* pattern was introduced in the .NET Framework 1.0 as a high-performance approach to implementing asynchronous APIs. However, it requires additional work to interact with the UI thread, it's difficult to implement correctly and it can be difficult to consume. The event-based pattern was introduced in the .NET Framework 2.0 to ease the UI-aspects left unaddressed by *IAsyncResult*, and is focused mostly on scenarios where a UI application launches a single asynchronous application and then works with it.

Task Pattern

A new type, *System.Threading.Tasks.Task*, was introduced in the .NET Framework 4 as a way to represent asynchronous operations. A *Task* can represent an ordinary computation that executes on a CPU:

```
static void Main() {
    Task<double> task = Task.Factory.StartNew(() => {
        double result = 0;
        for (int i = 0; i < 10000000; i++)
            result += Math.Sqrt(i);
        return result;
    });

    Console.WriteLine("The task is running asynchronously...");
    task.Wait();
    Console.WriteLine("The task computed: {0}", task.Result);
}
```

Tasks created using the *StartNew* method correspond to *Tasks* that execute code on the thread pool by default. However, *Tasks* are more general and can represent arbitrary asynchronous operations—even those that correspond to, say, communication with a server or reading data from the disk.

TaskCompletionSource is the general mechanism for creating *Tasks* that represent asynchronous operations. *TaskCompletionSource* is associated with exactly one task. Once the *SetResult* method is called on the *TaskCompletionSource*, the associated *Task* completes, returning the result value of the *Task* (see **Figure 4**).

Here I use a thread-pool thread to call *SetResult* on the *TaskCompletionSource*. However, an important point to notice is that the *SetResult* method could be called by any code that has access to the *TaskCompletionSource*—an event handler for a *Button.Click* event, a *Task* that completed some computation, an event raised because a server responded to a request, and so forth.

So, the *TaskCompletionSource* is a very general mechanism for implementing asynchronous operations.

Converting an *IAsyncResult* Pattern

To use *Tasks* for asynchronous programming, it's important to be able to interoperate with asynchronous operations exposed using the older models. While *TaskCompletionSource* can wrap any asynchronous operation and expose it as a *Task*, the *Task* API provides a convenient mechanism to convert an *IAsyncResult* pattern to a *Task*: the *FromAsync* method.

This example uses the *FromAsync* method to convert the *IAsyncResult*-based asynchronous operation *Dns.BeginGetHostAddresses* into a *Task*:

```
static void Main() {
    Task<IPAddress[]> task =
        Task<IPAddress[]>.Factory.FromAsync(
            Dns.BeginGetHostAddresses,
            Dns.EndGetHostAddresses,
            "http://www.microsoft.com", null);
    ...
}
```

FromAsync makes it easy to convert *IAsyncResult* asynchronous operations to tasks. Under the covers, *FromAsync* is implemented in a manner similar to the example for *TaskCompletionSource* utilizing the *ThreadPool*. Here's a simple approximation of how it's implemented, in this case targeting *GetHostAddresses* directly:

```
static Task<IPAddress[]> GetHostAddressesAsTask(
    string hostNameOrAddress) {
    var tcs = new TaskCompletionSource<IPAddress[]>();
    Dns.BeginGetHostAddresses(hostNameOrAddress, iar => {
        try {
            tcs.SetResult(Dns.EndGetHostAddresses(iar));
        } catch (Exception exc) { tcs.SetException(exc); }
    }, null);
    return tcs.Task;
}
```

Converting an Event-Based Pattern

Event-based asynchronous operations can also be converted to *Tasks* using the *TaskCompletionSource* class. The *Task* class does not provide a built-in mechanism for this conversion—a general

Figure 6 Downloading Strings Asynchronously

```
static void Main() {
    Task<string> page1Task = DownloadStringAsTask(
        new Uri("http://www.microsoft.com"));
    Task<string> page2Task = DownloadStringAsTask(
        new Uri("http://www.msdn.com"));

    Task<int> count1Task =
        page1Task.ContinueWith(t => CountParagraphs(t.Result));
    Task<int> count2Task =
        page2Task.ContinueWith(t => CountParagraphs(t.Result));

    Task.Factory.ContinueWhenAll(
        new[] { count1Task, count2Task },
        tasks => {
            Console.WriteLine(
                "<P> tags on microsoft.com: {0}",
                count1Task.Result);
            Console.WriteLine(
                "<P> tags on msdn.com: {0}",
                count2Task.Result);
        });

    Console.ReadKey();
}
```

We didn't invent the Internet...

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



applications

powered by 

connectivity

powered by 

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

Each day, as you click around the Web or use any connected application, chances are that directly or indirectly some bits are flowing through applications that use our components, on a server, on a device, or right on your desktop. It's your code and our code working together to move data, information, and business. We give you the most robust suite of components for adding Internet Communications, Security, and E-Business Connectivity to

any application, on any platform, anywhere, and you do the rest. Since 1994, we have had one goal: to provide the very best connectivity solutions for our professional developer customers. With more than 100,000 developers worldwide using our software and millions of installations in almost every Fortune 500 and Global 2000 company, our business is to connect business, one application at a time.

To learn more please visit our website →

www.nsoftware.com

mechanism is impractical because the event-based asynchronous pattern is a convention only.

Here's how to convert an event-based asynchronous operation into a task. The code sample shows a method that takes a Uri and returns a Task that represents the asynchronous operation WebClient.DownloadStringAsync:

```
static Task<string> DownloadStringAsTask(Uri address) {
    TaskCompletionSource<string> tcs =
        new TaskCompletionSource<string>();
    WebClient client = new WebClient();
    client.DownloadStringCompleted += (sender, args) => {
        if (args.Error != null) tcs.SetException(args.Error);
        else if (args.Cancelled) tcs.SetCanceled();
        else tcs.SetResult(args.Result);
    };
    client.DownloadStringAsync(address);
    return tcs.Task;
}
```

Using this pattern and the pattern in the previous section, you can convert any existing asynchronous pattern—event-based or IAsyncResult-based—into a Task.

Manipulating and Composing Tasks

So, why would you use Tasks to represent asynchronous operations? The main reason is that Tasks expose methods to conveniently manipulate and compose asynchronous operations. Unlike both the IAsyncResult and event-based approaches, a Task provides a single object that maintains all relevant information about the asynchronous operation, how to join with it, how to retrieve its result and so on.

One useful thing you can do with a Task is to wait until it completes. You can wait on one Task, wait until all Tasks in a set complete, or wait until any Task in a set completes.

```
static void Main() {
    Task<int> task1 = new Task<int>(() => ComputeSomething(0));
    Task<int> task2 = new Task<int>(() => ComputeSomething(1));
    Task<int> task3 = new Task<int>(() => ComputeSomething(2));

    task1.Wait();
    Console.WriteLine("Task 1 is definitely done.");

    Task.WaitAny(task2, task3);
    Console.WriteLine("Task 2 or task 3 is also done.");

    Task.WaitAll(task1, task2, task3);
    Console.WriteLine("All tasks are done.");
}
```

Another useful capability of Tasks is the ability to schedule continuations: Tasks that execute as soon as another Task completes. Similar to waiting, you can schedule continuations that run when a particular Task completes, when all Tasks in a set complete or when any Task in a set completes.

This example creates a task that will query DNS for the address www.microsoft.com. Once that task completes, the continuation task is kicked-off and will print the result to the console:

```
static void Main() {
    Task<IPAddress[]> task =
        Task<IPAddress[]>.Factory.FromAsync(
            Dns.BeginGetHostAddresses,
            Dns.EndGetHostAddresses,
            "www.microsoft.com", null);

    task.ContinueWith(t => Console.WriteLine(t.Result));
    Console.ReadKey();
}
```

Let's take a look at more interesting examples that show off the power of the task as a representation of an asynchronous operation. **Figure 5** shows an example that runs two DNS lookups in

parallel. When the asynchronous operations are represented as tasks, it's easy to wait until multiple operations have completed.

Let's take a look at another example of composing tasks that takes the following three steps:

1. Asynchronously download multiple HTML pages in parallel
2. Process the HTML pages
3. Aggregate the information from the HTML pages

Figure 6 shows how such computation would be implemented, by taking advantage of the DownloadStringAsTask method shown earlier in this article. One notable benefit of this implementation is that the two different CountParagraphs methods execute on different threads. Given the prevalence of multi-core machines today, a program that spreads its computationally expensive work across multiple threads will get a performance benefit.

Running Tasks in a Synchronization Context

Sometimes it's useful to be able to schedule a continuation that will run in a particular synchronization context. For example, in applications with a UI, it's often useful to be able to schedule a continuation that will execute on the UI thread.

The easiest way to have a Task interact with a synchronization context is to create a TaskScheduler that captures the context of the current thread. To get a TaskScheduler for the UI thread, invoke the FromCurrentSynchronizationContext static method on the TaskScheduler type while running on the UI thread.

This example asynchronously downloads the www.microsoft.com Web page and then assigns the downloaded HTML into the Text property of a WPF text box:

```
void Button_Click(object sender, RoutedEventArgs e) {
    TaskScheduler uiTaskScheduler =
        TaskScheduler.FromCurrentSynchronizationContext()

    DownloadStringAsTask(new Uri("http://www.microsoft.com"))
        .ContinueWith(
            t => { textBox1.Text = t.Result; },
            uiTaskScheduler);
}
```

The body of the Button_Click method will set up the asynchronous computation that eventually updates the UI, but Button_Click does not wait until the computation completes. That way, the UI thread will not be blocked, and can continue updating the user interface and responding to user actions.

As I mentioned previously, prior to the .NET Framework 4, asynchronous operations were typically exposed using either the IAsyncResult pattern or the event-based pattern. With the .NET Framework 4, you can now employ the Task class as another useful representation of asynchronous operations. When represented as tasks, asynchronous operations are often easier to manipulate and compose. More examples on using tasks for asynchronous programming are included in the ParallelExtensionsExtras samples, available for download at code.msdn.microsoft.com/ParExtSamples. ■

IGOR OSTROVSKY is a software development engineer on the Parallel Computing Platform team at Microsoft. Ostrovsky documents his adventures in programming at igoro.com and contributes to the Parallel Programming with .NET blog at blogs.msdn.com/pfxteam.

THANKS to the following technical experts for reviewing this article:
Concurrency Runtime team

GET THE FASTEST CONTROLS...



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

Infragistics
KILLER APPS. NO EXCUSES.

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

Throttling Concurrency in the CLR 4.0 ThreadPool

Erika Fuentes

The CLR ThreadPool in the latest release (CLR 4.0) has seen several major changes since CLR 2.0. The recent shift in technology trends, such as the widespread use of manycore architectures and the resulting desire to parallelize existing applications or write new parallel code, has been one of the biggest motivating factors in the improvement of the CLR ThreadPool.

In the December 2008 *MSDN Magazine* CLR Inside Out column, “Thread Management in the CLR” (msdn.microsoft.com/magazine/dd252943), I discussed some of the motivations and associated issues such as concurrency control and noise. Now I’ll describe how we’ve addressed these in the CLR 4.0 ThreadPool, the associated implementation choices and how these can impact its behavior. Also, I’ll focus on the approach taken toward automating concurrency control in the current CLR 4.0 ThreadPool (hereafter referred to only as ThreadPool for convenience). I will also give a brief outline of the ThreadPool architecture. This article covers implementation details subject to change in future versions. However, those readers designing and writing new concurrent applications who are interested in improving old applications by taking advantage of concurrency, or making use of ASP.NET or Parallel Extension technologies (all in the context of CLR 4.0), may find this material useful to understand and take advantage of current ThreadPool behavior.

Overview of the ThreadPool

A thread pool is meant to provide key services such as thread management, abstractions for different types of concurrency and

throttling of concurrent operations. By providing these services, a thread pool takes away some burden from the user to do it manually. For the inexperienced user, it’s convenient not having to learn and deal with the details of a multi-threaded environment. For the more experienced user, having a reliable threading system means that she can focus on improving different aspects of the application. The ThreadPool provides these services for managed applications and support for portability across platforms, running certain Microsoft .NET Framework applications on the Mac OS, for example.

There are different types of concurrency that can be related to various parts of the system. The most relevant are: CPU parallelism, I/O parallelism; timers and synchronization; and load balancing and resource utilization. We can briefly outline the architecture of the ThreadPool in terms of the different aspects of concurrency (for more details on the ThreadPool architecture and related APIs usage, see “The CLR’s Thread Pool” (msdn.microsoft.com/magazine/cc164139)). Specifically, it’s worth mentioning that there are two independent implementations of the ThreadPool: one deals with CPU parallelism and is referred to as the *worker ThreadPool*; the other deals with I/O Parallelism and can be dubbed *I/O ThreadPool*. The next section will focus on CPU parallelism and the associated implementation work in the ThreadPool—in particular, on strategies for throttling concurrency.

The Worker ThreadPool Designed to provide services at the level of CPU parallelism, the worker ThreadPool takes advantage of multi-core architectures. There are two main considerations for CPU parallelism: dispatching work quickly and optimally; and throttling the degree of parallelism. For the former, the ThreadPool implementation makes use of strategies such as lock-free queues to avoid contention and work-stealing for load balancing, areas that are out of the scope of this discussion (see msdn.microsoft.com/magazine/cc163340 for more insight on these topics). The latter—throttling the degree of parallelism—entails concurrency control to prevent resource contention from slowing down overall throughput.

CPU parallelism can be particularly challenging because it involves many parameters, such as determining how many work items can be run simultaneously at any given time. Another

This article discusses:

- The CLR 4.0 ThreadPool
- Concurrency in the ThreadPool
- Methodologies for throttling concurrency
- Signal processing to reduce noise

Technologies discussed:

CLR 4.0



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



Word processing components for Visual Studio .NET.

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

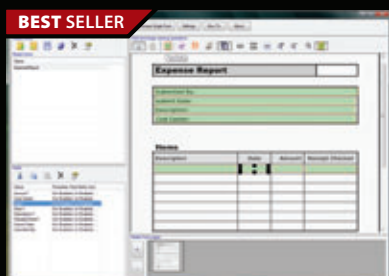


FusionCharts | from \$195.02



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

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

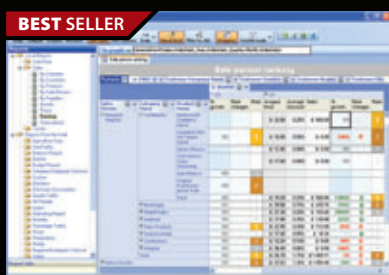


LEADTOOLS Recognition SDK | from \$3,595.50



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

- Features accurate, high-speed multi-threaded OCR and forms recognition
- Supports text, ICR, OMR, image, and 1D/2D barcode fields
- Auto-registration and clean-up to improve recognition results
- Includes .NET, C/C++, WPF, WF, WCF and Silverlight interfaces
- Latin character set support is included. Arabic and Asian support is available.



ContourCube | from \$900.00



OLAP component for interactive reporting and data analysis.

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

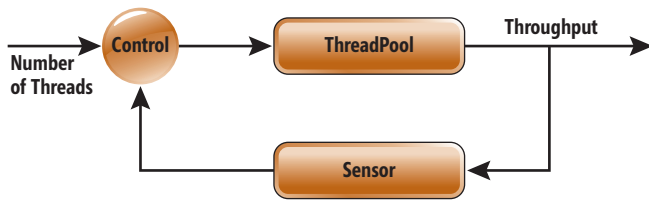


Figure 1 ThreadPool Feedback Loop

problem is the number of cores and how to tune for different types of workloads. For example, having one thread per CPU is optimal (in theory), but if the workload is constantly blocking, then CPU time is wasted because more threads could be used to execute more work. The size and type of workload is actually another parameter. For example, in the case of blocking workloads, it's extremely difficult to determine the number of threads that optimizes overall throughput because it's hard to determine when a request will be completed (or perhaps even how often it will arrive—this is closely related to I/O blocking). The API associated to this ThreadPool is `QueueUserWorkItem`, which queues a method (the work item) for execution (see msdn.microsoft.com/library/system.threading.threadpool.queueuserworkitem). It's recommended for applications that have work that could potentially be run in parallel (with other things). The work is handed to the ThreadPool, which automatically “figures out” when to run it. This facility takes away from the programmer the need to worry about how and when to create threads; however, it isn't the most efficient solution for all scenarios.

The I/O ThreadPool This part of the ThreadPool implementation, related to I/O parallelism, handles blocking workloads (that is, I/O requests that take a relatively long time to service) or asynchronous I/O. In asynchronous calls, threads aren't blocked and can continue to do other work while the request is serviced. This ThreadPool takes care of the coordination between the requests and the threads. The I/O ThreadPool—like the worker ThreadPool—has an algorithm for throttling concurrency; it manages the number of threads based on the completion rate of asynchronous operations. But this algorithm is quite different from the one in the worker ThreadPool and it's out of the scope of this document.

Concurrency in the ThreadPool

Dealing with concurrency is a difficult but necessary task that directly impacts the overall performance of a system. How the system throttles concurrency directly impacts other tasks such as synchronization, resource utilization and load balancing (and vice versa).

The concept of “concurrency control,” or more appropriately, “throttling concurrency,” refers to the number of threads that are allowed to do work at a particular time in the ThreadPool; it's a policy to decide how many threads can be run simultaneously without hurting performance. Concurrency control in our discussion is only in regard to the worker ThreadPool. As opposed to what may be intuitive, concurrency control is about *throttling* and *reducing* the number of work items that can be run in parallel in order to improve the worker ThreadPool throughput (that is, controlling the degree of concurrency is preventing work from running).

The concurrency control algorithm in the ThreadPool automatically chooses the level of concurrency; it decides for the user how

many threads are necessary to keep the performance generally optimal. The implementation of this algorithm is one of the most complex and interesting parts of the ThreadPool. There are various approaches to optimize the performance of the ThreadPool in the context of concurrency level (in other words, determine the “right” number of threads running at the same time). In the next section, I will discuss some of those approaches that have been considered or used in the CLR.

The Evolution of Concurrency Control in the ThreadPool

One of the first approaches taken was to optimize based on the observed CPU utilization, and *adding* threads to maximize it—running as much work as possible to keep the CPU busy. Using CPU utilization as a metric is useful when dealing with long or variable workloads. However, this approach wasn't appropriate because the criteria to evaluate the metric can be misleading. Consider, for example, an application where lots of memory paging is happening. The observed CPU utilization would be low, and adding more threads in such a situation would result in more memory being used, which consequently results in even lower CPU utilization. Another problem with this approach is that in scenarios where there's a lot of contention, the CPU time is really being spent doing synchronization, not doing actual work, so adding more threads would just make the situation worse.

Another idea was to just let the OS take care of the concurrency level. In fact, this is what the I/O ThreadPool does, but the worker ThreadPool has required a higher level of abstraction to provide more portability and more efficient resource management. This approach may work for some scenarios, but the programmer still needs to know how to do throttling to avoid over saturation of resources (for example, if thousands of threads are created, resource contention can become a problem and adding threads will actually make things worse). Furthermore, this implies that the programmer still has to worry about concurrency, which defeats the purpose of having a thread pool.

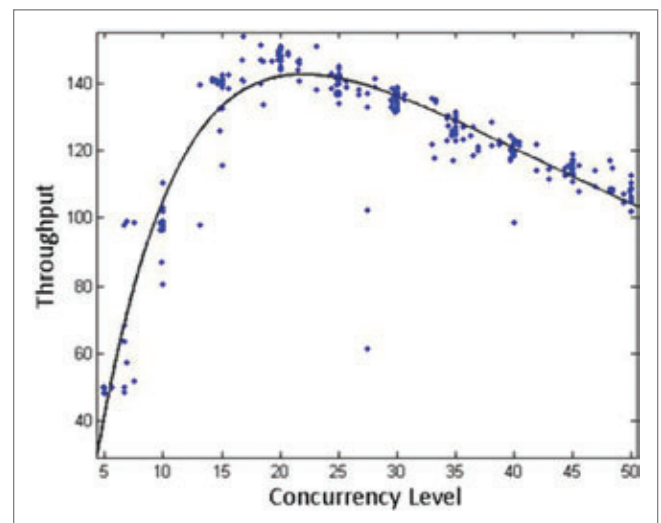


Figure 2 Throughput Modeled as a Function of Concurrency Level

A more recent approach was to include the concept of throughput, measured as completion of work items per unit of time, as a metric used to tune performance. In this case, when the CPU utilization is low, threads are added to see if this improves the throughput. If it does, more threads are added; if it didn't help, threads are removed. This approach is more sensible than previous ones because it takes into account how much work is being completed, rather than just how the resources are being used. Unfortunately, throughput is affected by many factors other than just the number of active threads (for example, work item size), which makes it challenging to tune.

Control Theory for Throttling Concurrency

To overcome some of the limitations of previous implementations, new ideas were introduced with CLR 4.0. The first methodology considered, from the control theory area, was the Hill Climbing (HC) algorithm. This technique is an auto-tuning approach based on an input-output feedback loop. The system output is monitored and measured at small time intervals to see what effects the controlled input had, and that information is fed back into the algorithm to further tune the input. Looking at the input and output as variables, the system is modeled as a function in terms of these variables. The goal is then to optimize the measured output.

In the context of the worker ThreadPool system, the input is the number of threads that are executing work concurrently (or concurrency level), and the output is the throughput (see **Figure 1**).

We observe and measure over time the changes in throughput as a result of adding or removing threads, then decide whether to add or remove more threads based on the observed throughput degradation or improvement. **Figure 2** illustrates the idea.

Having throughput as a (polynomial) function of the concurrency level, the algorithm adds threads until the maximum of the function is reached (about 20 in this example). At that point, a decrease in throughput will be seen and the algorithm will remove threads. Over each time interval t_i a sample of throughput measurements is taken and "averaged." This is then used to make a decision for the next time interval t_{i+1} . It's understandable that if the measurements are noisy, statistical information isn't representative of the actual situation, unless perhaps it's taken over a large interval of time. It's hard to tell whether an improvement was a result of the change in concurrency level or due to another factor such as workload fluctuations.

Adaptive approaches in real-life systems are complicated; in our case its use was particularly problematic because of the difficulty in detecting small variations or extracting changes from a very noisy environment over a short time. The first problem observed with this approach is that the modeled function (see the black trend in **Figure 2**) isn't a static target in real-world situations (see blue dots also in the graph), so measuring small changes is hard. The next issue, perhaps more concerning, is that noise (variations in measurements caused by the system environment, such as certain OS activity, garbage collection and more) makes it difficult to tell if there's a relationship between the input and the output, that is, to tell if the

JINX

CAN YOU FIND THE BUG?

Ever spent months looking for threading bugs?

Until now, no single tool could help you find them.

Jinx augments the bug detection power of Visual Studio and uncovers hard-to-find threading bugs in your software.

Download Jinx today from Corensic.
www.corensic.com

corensic

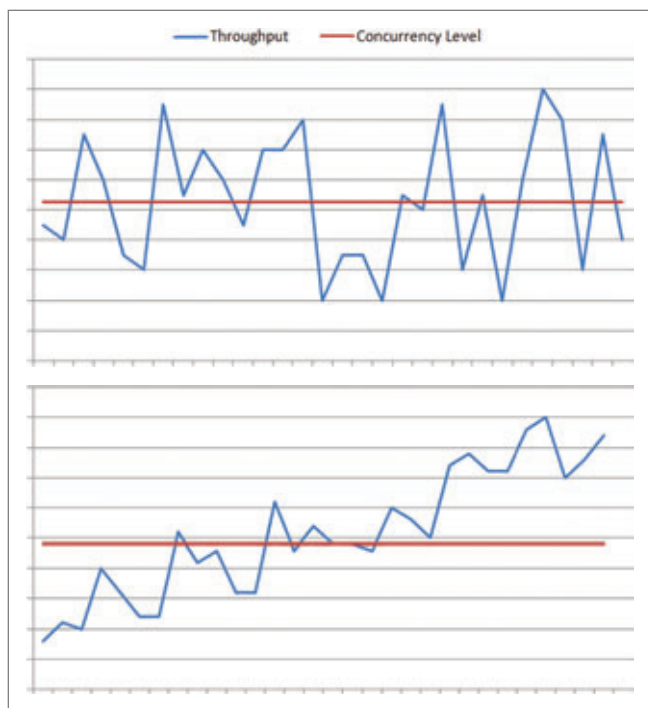


Figure 3 Example of Noise in the ThreadPool

throughput isn't just a function of the number of threads. In fact, in the ThreadPool, the throughput constitutes only a small part of what the real observed output is—most of it is noise. For example, take an application whose workload uses many threads. Adding just a few more won't make a difference in the output; an improvement observed in a time interval may not even be related to the change in concurrency level (Figure 3 helps to illustrate this issue).

In Figure 3, on the x-axis is time; on the y-axis, throughput and concurrency level measurements are over-imposed. The top graph illustrates that in some workloads, even if the number of threads (red) is kept constant, there may be observed changes in the throughput (blue). In this example, the fluctuations are noise. The bottom graph is another example where the general increase in the throughput is observed over time even in the presence of noise. However, the number of threads was kept constant so the improvement in throughput is due to a different parameter in the system.

In the next section, I'll discuss an approach to dealing with the noise.

Bringing in Signal Processing

Signal processing is used in many areas of engineering to reduce noise in signals; the idea is to find the input signal's pattern in the output signal. This theory can be applied in the context of the ThreadPool if we treat the input (concurrency level) and output (throughput) of the concurrency control algorithm as signals. If we input a purposely modified concurrency level as a "wave" with known period and amplitude, and look for that original wave pattern in the output, we can discern what is noise from the actual effect of input on the throughput. Figure 4 illustrates this idea.

Consider, for a moment, the system as a black box that generates output given an input. Figure 4 shows a simplified example of the HC input and output (green); below that is an example of how the input and output would look as waves using a filtering technique (black).

Instead of feeding a flat, constant input, we introduce a signal and then try to find it in the noisy output. This effect can be achieved by using techniques such as the *band pass filter* or *match filter*, generally used for extracting waves from other waves or finding very specific signals in the output. This also means that by introducing changes to the input, the algorithm is making decisions at every point based on the last small piece of input data.

The particular algorithm used in the ThreadPool uses a discrete Fourier transform, a methodology that gives information such as the magnitude and phase of a wave. This information can then be used to see if and how the input affected the output. The graph in Figure 5 shows an example of the ThreadPool behavior using this methodology on a workload running more than 600 seconds.

In the Figure 5 example, the known pattern of the input wave (phase, frequency and amplitude) can be traced in the output. The chart illustrates the behavior of the concurrency algorithm using filtering on a sample workload. The red trend corresponds to the input and the blue corresponds to the output. We vary the thread count up and down over time, but this doesn't mean we're creating or destroying threads; we keep them around.

Although the scale of the number of threads is different from that of the throughput, we can see how it's possible to map how the input affects the output. The number of threads is constantly changed by at least one in a time slice, but this doesn't mean that a thread is being created or destroyed. Instead, threads are kept "alive" in the pool, but they aren't actively doing work.

As opposed to the initial approach using HC, where the goal was to model the throughput curve and base the decision on that calculation, the improved methodology just determines whether or not a change in the input helped to improve the output. Intuitively, there is increased confidence that the changes that we artificially introduce are having the effect observed on the output (the maximum number of threads observed so far to be introduced in the signal is 20, which is quite reasonable—especially for scenarios that have many threads). One of the drawbacks of the approach that uses signal processing is that due to the artificial wave pattern introduced, the optimal concurrency level will always be off by at least one thread. Also, adjustments to the concurrency level happen relatively slowly (faster algorithms are based on CPU utilization metrics) because it's necessary to gather enough data to make the model stable. And the speed will depend on the length of work items.

This approach isn't perfect and works better for some workloads than for others; however, it's considerably better than previous

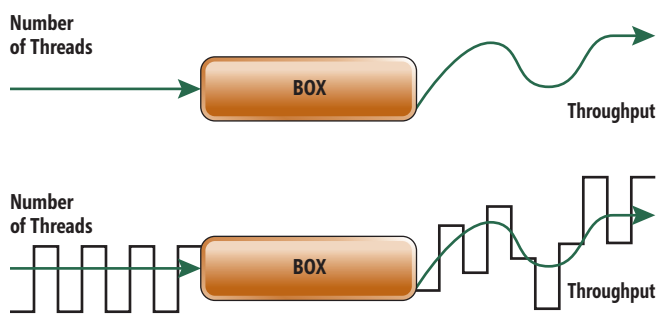


Figure 4 Determining Factors in ThreadPool Output

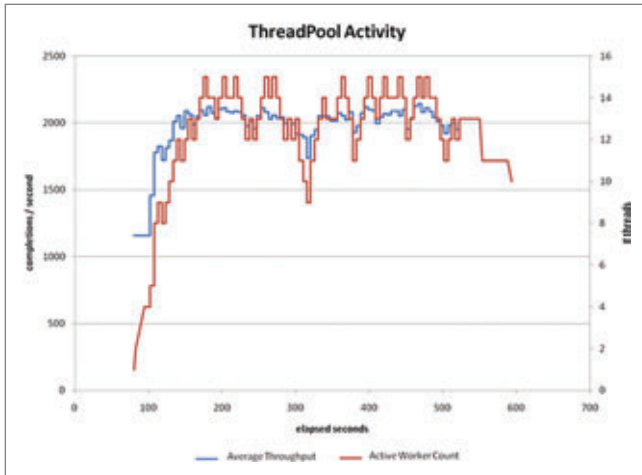


Figure 5 Measuring How Input Affects Output

methodologies. The types of workloads for which our algorithm works the best are those with relatively short individual work items, because the shorter the work item, the faster the algorithm is allowed to adapt. For example, it works pretty well with work item durations less than 250ms, but it's better when the durations are less than 10ms.

Concurrency Management—We'll Do It for You

Wrapping up, the ThreadPool provides services that help the programmer focus on things other than concurrency management. In

order to deliver such functionality, the ThreadPool implementation has incorporated high-end engineering algorithms that can automate many decisions for the user. One example is the concurrency control algorithm, which has evolved based on the technology and the needs expressed from different scenarios, such as a need to measure useful progress of work execution.

The purpose of the concurrency control algorithm in CLR 4.0 is to automatically decide how many work items can be run concurrently in an efficient manner, hence optimizing the throughput of the ThreadPool. This algorithm is difficult to tune because of noise and parameters such as the type of workload; it also depends on the assumption that every work item is a useful piece of work. The current design and behavior has been heavily influenced by ASP.NET and Parallel Framework scenarios, for which it has good performance. In general, the ThreadPool can do a good job executing the work efficiently. However, the user should be aware that there may be unexpected behavior for some workloads, or if, for example, there are multiple ThreadPools running at the same time. ■

ERIKA FUENTES, PH.D., is a software development engineer in Test on the CLR team, where she works in the Performance Team with particular focus on the Core Operating System area in Threading. She has written several academic publications about scientific computing, adaptive systems and statistics.

THANKS to the following technical experts for reviewing this article:
Eric Eilebrecht and Mohamed Abd El Aziz



Today Is Your Lucky Day.

Get Microsoft SharePoint® 2010
BCS Functionality in WSS 3.0 or Higher!



Enterprise Enabler®

Agile Integration Software



*Microsoft and SharePoint are registered trade-marks of Microsoft Corporation.

The first 100 buyers get it for
ONLY \$1,495

Free 30 Day Trial

, unlimited users, money back guarantee - what are you waiting for?

Plus check out our **Monthly Contests** under the Developer Zone tab.

www.enterpriseenabler.com

Actor-Based Programming with the Asynchronous Agents Library

Michael Chu and Krishnan Varadarajan

With multi-core processors now commonplace in the market, from servers to desktops to laptops, the parallelization of code has never been more important. To address this vital area, Visual Studio 2010 introduces several new ways to help C++ developers take advantage of these capabilities with a new parallel runtime and new parallel programming models. However, one main hurdle left for developers is deciding which programming model is correct for their applications. The correct model may significantly exploit underlying parallelism, but may also require a rethink of how your program is structured and actually executes.

The most common parallel programming models today involve general-purpose, concurrency-aware containers, and algorithms such as parallelizing loop iterations. While these traditional techniques can be a powerful method for scaling applications to take advantage of a

multi-core machine, they don't address one of the other major factors affecting parallel performance: the growing impact of latency. As parallelization techniques speed up computations and spread them out across multiple cores, Amdahl's law (wikipedia.org/wiki/Amdahl's_law) shows us that the performance improvement is limited by the slowest portion of the execution. In many cases, there's an increasing percentage of time spent waiting on data from I/O such as disks or networks.

Actor-based programming models deal quite well with problems such as latency and were first introduced in the early 1970s to exploit the resources of highly parallel computers with hundreds or thousands of independent processors. The fundamental concept behind an actor model is to treat the components of an application as individual actors that can interact with the world by sending, receiving and processing messages.

More recently, with the abundance of multi-core processors, the actor model has resurfaced as an effective method to hide latencies for efficient parallel execution. Visual Studio 2010 introduces the Asynchronous Agents Library (AAL), an exciting new actor-based model with message-passing interfaces where the agents are the actors. AAL enables developers to design their applications in a more dataflow-centric manner. Such a design typically makes for productive use of latency while waiting for data.

In this article, we'll provide an overview of the AAL and demonstrate how you can take advantage of it in your applications.

This article discusses:

- The Concurrency Runtime
- Message passing
- Message blocks
- Asynchronous agents

Technologies discussed:

Visual Studio 2010, Asynchronous Agents Library

The Concurrency Runtime

The foundation for concurrency support in Visual Studio 2010 and AAL is the new Concurrency Runtime, which is shipped as part of the C Runtime (CRT) in Visual Studio 2010. The Concurrency Runtime offers a cooperative task scheduler and a resource manager that has a deep understanding of the underlying resources of the machine. This allows the runtime to execute tasks in a load-balanced fashion across a multi-core machine.

Figure 1 shows an outline of the support in Visual Studio 2010 for concurrency in native code. The Scheduler is the main component that determines when and where tasks execute. It leverages information gathered by the Resource Manager to best utilize the execution resources. Applications and libraries themselves mainly interact with the Concurrency Runtime through the two programming models that sit on top of the scheduler, the AAL and the Parallel Patterns Library (PPL), although they can also directly interact with the runtime itself.

The PPL offers the more traditional parallelization techniques, such as `parallel_for` and `parallel_for_each` constructs, runtime-aware locks, and concurrent data structures such as queues and vectors. While not the focus of this article, the PPL is a powerful tool for developers that can be used in conjunction with all the new methods introduced in the AAL. For more information on the PPL, see the February 2009 installment of the Windows With C++ column (msdn.microsoft.com/magazine/dd434652).

In contrast, the AAL provides the ability to parallelize applications at a higher level and from a different perspective than traditional techniques. Developers need to think about applications from the perspective of the data to be processed, and consider how the processing of the data can be separated into components or stages that can execute in parallel.

The AAL provides two main components: a message-passing framework and asynchronous agents.

The message-passing framework includes a set of message blocks, which can receive, process and propagate messages. By chaining together message blocks, pipelines of work can be created that can execute simultaneously.

Asynchronous agents are the *actors* that interact with the world by receiving messages, performing local work on their own maintained state, and sending messages.

Together, these two components allow developers to exploit parallelism in terms of the flow of data rather than the flow of control, and to better tolerate latencies by utilizing parallel resources more efficiently.

Message-Passing Framework

The first important component of the AAL is the message-passing framework, a set of constructs to help develop dataflow networks to pipeline work. Pipelining work is a fundamental piece of the

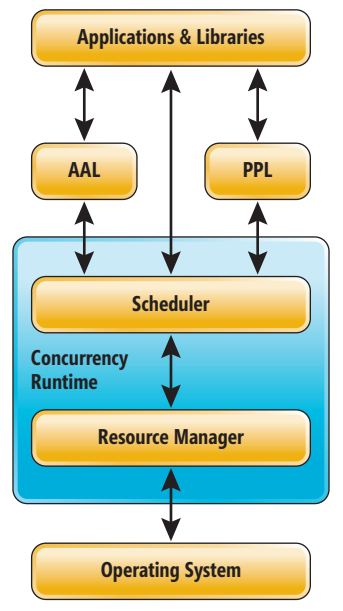


Figure 1 The Concurrency Runtime

dataflow model, as it allows streaming data to be processed in parallel whenever the data is ready by breaking up the work into multiple independent stages. When the processing of data in one stage finishes, that stage can pass the data off to the next stage while the first looks for new data on which to work.

As an example, consider an e-mail application that formats outgoing messages and censors them for inappropriate content. The code for this type of operation is shown here:

```
std::foreach(reader.begin(), reader.end();  
[](const string& word) {  
    auto w1 = censor(word);  
    auto w2 = format(w1);  
    writer.write_word(w2);  
});
```

For each word in the e-mail, the application needs to check if it exists in a dictionary of censored words, replacing it if it does. The code then formats each word according to a set of guidelines.

There's a significant amount of inherent parallelism within such a scenario. However,

traditional techniques for parallelism fall short. For example, a simple approach would be to use a `parallel_for_each` algorithm across the strings in the text to censor, then format them.

The first main deterrent to such a solution is that it must read the entire file so that an iterator can properly divide up the work. Forcing the entire file to be read makes the process I/O-bound and can diminish parallelization gains. Of course, you could use a smart iterator to overlap processing of words with reading the input.

Actor-based programming models deal quite well with problems such as latency.

The second major issue with a traditional parallelization approach is ordering. Obviously, in the case of an e-mail message, parallel processing of the text must maintain the order of the text or the meaning of the message is totally lost. To maintain the ordering of the text, a `parallel_for_each` technique would incur significant overhead in terms of synchronization and buffering, which is automatically handled by the AAL.

By processing the message in a pipeline, you can avoid these two issues while still taking advantage of parallelization. Consider

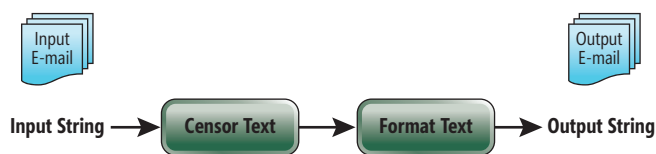


Figure 2 E-mail Processing Pipeline

Figure 3 AAL Message Blocks

Message Block	Purpose
unbounded_buffer<Type>	Stores an unbounded number of messages and propagates them to its targets.
overwrite_buffer<Type>	Stores a single message, which will be overwritten each time a new message is propagated to it, and broadcasts it to its targets.
single_assignment<Type>	Stores a single message, which is write-once, and broadcasts it to its targets.
transformer<Input,Output>	Takes a message of type Input and runs a user-provided function to transform it to a message of type Output. This transformed message is propagated to its targets.
call<Type>	Takes a message and runs a user-provided function with that message's payload as an argument. This is purely a message target.
timer<Type>	Propagates a message to its target after a user-defined amount of time. This can be repeating or non-repeating. This block is purely a message source.
choice<Type1,Type2,...>	Takes messages from multiple sources of multiple types and will only accept the message from the first block that propagated to the choice.
join<Type>	Takes messages from multiple sources and combines them together to output a single message. Asynchronously waits for messages to be ready from each source input.
multitype_join<Type1,Type2,...>	Takes messages from multiple sources of multiple types and combines them together. Asynchronously waits for messages to be ready from each source input.

Figure 2, where a simple pipeline was created. In this example, the main tasks of the application—censoring and formatting—are separated into two stages. The first stage takes a string and looks it up in a dictionary of censored words. If a match is found, the censor block substitutes the string with a different word from the dictionary. Otherwise, it outputs the same message that was inputted. Similarly, in the second stage, the format block takes in each word and properly formats it for a certain style.

The first important component of the AAL is the message-passing framework.

This example can benefit from the dataflow approach in several ways. First, because it removes the requirement to read the entire message before processing, the strings in the message can immediately start streaming through the censoring and formatting stages. Second, the pipeline processing allows one string to be processed by the format block while the next string is being processed by the censor block. Finally, because strings are processed in the order they appear in the original text, no additional synchronization needs to be done.

Message Blocks

The messages blocks receive, process, store and propagate messages. Message blocks come in one of three forms: sources, targets, and propagators. Sources only have the ability to propagate messages, while targets can receive, store and process them. The majority of blocks are propagators, which are both sources and targets. In other words, they have the ability to receive, store and process messages, as well as to turn around and send these messages out.

The AAL contains a set of message block primitives that cover the majority of use cases for developers. Figure 3 shows a brief overview of all message blocks included in the AAL. However, the model remains open, so if your application requires a message block with a specific behavior, you can write a custom block yourself that can interact with all the predefined blocks. Each block has its own unique characteristics for processing, storing and propagating messages.

One of the main benefits of the message block primitives supplied by the AAL is that they're composable. Therefore, you can combine them, based on the desired behavior. For example, you can easily create a block that adds together multiple inputs by attaching a transformer block to the end of a join block. When the join block succeeds in retrieving messages from each of its sources, it can pass them to the transformer, which sums the message payloads.

You could also connect a repeating timer block as a source of a join block. This would result in a block that throttles messages, only letting them through whenever the timer block fires its message. These two composable blocks are illustrated in Figure 4.

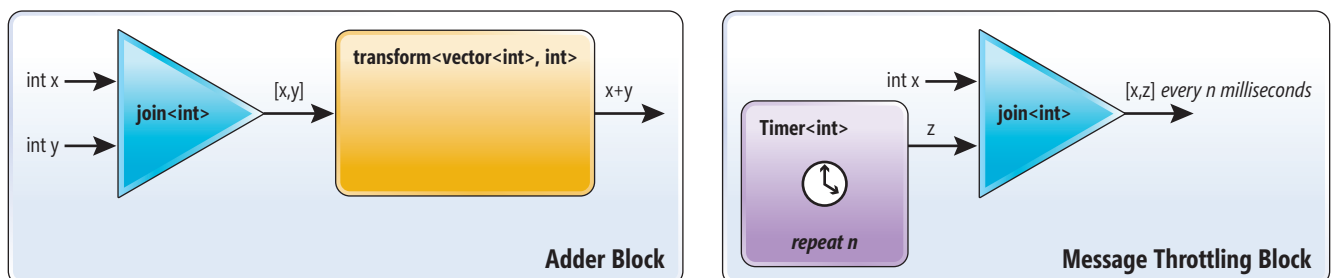


Figure 4 Composing Adder and Message Throttling Blocks from Primitives

Imagine...

...an **intranet** employees want to use

Why is **user adoption** such a large hurdle for intranets?

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

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

Learn more:



eIntranet[™]

Built on Ektron

<http://www.ektron.com/intranet>

ektron

What do you **want**
your **website** to do?

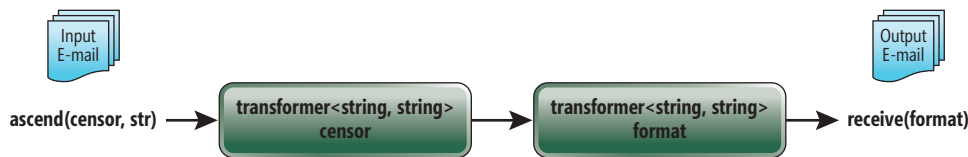


Figure 5 A Message Block Pipeline

Creating a Message-Passing Pipeline

Now let's take a look at the code to create the message-block pipeline shown earlier. We can replace the pipeline with two transformer message blocks, as shown in **Figure 5**. The purpose of a transformer block is to take a message of a certain type and execute a user-defined function on that message, which can modify the message's payload or even completely change the type of the message. For example, the censor block takes as input a message containing a string and needs to process it.

A message is simply an
envelope that wraps the data
that you want to pass around
your dataflow network.

The code for creating and connecting the message blocks is shown in **Figure 6**. This code begins with the instantiation of the two transformer message blocks. The C++0x lambda parameter on the censor block constructor defines the transformation function, which looks up the message's stored input string in a dictionary to see if it should be changed to a different string. The resulting string is returned, and within the censor block it's then wrapped in a message and propagated out of the block. A similar path is taken for the format transformer block, except its output is a string that has been changed by a format function.

Following the instantiation of the two blocks, the next line links the two blocks together by calling the `link_target` method on the censor block. Every source and propagator block has a `link_target` method that's used to determine to which message blocks the source should propagate its messages.

After the censor and format blocks have been linked together, any message propagated into the censor block will go through its transform function and the resulting message will implicitly be passed on to the format block for processing. If a message block is a source or propagator yet has no connected targets, the message block can store the message in a block-specific manner until either a target is linked, or the message is retrieved.

The last three lines of the example code show the process of initiating messages into a block and retrieving a message out of a block. There are two message initiation APIs in the AAL: `send` and `asend`. These input a message into a block synchronously and asynchronously, respectively.

The main difference is that when a `send` call returns, it's guaranteed to have already pushed its message into and through the block to which the message is being sent. The `asend` call can return immediately and will allow the Concurrency Runtime to schedule

its propagation. Similarly, there are two message retrieval APIs in the AAL: `receive` and `try_receive`. The `receive` method will block until a message arrives, whereas the `try_receive` will return immediately if it's unable to retrieve a message.

In **Figure 6**, the string "foo" is sent in asynchronously to the censor block. The censor block will take the message, check if its string is in the dictionary of censored words, and then propagate the resulting string in a message. This will then be passed to the format block, which will take the string, capitalize each letter, and because it has no targets, hold on to the message. When `receive` is called, it will grab the message from the format block. Thus, assuming "foo" was not in the dictionary, the output of this example would be "FOO." While this example only pushes a single string through the network, you can see how a stream of input strings forms a pipeline of execution.

Looking at this messaging example, notice the distinct lack of references to messages themselves. A message is simply an envelope that wraps the data you want to pass around your dataflow network. The message passing itself is handled through a process of offering and accepting. When a message block receives a message, it has the ability to store that message in any way it wants. If it later wishes to send a message out, it offers the message to each of its connected targets. To actually take the message away, the receiver must accept the offered message to complete the

Figure 6 Simple Message Pipeline

```

dictionary dict;

transformer<string, string>
censor([&dict](const string& s) -> string {

    string result = s;
    auto iter = dict.find(s);

    if (iter != dict.end()) {
        result = iter->second;
    }

    return result;
});

transformer<string, string>
format([&](const string& s) -> string {

    string result = s;
    for (string::size_type i = 0; i < s.size(); i++) {
        result[i] = (char)Format(s[i]);
    }

    return result;
});

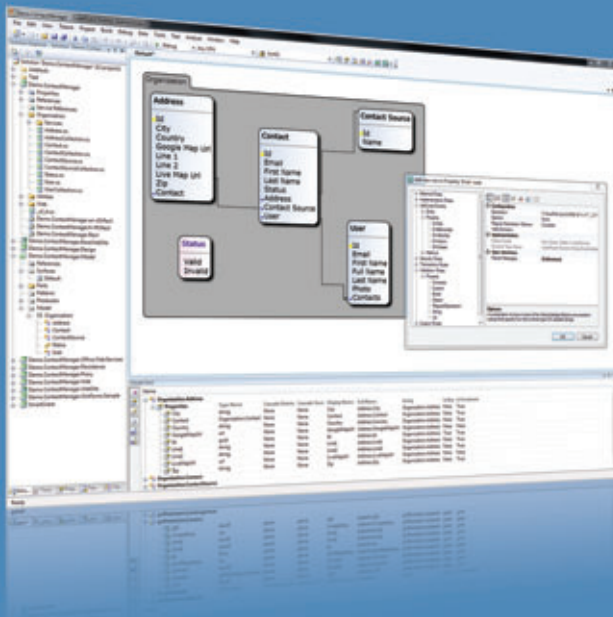
censor.link_target(&format);

asend(&censor, "foo");
string newStr = receive(format);
printf("%s\n", newStr);
  
```




Need a **HAND?**

Use **CODEFLUENT ENTITIES!**



APPLICATION BLOCKS

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

SUPPORTED ARCHITECTURES

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

FEATURED TECHNOLOGIES

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

CUSTOMER APPROVED MODEL-DRIVEN SOFTWARE FACTORY

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

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

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

Download your **FREE** trial today at:

www.CodeFluentEntities.com/Msdn

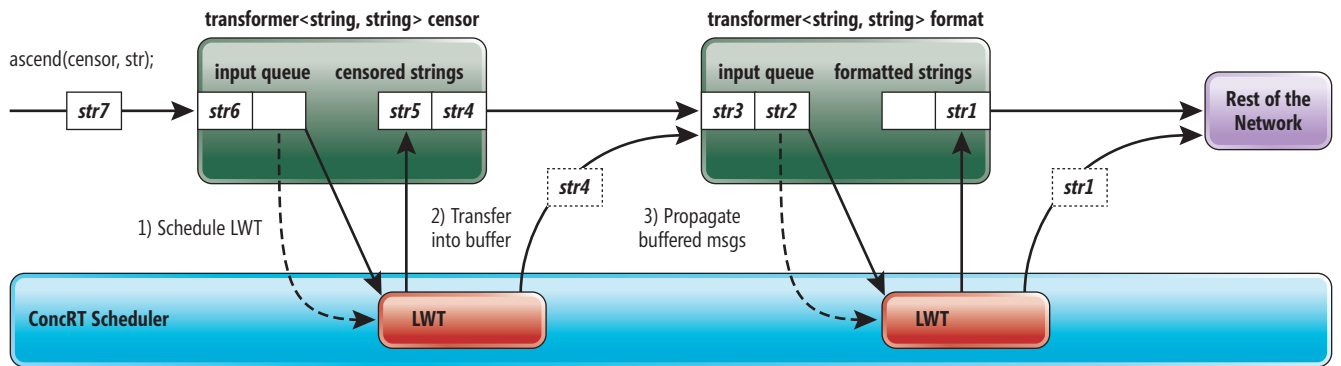


Figure 7 Message-Passing Protocol

transaction. This entire process of message passing between blocks is scheduled and handled by tasks that are scheduled and executed by the Concurrency Runtime.

Message-Block Propagation

Now that you've seen how message blocks are created and tied together, and how messages can be initiated and retrieved from each of them, let's take a brief look at how messages are passed between blocks and how the Concurrency Runtime fits at the heart of the AAL.

This information is not necessary for using message blocks or the AAL, but can help give a deeper understanding of how the message-passing protocols work and how you can take advantage of them. For the rest of this section, I'll discuss propagator blocks, because they're both sources and targets. Obviously, a pure source or pure target block would simply be a subset of the propagator block implementation.

Internally, each propagator block has an input queue for messages and another block-specific storage container for messages. Other blocks that are linked to this propagator block send messages that are stored into the input queue.

Messages can buffer up in the input queue while they await processing.

For example, in **Figure 7**, the censor transformer block has an input queue that's currently storing a message with a string `str6` in it. The actual transformer itself contains two messages: `str4` and `str5`. Because this is a transformer, its block-specific storage is another queue. Different block types can have different storage containers. For example, the `overwrite_buffer` block only stores a single message that would always get overwritten.

When a message is presented to a block from one of its linked sources (or the `send/ascend` APIs), the block first checks a filter function to determine whether or not to accept the message. If it decides to accept the message, the message is placed into the input queue. A filter is an optional function that can be passed into

the constructor of each target or propagator block that returns a Boolean that determines whether a message offered from a source should be accepted. If the message is declined, the source will continue to its next target to offer the message.

Once a message is placed in the input queue, the source block it came from no longer holds on to the message. However, the accepting block does not yet have the message ready for propagation. Thus, messages can buffer up in the input queue while they await processing.

When a message arrives in an input queue on a message block, a lightweight task (LWT) is scheduled within the Concurrency Runtime scheduler. The purpose of this LWT is twofold. First, it must move messages from the input queue to the internal storage of the block (which we refer to as message processing). Second, it must also try to propagate messages to any targets (which we refer to as message propagation).

For example, in **Figure 7**, there were messages in the input queue that prompted the LWT to be scheduled. The LWT then processed the message by first executing the transformer's user-provided function

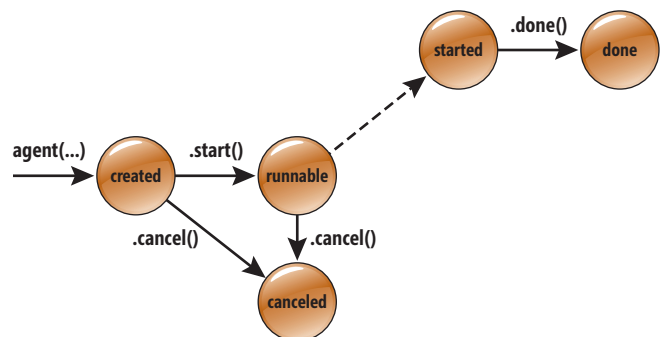


Figure 8 The Asynchronous Agent Lifecycle

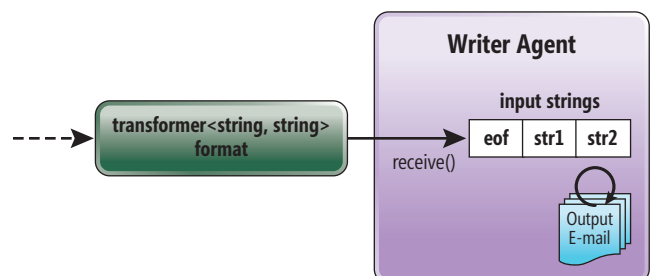
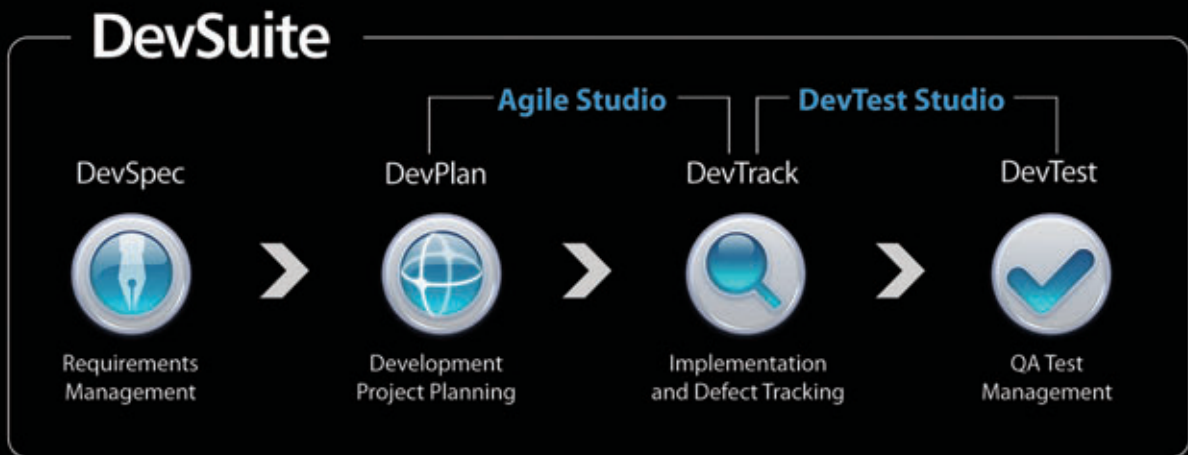


Figure 9 An Agent Capturing the Output of the Format Block

Gain Full Traceability

Requirements Driven Quality Management



A Modular Approach to ALM

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

Agile Ready

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



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

www.techexcel.com

1-800-439-7782

Figure 10 WriterAgent

```
class WriterAgent : public agent {
public:
    WriterAgent(ISource<string> * src) : m_source(src) {
    }

    ~WriterAgent() {
        agent::wait(this);
    }

    virtual void run() {
        FILE *stream;
        fopen_s( &stream, ... );

        string s;
        string eof("EOF");

        while (!feof(stream) && ((s=receive(m_source)) != eof)) {
            write_string(stream, s);
        }

        fclose(stream);
        done();
    }

private:
    ISource<string> * m_source;
};
```

on the message, checking it in the censored string dictionary, then moving the message to the storage buffer for the block.

After transferring it into a storage buffer, the LWT begins the propagation step where messages are sent to the target format block. In this case, because message str4 was at the head of the transformer, it's propagated to the format block first, and then the next message, str5, is propagated. The same entire process occurs on the format block.

Message processing can differ, depending on the type of message block. For example, an unbounded_buffer had a simple processing step of moving a message to its storage buffer. The transformer processes messages by calling its user-defined function on the message before moving it to a storage buffer. Other blocks can become even more complex, such as the join, which must combine multiple messages from different sources and store them to a buffer in preparation for propagation.

For performance efficiency, the AAL is intelligent in its creation of LWTs so that only one is scheduled at a time for each message block. If further messages arrive in an input queue while the processing LWT is active, it will continue to pick up and process those messages. Thus, in **Figure 7**, if the transformer's LWT is still processing when message str7 enters the input queue, it will pick up and process this message rather than starting a new processing and propagation task.

The fact that each message block has its own LWT that handles processing and propagation is central to the design, which allows the message-passing framework to pipeline work in a dataflow manner. Because each message block does its processing and propagation of its messages in its own LWT,

the AAL is able to decouple the blocks from one another and allow parallel work to be executed across multiple blocks. Each LWT must simply propagate its messages into its target blocks' input queues, and each target will simply schedule an LWT to handle its own inputs. Using a single LWT to process and propagate ensures that message ordering is maintained for the message blocks.

Asynchronous Agents

The second main component of the AAL is the asynchronous agent. Asynchronous agents are coarse-grained application components that are meant to asynchronously deal with larger computing tasks and I/O. Agents are expected to communicate with other agents and initiate lower-level parallelism. They're isolated because their view of the world is entirely contained within their class, and they can communicate with other application components by using message passing. Agents themselves are scheduled as tasks within the Concurrency Runtime. This allows them to block and yield cooperatively with other work executing at the same time.

An asynchronous agent has a set lifecycle, as shown in **Figure 8**. The lifecycle can be monitored and waited on. States in green signify running states, while states in red are the terminal states. Developers can create their own agents by deriving from the base agent class.

Agents are expected
to communicate with other
agents and initiate
lower-level parallelism.

Three base class functions—start, cancel and done—transition the agent between its different states. Once constructed, agents are in the created state. Starting an agent is similar to starting a thread. They will not execute anything until the start method is called on them. At that time, the agent is scheduled to execute and the agent moves into the runnable state.

When the Concurrency Runtime picks up the agent, it moves into the started state and continues to run until the user calls the done method, indicating its work has completed. Any time after the agent has been scheduled but not yet started, a call to cancel will transition the agent to a canceled state and it will never execute.

Let's look back at the e-mail filtering example, where the pipelined message blocks introduced dataflow into the application and improved

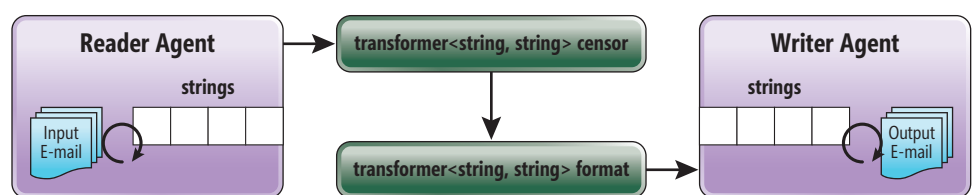
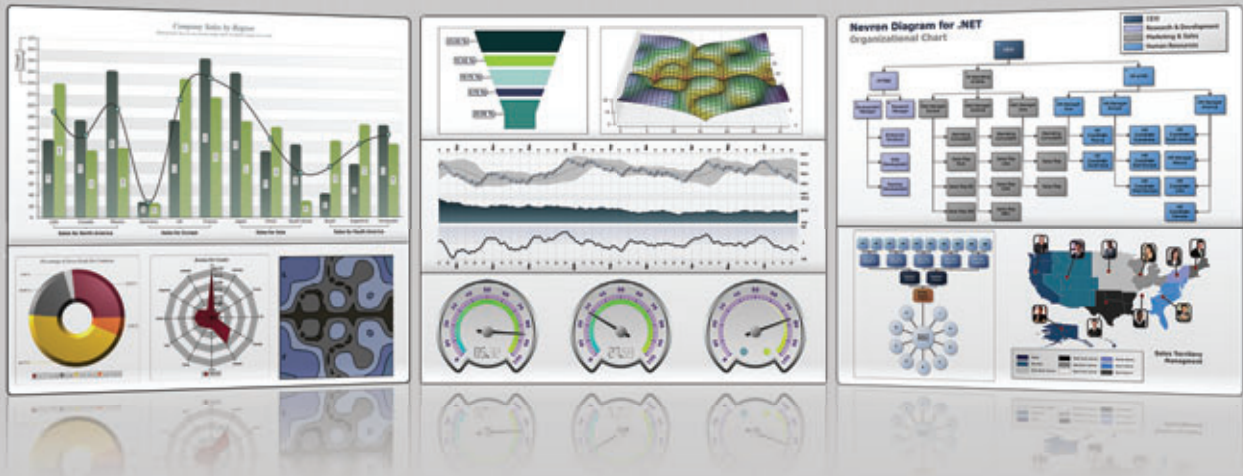


Figure 11 Agents Used to Process E-mail Messages

Let us help you visualize your success





Nevron provides the essential components for the creation of advanced digital dashboards, scientific and financial applications, diagrams and MMI interfaces for a variety of .NET centric technologies.



Nevron components integrate seamlessly in web and desktop applications, SQL Server Reporting Services 2005/2008 reports and SharePoint 2007/2010 portals and deliver an unmatched set of enterprise-grade features. That is why Nevron is the trusted vendor by many Fortune 500 companies for their most demanding data visualization needs.



Developers

Nevron .NET Vision incorporates components that help you create enterprise grade digital dashboards, scorecards, diagrams, maps, MMI interfaces and much more.



-  Chart for .NET
-  Diagram for .NET
-  Gauge for .NET
-  Map for .NET
-  User Interface for .NET

IT Professionals

Nevron Reporting Services Vision instantly enhances your SQL Server Reporting Services 2005/2008 reports with the industry leading data visualization technology.

-  Chart for SSRS
-  Gauge for SSRS

Nevron SharePoint Vision instantly converts your SharePoint pages into advanced dashboards and reports, that unite powerful data analysis with industry leading data visualization.

-  Chart for SharePoint
-  Gauge for SharePoint

MAKE SURE THAT YOUR DATA IS MAKING THE VISUAL STATEMENT IT DESERVES BY DOWNLOADING YOUR FREE EVALUATION COPY FROM [WWW.NEVRON.COM](http://www.nevron.com) TODAY.



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

its ability to parallel process words. However, the example did not show how to handle the I/O of dealing with the e-mails themselves and breaking them into streams of strings for the pipeline to process. Also, once the strings have been passed through the pipeline, the strings must be gathered so that the text can be rewritten in its newly censored and formatted state. This is where agents can come into play in order to help tolerate the differences in latencies with I/O.

For example, consider the end of our e-mail pipeline. At this point, strings are being outputted by the format and need to be written to files in a mailbox. **Figure 9** shows how an output agent can capture strings and create output e-mail messages. The run function of the `WriterAgent` receives messages from the format block in a loop.

While the majority of the processing done in this application is using dataflow, the `WriterAgent` shows how some control-flow can be introduced into the program. For example, when an end-of-file message arrives, the `WriterAgent` must have different behavior depending on the input string being received; it must know to cease operation. The code for the `WriterAgent` is in **Figure 10**.

There are a few interesting portions of this code to note. First, within the destructor, a call is made to a static function `agent::wait`. This function can be called with a pointer to any agent and will block until the agent enters one of the terminal states: `done` or `canceled`. While calling `wait` in the destructor is not necessary for all agents, in most cases it should be done, as it ensures the agent is no longer executing any code when destructing.

Second, an interesting aspect of this code is the run method itself. This method defines the main execution of the agent. In this code, the agent is dealing with writing out the strings it reads from its source (in our example, the format block).

Finally, note the last line of the run method, which is a call to the agent function `done`. The call to the `done` method moves the agent from the running state to the `done` state. In most cases, this will need to be called at the end of the run method. However, in some circumstances, applications may want to use agents to set up state,

such as in a dataflow network, which should remain alive past the lifetime of the run method.

Tying Everything Together

Now that we've created a messaging pipeline to filter and format strings, and an output agent to process them, we can add an input agent that has very similar behavior to the output agent. **Figure 11** shows an example of how this application fits together.

One of the benefits of agent processing is the ability to use asynchronous actors in the application. Thus, when data arrives for processing, the input agent will asynchronously start sending the strings through the pipeline and the output agent can likewise read and output files. These actors can start and stop processing entirely independently and totally driven by data. Such behavior works beautifully in many scenarios, especially latency-driven and asynchronous I/O, like the e-mail processing example.

In this example, I added a second agent, a `ReaderAgent`, which acts similarly to the `WriterAgent`, except it handles the I/O to deal with reading the e-mails and sending strings to the network. The code for the `ReaderAgent` is in **Figure 12**.

Now that we have both a `ReaderAgent` and a `WriterAgent` to asynchronously handle the I/O for the program, we simply need to link them up to the transformer blocks in the network to begin processing. This can be done easily after linking the two blocks together:

```
censor.link_target(&format);
```

```
ReaderAgent r(&censor);
r.start();
```

```
WriterAgent w(&format);
w.start();
```

The `ReaderAgent` is created with a reference to the `censor` so it can properly send messages to it, while the `WriterAgent` is created with a reference to the `format` so it can retrieve messages. Each agent is started with its `start` API, which schedules the agents for execution within the Concurrency Runtime. Because each agent calls the `agent::wait(this)` in its own destructor, the execution will wait until both agents have reached their `done` state.

Figure 12 `ReaderAgent`

```
class ReaderAgent : public agent {
public:
    ReaderAgent(ITarget<string> * target) : m_target(target) {
    }

    ~ReaderAgent() {
        agent::wait(this);
    }

    virtual void run() {
        FILE *stream;
        fopen_s(&stream, ...);

        while (!feof(stream)) {
            asend(m_target, read_word(stream));
        }

        fclose(stream);

        asend(m_target, string("eof"));
        done();
    }

private:
    ITarget<string> * m_target;
};
```

Syncing Up

This article was written to give you a glimpse into some of the new possibilities for actor-based programming and dataflow pipelining built into Visual Studio 2010. We encourage you to try it out.

If you want to dig deeper, there are plenty of other features we weren't able to cover in this article: custom message block creation, filtering messages, and much more. The Parallel Computing developer center on MSDN (msdn.microsoft.com/concurrency) contains more details and walkthroughs of how this exciting new programming model can help you parallelize your program in entirely new ways. ■

MICHAEL CHU is a software development engineer in the Parallel Computing Platform group at Microsoft. He works on the Concurrency Runtime team.

KRISHNAN VARADARAJAN is a software development engineer in the Parallel Computing Platform group at Microsoft. He works on the Concurrency Runtime team.

THANKS to the following technical experts for reviewing this article: Concurrency Runtime team

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

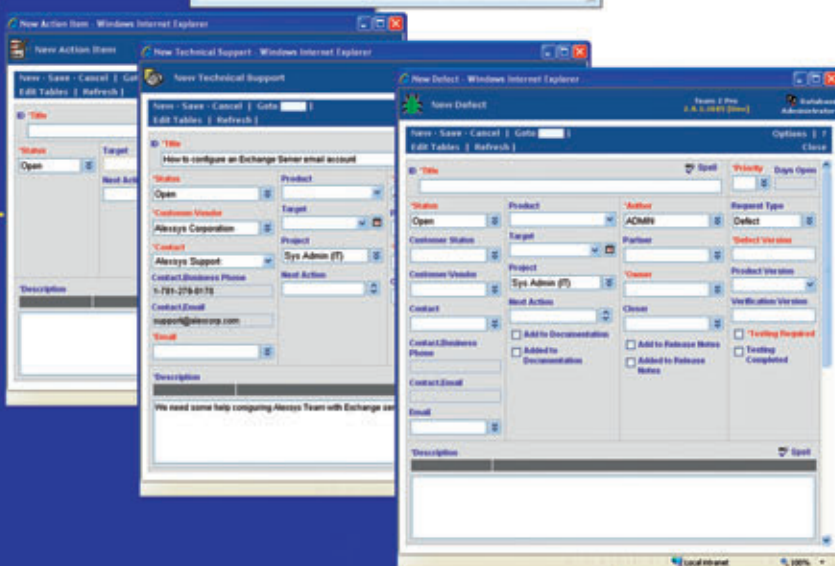
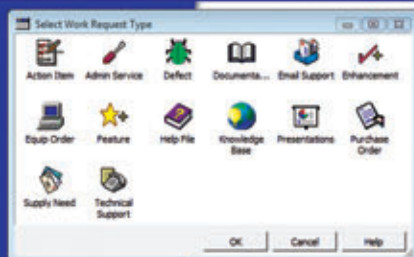
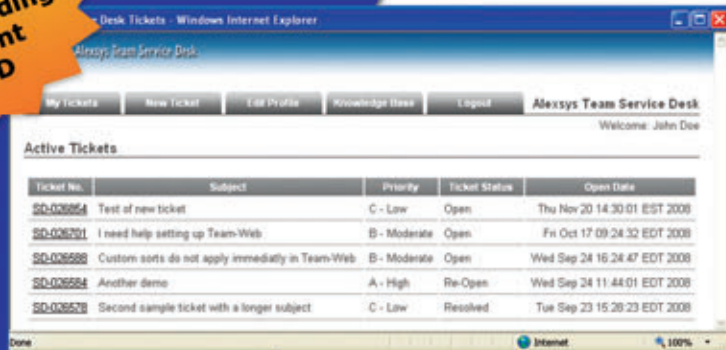
- Full Windows 7 Support
- Windows Single Sign-on
- System Audit Log
- Trend Analysis
- Alternate Display Fields for Data Normalization
- Lookup Table Filters
- XML Export
- Network Optimized for Enterprise Deployment

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



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.

Migrate Your ASP.NET 1.1 Apps to Visual Studio 2010

Jonathan Waldman

If you've been working with ASP.NET for more than a few years, chances are you've written solutions using Visual Studio 2003 for the Microsoft .NET Framework 1.1. In more recent years, newer, feature-rich .NET Framework 2.0, 3.0 and 3.5 versions have debuted—and you may have wondered whether it would be worthwhile or feasible to upgrade your trusty 1.1-Framework apps to one of them.

Today, as the new .NET Framework 4 is being embraced by developers around the globe, you might feel more compelled than ever to seriously consider a migration effort. If you decide to do so, rest assured that Microsoft has provided useful tools to facilitate such an undertaking, resulting in modernized ASP.NET apps that can take advantage of the latest .NET Framework innovations. Specifically, I'll show you how to reinvigorate ASP.NET 1.1 solutions by migrating them to Visual Studio 2010 (Professional, Premium or Ultimate)—enabling them to target the .NET Framework 2.0, 3.0, 3.5 or 4 versions.

This article discusses:

- Why you should migrate
- Issues with ASP.NET project types
- Main steps in a migration
- Post-migration considerations

Technologies discussed:

ASP.NET, .NET Framework, Visual Studio 2010, XHTML

Why Migrate?

Even if your goal is to simply maintain your existing ASP.NET 1.1 sites, your support and extensibility options are dwindling. As a 1.1 developer, you should be concerned that Microsoft retired Mainstream Support for Visual Studio 2003 on Oct. 14, 2008—and has stated that it will issue no further service packs for Visual Studio 2003 or for the .NET Framework 1.1. (Microsoft will, however, offer Extended Support for Visual Studio 2003 until Oct. 8, 2013.)

You should be downright alarmed, however, that a growing number of third-party vendors have ceased offering or supporting components that can run against the .NET Framework 1.1 or that extend the Visual Studio 2003 IDE. Indeed, your .NET Framework 1.1 Web apps and the Visual Studio 2003 IDE are being alienated and are being eased into oblivion.

Yet other reasons abound. There have been so many enhancements to the .NET Framework, the C# and Visual Basic .NET programming languages, and to the Visual Studio IDE that migrating your .NET 1.1 sites to .NET 2.0 or beyond (hereafter referred to as 2.0+) would finally enable you to enhance them using the latest tools and technologies, as well as all the modern language and framework features (such as master pages, AJAX, Windows Communication Foundation [WCF], Silverlight, LINQ, partial classes, generics, lambda expressions and anonymous methods) already embraced by leading .NET developers. In general, the higher the framework version you choose your ASP.NET app to target, the greater potential and power you will have at your

fingertips. You would also find solid support not only from Microsoft itself (as of this writing, Microsoft offers Mainstream Support for all 2.0+ framework versions), but also within active developer forums and by third-party tool vendors. There are also widely available books covering all aspects of the latest technologies—and many sites offering video-based courses, blogs, white papers and other training options.

The latest, state-of-the-art Visual Studio 2010 IDE is almost itself a justification for migration. It's generations ahead of the Visual Studio 2003 IDE, supports new features such as IntelliTrace and can be extended using a variety of powerful third-party plug-ins that virtually guarantee new levels of programming productivity. While some rare users will need to continue using older versions of Visual Studio for various reasons, Visual Studio 2010 makes running side-by-side installations of Visual Studio 2005 and Visual Studio 2008 virtually unnecessary because Visual Studio 2010 can target the 2.0+ frameworks. Yet, like Visual Studio 2005 and Visual Studio 2008, Visual Studio 2010 *can't* target the 1.1 framework. This means you either have to continue to run Visual Studio 2003 or you have to migrate your ASP.NET 1.1 project to a newer framework.

Finally, developers with 1.1-framework-only experience simply aren't as marketable as those with 2.0+ framework experience. Competition for developer positions is high, so you want to give yourself every possible edge; leveraging newer framework features in your code will enhance your candidacy and credibility as a professional software developer.

Migration Issues

So why hasn't everyone already migrated up from ASP.NET 1.1 during any one of the major Visual Studio and .NET Framework releases since Visual Studio 2003? One reason is that developers would need to implement the ASP.NET 1.1-to-ASP.NET 2.0 migration option Visual Studio 2005 offered. That doesn't sound so bad, but realize that when Visual Studio 2005 was first rolled out, it required that an ASP.NET 1.1 project be converted to a newfangled Web-project type. This meant that the very architecture on which an ASP.NET 1.1 site was built had to be painstakingly reworked and rigorously regression-tested. Many developers considered this process tantamount to rewriting the site from scratch, presenting a cost and technical challenge deemed insurmountable. As a result, decision makers often left ASP.NET 1.1 sites as they were.

To better appreciate this ASP.NET 1.1-to-ASP.NET 2.0 migration challenge, it's important to understand that all ASP.NET 1.1 Web projects are organized using a file-folder configuration that is *project-driven*. This project type was originally dubbed the "2003 Web project model" and is now known as a Web Application Project (WAP). When Visual Studio 2005 was released, Microsoft introduced a new *folder-driven* project type that was designed for the .NET Framework 2.0. This project type was originally dubbed the "2005 Web project model" and is now known as a Web Site Project (WSP). (Confused? Use the simple trick of remembering

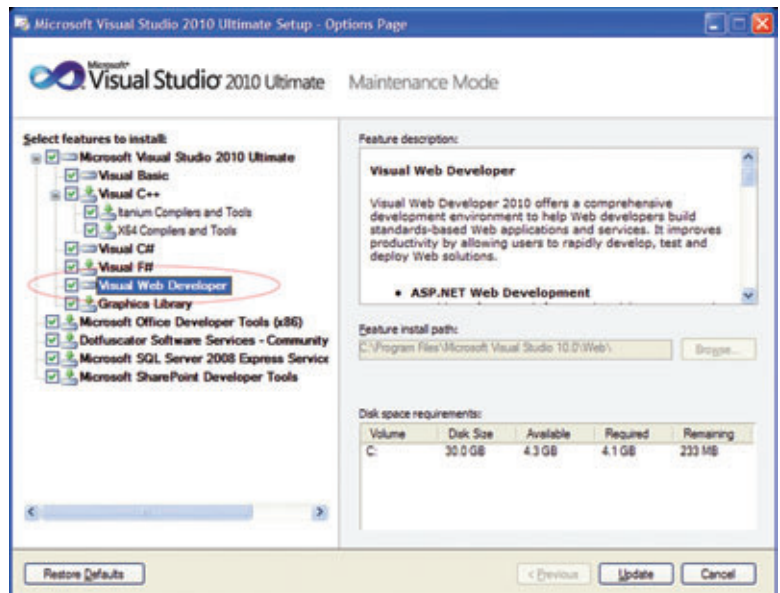


Figure 1 Ensure the Visual Web Developer Component Is Installed Before Launching the Conversion Wizard

the acronyms in their alphabetical order: WAP comes before WSP just as Visual Studio 2003 comes before Visual Studio 2005.)

While an in-depth discussion of WAPs and WSPs is out of scope here, suffice it to say that Microsoft originally intended all migrated 1.1 Web applications to become WSPs via its included Conversion Wizard. Concerned developers immediately protested because the original WAP project type offered specific technical and performance advantages absent in the newer WSP project type. An ASP.NET development community furor quickly ensued as developers, managers and stakeholders learned that Microsoft had provided only an ASP.NET 1.1 WAP-to-ASP.NET 2.0 WSP migration option—and that this proved to be time-consuming and costly, especially for sites of any complexity. Detailing the technical issues that arise during such a migration would constitute a separate multi-part article; you can learn more by reading the MSDN library article, "Common Web Project Conversion Issues and Solutions" (msdn.microsoft.com/library/aa479312).

Fortunately, Microsoft responded quickly and soon offered a revised Conversion Wizard in Visual Studio 2005 SP1. This time, the Conversion Wizard would convert an ASP.NET 1.1 WAP to an ASP.NET 2.0 WAP. Yet many developers missed learning about this migration option and thus never explored it. In Visual Studio 2010, the Conversion Wizard converts an ASP.NET 1.1 WAP to an ASP.NET 2.0+ WAP (2.0+ refers to the fact that, once converted, you can target 2.0, 3.0, 3.5 or 4), still providing an opportunity to migrate older Web apps. (You can't convert an ASP.NET 1.1 app to a WSP app—nor would you likely want to—using the Visual Studio 2010 Conversion Wizard; you would need to have access to the Conversion Wizard provided in the original Visual Studio 2005 that was released before SP1.)

It might interest you to know that whenever you start a new project in Visual Studio 2010, you can choose whether you want to create a WAP or a WSP. Microsoft has promised to offer WAP and WSP support in current and all future .NET Framework releases.



Figure 2 The Visual Studio Conversion Wizard Introductory Dialog

The Process

Migrating your ASP.NET 1.1 apps using Visual Studio 2010 consists of the following broad steps (I'll discuss each in detail):

- Run the Conversion Wizard.
- Set the target framework and startup page.
- Compile and fix.
- Convert pages and user controls to partial classes.
- Compile and fix.

Run the Conversion Wizard To use the Conversion Wizard for a 1.1 WAP-to-2.0+ WAP conversion (our goal), you must install the Visual Web Developer option during the Visual Studio 2010 setup. If you didn't, you need to add it by re-running Visual Studio 2010 Setup and using its Change or Remove Microsoft Visual Studio 2010 option; then choose Add or Remove Features; then ensure that the Visual Web Developer feature is checked (see **Figure 1**).

If Visual Web Developer isn't installed, the Conversion Wizard still launches and runs. It will be able to convert all projects in your solution except for your Web project. It will advise you that you need to install Visual Web Developer in order to convert your Web project.

Before you undertake a conversion effort, it just makes good sense to ensure that your application not only builds and runs, but also that it's as clean and organized as possible. Therefore:

1. Clean your application by removing any unused or unneeded files, thereby reducing your Web application to its essential components.
2. Ensure every project in your solution compiles with no compile-time errors.
3. While I recommend adding your solution to source control if it isn't already so configured, do verify that no files are checked out exclusively from the repository. Having your solution under source control will let you examine specific changes made by the wizard after the conversion process has completed.
4. Ensure that no files or folders outside of source control are marked as read-only. This lets the wizard update those files/folders as needed.

5. Create a backup. If you run your solution in a virtual machine, it's best to simply back up the entire virtual machine (or create a snapshot of it). Otherwise, back up your entire application—including all dependencies. If you're not sure about the dependencies, a good place to look is in your application's solution (.sln) file as well as in the individual project (.proj) files (these are text files that can be visually inspected). Many ASP.NET solutions contain projects that point to network locations or to folders outside the Web root or application folders. Not backing up these dependencies can cause you tremendous grief should you wish to restore your application from the backup files. Keep in mind that the full migration process makes changes to every project in your application, so it's advantageous to understand your application's dependencies before getting started. In addition, if you're using projects that are shared among several ASP.NET 1.1 applications, realize that once these solutions have been migrated, they can no longer be opened in Visual Studio

2003. If you work in a team environment, making changes to shared projects will also affect team members' solutions, which could end up referencing converted projects. Thus, whether shared across projects or teams, you should decouple shared projects by making copies of them and ensuring that a local copy exists on your development workstation. Your solution file will change if you move a project (in order to decouple it, for example), so you might want to think about backing up the solution file before making changes to any of the projects it references—and then back up again after the project depen-

Before you undertake a conversion effort, it just makes good sense to ensure that your application not only builds and runs, but also that it's as clean and organized as possible.

dencies have been properly decoupled but before running the Conversion Wizard. Finally, you may discover errors in your original solution files after the conversion process completes; having a handy backup of your pre-converted solution will make it easy for you to go back and fix those errors before re-running the Conversion Wizard.

6. Consider setting up a parallel environment: Visual Studio 2003 and Visual Studio 2010 can run side-by-side on the same machine, so you can run your ASP.NET 1.1 site alongside your ASP.NET 2.0+ site. This will facilitate your QA-testing efforts post-migration.

INSPIRE! EMPOWER! IMPRESS!



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

Infragistics
KILLER APPS. NO EXCUSES.

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

To launch the wizard, open your Visual Studio 2003 solution file using the File, Open, Project/Solution menu option in Visual Studio 2010. You will soon see the Visual Studio Conversion Wizard introductory dialog (see **Figure 2**). Click Next.

The Conversion Wizard may identify issues while it processes and you may find yourself wanting to restore your solution from a backup, make a few changes and start the process again from scratch. This step makes it a snap to create a backup (see **Figure 3**) even though it places all of your solution's dependencies within the backup folder you specify, which means that to restore properly, you have to be absolutely certain you know the original arrangement of those backed-up folders.

Given that, to back up your solution at this step, simply supply a folder path and the wizard will create a child folder called Backup into which it will place the solution's files. Every project in the solution will be backed up to a corresponding subfolder under the Backup folder—even if it originates outside the original solution's root—so for the backup to be successful, it's important to ensure that project names are unique across your solution. Click Next.

You will now see a final dialog box containing an advisory regarding source control and read/write permissions (see **Figure 4**).

If you chose to do a backup, you will see the type of backup you requested and to what location the backup files will be written. You will see a summary of the solution name being converted along with all of its projects. If everything looks accurate and acceptable, click Finish.

Now you may be prompted for logon credentials for your source control repository (such as Visual SourceSafe) so that your project can be checked out.

You will now see a progress bar as each of your solution's projects is converted. The process may take several minutes as the wizard iterates across the projects in your solution.

The Conversion Wizard displays a message explaining that you've completed the first step and that your Web application now needs to be converted (see **Figure 5**).

It then alerts you that it has completed its initial process (see **Figure 6**).

You can then review the conversion log using the Conversion Report template. This file, called Upgrade-Log.XML, is placed in the same folder as your solution's .sln file. The upgrade log shows which project files were converted and displays any relevant errors and warning messages. It also provides useful comments as well as comments regarding the framework version that each project targets. I have included an excerpt from this report in **Figure 7**.

All warnings and errors should be reviewed. Warnings (such as a change to the relative path of a backup path) generally concern minor technical concerns that don't typically require further action in order for you to be able to successfully compile your converted

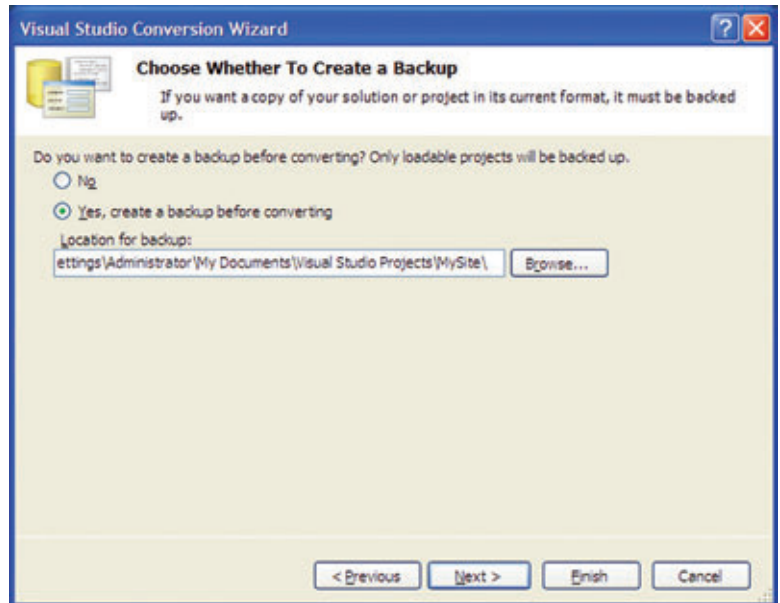


Figure 3 The Conversion Wizard Gives You an Opportunity to Create a Backup Before Proceeding

solution. However, error messages (such as a missing file reference) need to be carefully reviewed and acted upon because they generally involve issues that will prevent you from being able to successfully compile your converted solution. If you have many errors, the Conversion Log does a good job articulating the kind of action you need to take, and you can usually find plenty of additional help using the Visual Studio 2010 help files or at various technical Web sites. In some cases, you may find it best to take note of the Conversion Log errors and address them in your original Visual Studio 2003 project files rather than in the converted solution (this is where the backup you made comes in handy). You can always re-run the Conversion Wizard against your modified Visual Studio 2003 solution.

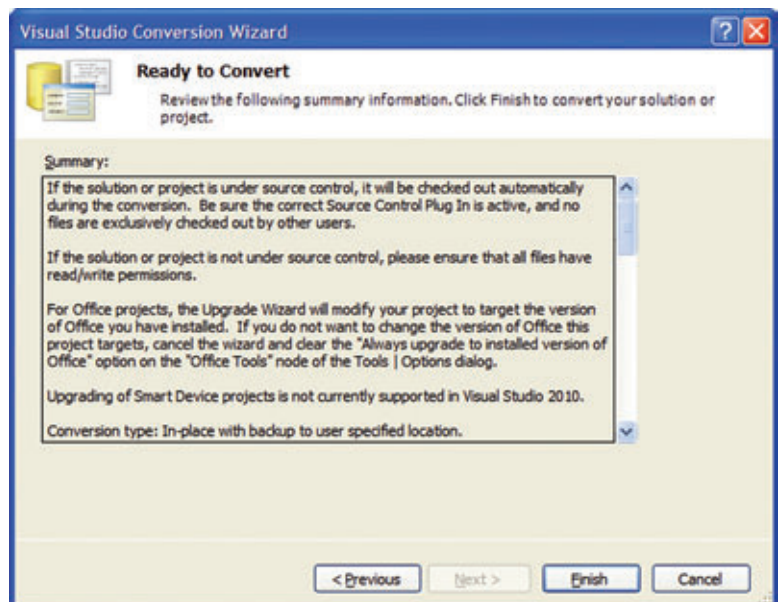


Figure 4 The Conversion Wizard Advises You Before Letting You Continue

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

Word Processing Components
for Windows Forms & ASP.NET

WWW.TEXTCONTROL.COM

Microsoft
Visual Studio
PARTNER

TX Text Control Sales:

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



Figure 5 The Conversion Wizard Lets You Know that You Will Need to Convert Your Web Project in Order to Complete the Migration

Once you've reviewed all warnings and have resolved all errors, you can dismiss the Conversion Wizard's final screens. The Conversion Wizard has now completed its tasks. Your solution (.sln) file and its project (.proj) files are now in the Visual Studio 2010 format. However, you still have some work ahead to complete the migration.

Set the Target Framework and Startup Page When the Conversion Wizard finishes, it will have configured your Visual Studio 2003 Web project to run against the .NET Framework 4 while setting your solution's other projects to run against the .NET Framework 2.0. While it's acceptable to mix-and-match framework versions, I recommend using a single framework target across all of your solution's projects unless you have restrictions imposed on by your Web-hosting company or organization infrastructure. (Visual Studio 2010 modifies its IDE feature set depending on the target framework in effect for the active project. If your projects target different framework versions, you may find the IDE's behavior puzzling as you switch among projects. Having all projects at the same framework version lets the IDE present a consistent interface across all of your projects and also lets you program against a consistent framework.)

If you wish to change the target framework for any of your converted projects, simply right-click the project root in Solution Explorer and select Properties. In the configuration page that appears (see Figure 8), select the Application tab and change the Target Framework to any of the 2.0+ framework values.

Finally, set the start page for your Web project. To do this, find the page in Solution Explorer, right-click it, and choose the Set as Start Page option.

Compile and Fix Now that the solution and project files have been upgraded to the Visual Studio 2010 format, your ASP.NET 2.0+ WAP is ready to be compiled. In order to force-build all projects in your solution, I recommend compiling within Visual Studio 2010 using the Build, Rebuild Solution menu option. After rebuilding, you can view any build issues by displaying the Error List window (you can display this by selecting the View menu and the Error List option). The Error List window displays errors, warnings and messages (you can specify which of these you see in the window by toggling the corresponding Errors, Warnings and Messages buttons). Visual Studio 2010 usually provides clear guidance on how to resolve the items in the Error List window. If you don't understand the error/warning/message text, simply select the Error List window line containing the advisory and hit <F1>. A help window will appear containing more detailed information about the issue (if you didn't install local help, you can connect to the Internet to view it). Yet most of the errors you see will concern changes to the framework that cause naming conflicts with your own code. You can usually resolve those by fully qualifying your references with a namespace. And most of the warnings you see will concern obsolete members. While you can still use obsolete members, you should know that they likely won't be supported in the next-higher version of the .NET Framework. If you see obsolete members and you're targeting the 2.0 framework, you will likely not be able to use that member when you decide to target a 3.x or the 4 framework.

Be prepared to spend some time resolving these Error List window issues. You should be able to resolve all errors and most, if not all, warnings before moving on to the next steps.

Convert Pages and User Controls to Partial Classes

The next step involves running a Convert to Web Application (hereafter, CWA) command. Although you can run this command against an individual page or user control to get a feel for the kind of changes it makes, it's quicker to run it against the entire solution. To do this, right-click the Solution node in Solution Explorer and choose Convert to Web Application. This process does the following:

1. Implements partial classes by adding a new "designer" document for pages and user control.
2. Sets AutoEventWireup for pages and user control.
3. Adds declarative event handlers for each control on pages and user controls.

ASP.NET 1.1 applications have code-behind modules (aspx.cs and ascx.cs for C# and aspx.vb and ascx.vb for Visual Basic .NET) containing both developer-authored and Web-form designer-generated code. When you create pages or user controls in Visual Studio 2003 and add controls to them using the Web form designer, the IDE adds protected instance fields to your code-behind

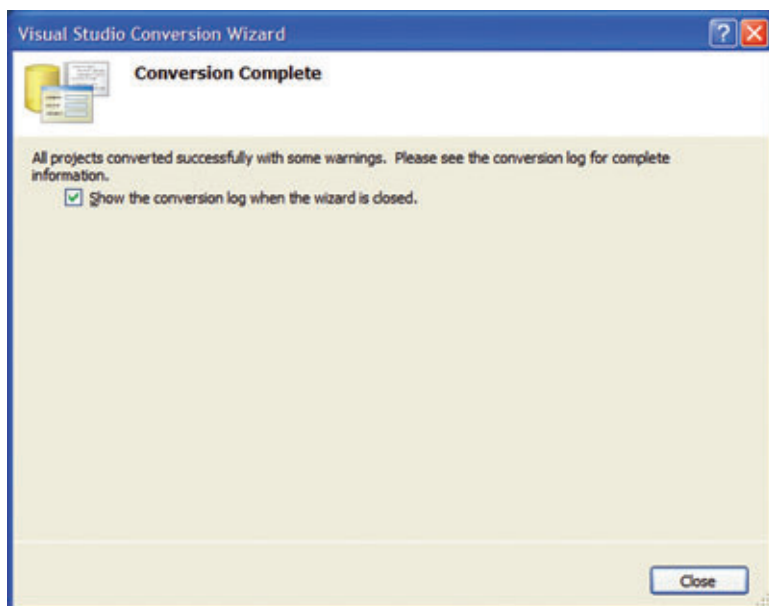


Figure 6 The Conversion Wizard Alerts You When the Conversion Is Complete

Celebrating 100,000 Users

World's Largest File Format Community

- . Create Files
- . Print Files
- . Importing
- . Modify Files
- . Merge Files
- . Exporting
- . Save Files
- . Convert Files
- . Reporting

100% Standalone - No Automation

Serving more than 50% of the Fortune 100 Companies!



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

**DOCX PDF PPT ODF Report SWF InfoPath
XLSX BarCode MPP(Project) MSG(Outlook) ++**

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

Get your **FREE** evaluation copy at www.aspose.com

modules so you can refer to those added controls. After running the CWA command, a designer document appears for each page and user control (the Solution Explorer shows this file only when its Show All Files option has been enabled). You will observe that designer file names are the same as the page or user control along with a designer.cs (C#) or designer.vb (Visual Basic .NET) extension. For example, if you have a page called MyPage.aspx in C#, there will be a new document called MyPage.aspx.designer.cs. This designer document contains protected instance fields that used to be in your code-behind module. Those fields have shifted to the designer module and thus are no longer mixed in with your own code. This is possible because designer modules are partial classes, which means that the CWA command also turns the code-behind code for the corresponding page or user control into a partial class.

For example, instance fields in C# and Visual Basic .NET appear as follows in the code-behind documents of Visual Studio 2003 projects:

```
[VB]
Protected WithEvents MyButton As System.Web.UI.WebControls.Button
```

```
[C#]
protected System.Web.UI.WebControls.Button MyButton;
```

The CWA command moves each to a corresponding designer file:

```
[VB]
Protected WithEvents MyButton As Global.System.Web.UI.WebControls.Button
```

```
[C#]
protected global::System.Web.UI.WebControls.Button MyButton;
```

(global:: indicates that the namespace search for System should begin at the global namespace level and thus assures that the framework System namespace won't be hidden by your own System namespace.)

The creation of the designer file is dynamic and can be regenerated at any time. Thus, you can safely delete your designer.cs or designer.vb document and regenerate it (or restore it if it's missing) simply by right-clicking the page or user-control node in Solution Explorer and re-running the CWA command against it. The CWA

command scans for server controls in the HTML markup for a page or user control and generates the necessary instance variables in the designer partial-class file. It then removes any instance variables that still appear in your own code-behind file (aspx.cs, ascx.cs, aspx.vb or ascx.vb).

Partial classes allow the source code for a single class, struct or interface to be written across two or more physical files within a namespace. The compiler later unites these partial definitions to form a single declaration for each type. While partial classes remain the de facto mechanism used by Visual Studio to cleanly separate developer-authored code from IDE-generated code, developers also leverage them in code-behind modules—especially when working in a team environment.

Most of the errors you see will concern changes to the framework that cause naming conflicts with your own code.

Because partial classes are the norm for ASP.NET 2.0+ applications, you should break up your ASP.NET 1.1 classes into partial classes. If you skip this step, your pages and user controls will continue to function but you will need to manually update control field declarations in the code-behind files when you modify the controls on a page (.aspx) or user control (.ascx).

The CWA command also changes the value of AutoEventWireup as well as the way events are declared and wired up, and I think the impact of this is important enough to discuss in detail. AutoEventWireup is a Boolean attribute that specifies whether Page-object event handlers are wired up implicitly (when True)

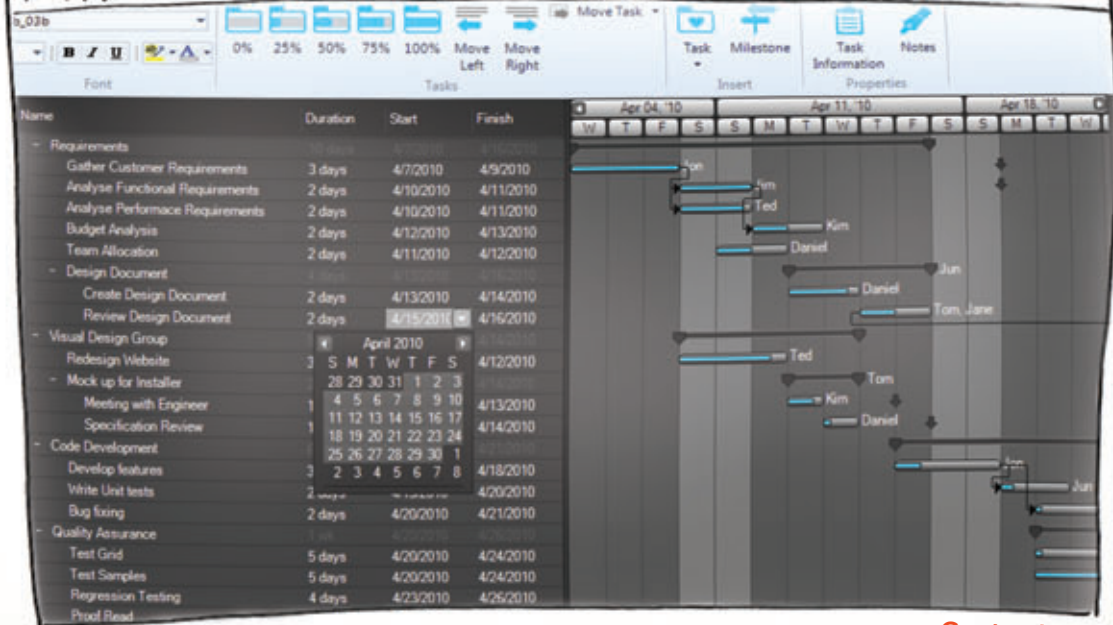
Project: MailConfiguration			
Filename	Status	Errors	Warnings
..\SharedModules\DS\Mail\MailConfiguration\AssemblyInfo.cs		0	0
..\SharedModules\DS\Mail\MailConfiguration\MailConfiguration.csproj	Converted	0	2
Conversion Report - ..\SharedModules\DS\Mail\MailConfiguration\MailConfiguration.csproj: Project file successfully backed up as C:\Documents and Settings\Administrator\My Documents\Visual Studio Projects\MySite\Backup\Backup\MailConfiguration\MailConfiguration.csproj Project converted successfully Scan complete: Upgrade not required for project files. Warning: The project file is being backed up to a relative path that differs from the original solution relative path. The difference in folder hierarchy may create problems in opening or building the backed up solution and project. Your project is targeting .NET Framework 2.0 or 3.0. If your project uses assemblies requiring a newer .NET Framework, your project will fail to build. You can change the .NET Framework version by clicking Properties on the project menu and then selecting a new version in the '.NET Framework' dropdown box. (In Visual Basic, this is located on the Compile tab by clicking the 'Advanced Compiler Options...' button.)			
..\SharedModules\DS\Mail\MailConfiguration\MailConfiguration.csproj.user		0	0
Conversion Report - ..\SharedModules\DS\Mail\MailConfiguration\MailConfiguration.csproj.user: Project user file successfully backed up as C:\Documents and Settings\Administrator\My Documents\Visual Studio Projects\MySite\Backup\Backup\MailConfiguration\MailConfiguration.csproj.user			
..\SharedModules\DS\Mail\MailConfiguration\ModuleConfig.cs		0	0
..\SharedModules\DS\Mail\MailConfiguration\ModuleSettings.cs		0	0
5 files	Converted: 1 Not converted: 4	0	2

Figure 7 The Conversion Log Shows Details About the Converted Web Application

INNOVATION TO SPARK KILLER APPS

KILLER APPS. No Excuses.

3:53Q



Gantt Chart

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

Infragistics
KILLER APPS. No Excuses.

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

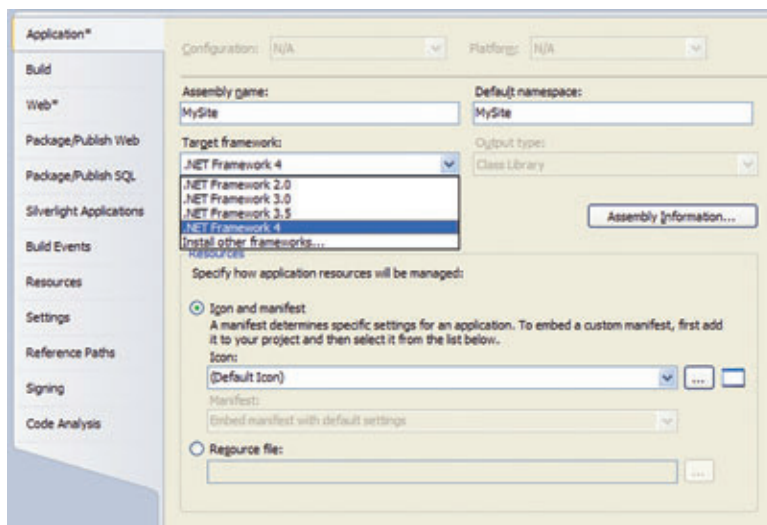


Figure 8 Change the Target Framework for Any Project to 2.0, 3.0, 3.5 or 4

or explicitly (when False). For pages, `AutoEventWireup` is set in the `@Page` tag; for user controls, `AutoEventWireup` is set in the `@Control` tag. The CWA command sets `AutoEventWireup` to True for C# pages and user controls, and sets it to False for Visual Basic .NET pages and user controls.

Developers have different preferences, and it's quite possible that some pages or user controls in your ASP.NET 1.1 application set `AutoEventWireup` to True or False—or don't specify it at all, in which case its default comes from `web.config` or, if not specified there, from `machine.config`. It's important to know that the value of `AutoEventWireup` can change after running the CWA command. This change can cause unanticipated behavior—such as page events firing twice. This most often occurs when you created your own naming convention for Page-object events in your ASP.NET 1.1 application. For example, consider this C# code in which a `Page_Load2` handler is wired up to the `Page.Load` event delegate:

```
this.Load += new System.EventHandler(this.Page_Load2);
```

When `AutoEventWireup` is False, the event will fire once, as expected—even if there is a code-behind function called `Page_Load`. However, when `AutoEventWireup` is True, both events fire—once for the explicit wire-up code shown here, and once for the implicit wire-up code that subscribes the `Page_Load` event handler to the `Page.Load` event. Consider the code in **Figure 9**.

The code in **Figure 9** generates this output:

```
In Page_Load().
In Page_Load2().
Top of Form 1
Bottom of Form 1
```

The same thing happens in Visual Basic .NET when `AutoEventWireup` is set to True. Consider the following code:

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)

    Response.Write("In Page_Load.<br />")

End Sub

Private Sub Page_Load2(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles MyBase.Load

    Response.Write("In Page_Load2.<br />")

End Sub
```

When `AutoEventWireup` is True, both event handlers fire, causing the page to display:

```
In Page_Load2.
In Page_Load.
```

You can see that not only do two event handlers execute, but also the order in which they execute—as with all multicast delegates—might not be what you expect. Finally, realize that when `AutoEventWireup` is True, page event handlers with the proper function name, such as `Page_Load`, will fire whether they're defined with or without arguments. For example:

```
protected void Page_Load()
{
    Response.Write("In Page_Load().<br />");
}
```

is the same as:

```
protected void Page_Load(object sender, EventArgs e)
{
    Response.Write("In Page_Load().<br />");
}
```

If both are present, only the one with arguments fires—another issue to consider when troubleshooting. Thus, in general, be careful testing pages and user controls, especially when the `AutoEventWireup` setting was changed by the CWA command.

Finally, the CWA command removes explicit C# and Visual Basic .NET code that wires up control events and instead uses declarative event attributes in the page's or user control's markup. For example, in ASP.NET 1.1, a click event on a button would typically have an event handler in code-behind, such as:

```
this.MyButton.Click += new System.EventHandler(this.MyButton_Click);
```

It's important to regression test and QA test your migrated code, and be especially careful to ensure that events fire as expected.

The CWA command removes this and instead adds an `OnClick` attribute to the server-control declaration as follows:

```
<asp:Button ID="MyButton" runat="server" Text="MyButton" onclick="MyButton" />
```

In Visual Basic .NET, declarative events aren't added. Instead, the `Handles` keyword is added to the code-behind. Thus, the markup for a Button control will appear as:

```
<asp:Button ID="MyButton" runat="server" Text="Button" />
```

while the code-behind wires up the control to the handler:

```
Protected Sub MyButton_Click(
    ByVal sender As Object, ByVal e As EventArgs) _
    Handles MyButton.Click
```

```
End Sub
```

Event delegates for these declarative event-handler constructs are created at compile time.

Compile and Fix The migration is now complete. I recommend that you rebuild your solution and go through the exercise of

way Go[^]Beyond RIA Services

with DevForce Silverlight 2010

And celebrate your
giant leap forward
in productivity
and functionality!

**When you step up from WCF RIA Services
to DevForce Silverlight, you get:**

- ✓ Full LINQ support from the client
- ✓ Huge performance boost with client-side caching
- ✓ A standardized development environment for all .NET client technologies: Silverlight, WPF, WinForms and ASP.NET
- ✓ More loosely coupled model development
- ✓ A mature product supported by a company whose only focus is data services

Learn more at IdeaBlade.com/GoBeyond
Contact sales at 510-596-5100 • e-mail: sales@ideablade.com



IDEABLADE™

Figure 9 Testing AutoEventWireup Behavior

```
public partial class _Default : System.Web.UI.Page
{
    override protected void OnInit(EventArgs e)
    {
        InitializeComponent();
        base.OnInit(e);
    }

    private void InitializeComponent()
    {
        this.Load += new System.EventHandler(this.Page_Load2);
    }

    protected void Page_Load()
    {
        Response.Write("In Page_Load().<br />");
    }

    protected void Page_Load2(object sender, EventArgs e)
    {
        Response.Write("In Page_Load2().<br />");
    }
}
```

addressing any remaining compiler errors or warnings you may receive. Note that you need to resolve compilation errors before you can successfully build your Web project and before you can view your migrated Web site in a browser. You should also address compilation warnings, but they aren't considered critical and won't prevent you from using your Web application. Finally, take note of all compiler messages because they generally provide helpful recommendations for improvements you should consider implementing when time allows. Beyond errors and warnings, it's important to regression test and QA test your migrated code, and be especially careful to ensure that events fire as expected.

Post-Migration Considerations

Because your ASP.NET project now targets a newer framework and is running under a more modern version of the IDE, you should be aware of a few additional issues.

If you migrated your ASP.NET 1.1 solution from a 32-bit OS to a 64-bit OS, you should realize that IIS 6.0 and later support both 32-bit and 64-bit operating modes. Because ASP.NET 1.1 runs only in 32-bit mode, you may find that your converted ASP.NET app still has 32-bit dependencies (such as COM or P/Invoke calls) which may not continue to function properly post-migration. To fix this, access the Advanced Settings of your application's Application Pool and set the Enable 32-bit Applications value to True.

Visual Studio 2010 expects Web pages to be XHTML-compliant. Your pages from ASP.NET 1.1 likely are not. Most pages will therefore show XHTML validation warnings (to see these, view your page in Source or Design mode, then access the View menu and select Error List). While these warnings won't prevent a page from running, they indicate that the pages may not render properly in modern browsers. As time permits, you should update your pages and user controls so they're XHTML-compliant, which will ensure that they render properly in modern browsers. If your Web app specifically targets an older browser—or you

don't want to be bothered with markup validation errors at this time—you can change how markup is validated by going to the Tools menu, choosing Options and going to the Text Editor node, then changing Target to Internet Explorer 6 (see Figure 10). This approach is suitable only for developers who are required to target Internet Explorer 6, as might be the case if your app is a corporate intranet app, for example. This effectively sets rendering validation to a level similar to the one you likely used in Visual Studio 2003.

For those apps that will need to display properly in browsers other than Internet Explorer or in versions of Internet Explorer beyond version 6, you should continue to show XHTML markup errors in the HTML editor and you should make use of the new .NET Framework 4 `controlRenderingCompatibilityVersion` configuration setting, available as a property in the Web.config system.web section:

```
<system.web>
  <pages controlRenderingCompatibilityVersion="4.0"/>
</system.web>
```

If `controlRenderingCompatibilityVersion` isn't set in Web.config, it defaults to the running version of ASP.NET. When you specify the `controlRenderingCompatibilityVersion` value, however, you can set it to either "3.5" or "4.0" (the conversion wizard sets it to "3.5," which renders pages the same way as in ASP.NET 3.5). This setting determines how markup specified in your .aspx files is ultimately rendered to the browser. To cite an example from the Visual Studio 2010 online help, when `controlRenderingCompatibilityVersion` is set to "3.5," a server-side ASP.NET Label control with its `IsEnabled` property set to false will render an HTML span element with its "disabled" attribute set to "disabled"; when `controlRenderingCompatibilityVersion` is set to "4.0," the resulting span element will instead include a "class" attribute with a reference to a CSS class.

You should be aware that using the "4.0" setting generates modern XHTML markup—and this can break client script or CSS rules that worked properly in the ASP.NET 1.1 version of the page, thereby affecting the behavior and/or aesthetics of the rendered content. Thus, until you are fully committed to generating valid XHTML, I suggest that you set `controlRenderingCompatibilityVersion` to "3.5." And if you use this "3.5" setting, you need to know about `xhtmlConformance` (which applies only if `controlRenderingCompatibilityVersion` is set to "3.5"), which can be set to "Legacy," "Strict"

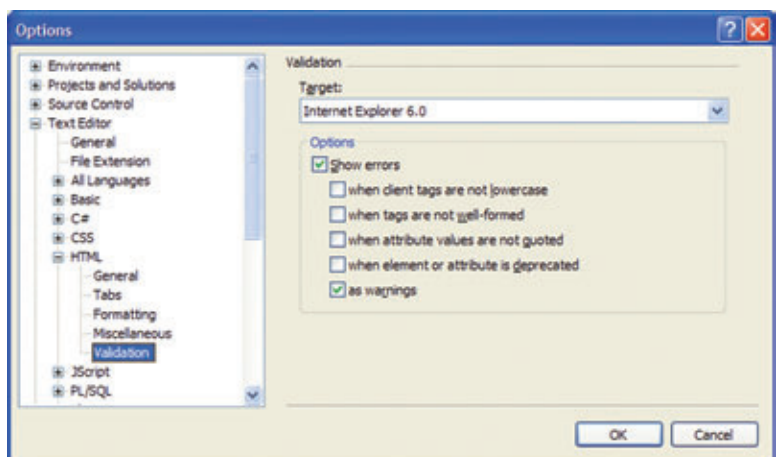


Figure 10 You Can Suppress XHTML Validation Errors by Changing the HTML Validation Target to Internet Explorer 6

or “Transitional” (the default value). “Strict” renders XHTML 1.0 Strict markup; “Transitional” renders XHTML 1.0 Transitional markup; “Legacy” renders HTML similar (but not necessarily exactly) to the way it was rendered in ASP.NET 1.1:

```
<system.web>
  <pages controlRenderingCompatibilityVersion="4.0"/>
  <xhtmlConformance mode="Transitional"/>
</system.web>
```

In my experience, you should avoid “Legacy” mode, as it can interfere with the proper functioning of the ASP.NET AJAX UpdatePanel. Also, note that the controlRenderingCompatibilityVersion value doesn’t change the DOCTYPE of your Web page—it simply changes the way ASP.NET controls render themselves. Thus, getting your pages to render properly is largely due to the combination of controlRenderingCompatibilityVersion, xhtmlConformance and DOCTYPE values as well as the target browser type and version used.

Another thing to consider: You may want to change the virtual directory under which your newly migrated site runs—especially if you plan to run it in parallel with the ASP.NET 1.1 version. To do this, right-click the Web project in Solution Explorer, choose Properties and then access the Web tab. Under Servers, you’ll see an option for Use Local IIS Web Server. Be sure that option is selected, specify a Project URL (such as <http://localhost/mysitemigrated>), and click the Create Virtual Directory button if the virtual directory doesn’t exist.

ASP.NET 1.1 applications use the Windows ASPNET user to assign privileges on files and folders under the virtual root. ASP.NET 2.0+ uses the NETWORK SERVICE user. If your application requires that ASP.NET have write access to certain files or folders, for example, it’s important to grant those rights to the NETWORK SERVICE user. If you’re ever unsure which user needs this access, you can view that user name by examining the value of the Environment.UserName property while running an ASP.NET application.

If you use any third-party add-ons or dependencies (whether as binary or source), you will want to check with the vendor to ensure that you have the latest version. For example, the popular logging program, NLog, offers 1.1 and 2.0 library builds. Grab the 2.0 build and spare yourself the effort of migrating the 1.1 code yourself. Also, vendors will be providing updates to productivity add-ons designed for the Visual Studio IDE. Be sure to upgrade to obtain the most current versions of those products for your new Visual Studio 2010 IDE.

After migrating your ASP.NET 1.1 Web apps, you’ll realize two immediate benefits. First, you’ll no longer require Front Page Server Extensions (FPSE) to power your site (unless you optionally wish to continue using it). Second, you’ll no longer need to have IIS installed on your development machine because Visual Studio 2010 offers its own built-in ASP.NET Development Server. And while you can continue to run Visual Studio 2003 ASP.NET applications side-by-side with Visual Studio 2010 ASP.NET applications (for QA/debugging purposes, for example), your converted Visual Studio 2010 ASP.NET application will no longer require Visual Studio 2003 or access to the .NET Framework 1.1.

If you’re a C# developer, you may notice that page and user-control events no longer appear in the Properties window as a yellow thunderbolt icon when viewing them in design mode. To

view/create these events as in Visual Studio 2003, you will need to right-click the page (.aspx) or user control (.ascx) in the Solution Explorer and choose View Component Designer. At that point, you’ll see a Properties window that contains the familiar list of events. You can double-click any of the events in the list to create your event procedure and delegate wire-up code (this gets added to the InitializeComponent function).

When it’s time to host your site, you need to know which CLR is required. When you use the 3.0 and 3.5 .NET Frameworks, you’re running against the CLR 2.0; when you use the .NET Framework 4, you’re running against the CLR 4.0. Your hosting company must support the CLR 4.0 in order to host your ASP.NET 4.0 solution.

I hope I’ve helped you transition your Visual Studio 2003 ASP.NET apps to Visual Studio 2010. Once you do so, you’ll have a whole new world of programming technologies at your disposal. I believe it’s a relatively painless technology decision you’ll never regret making. ■

JONATHAN WALDMAN has written for PC Magazine and is a senior Microsoft Certified Professional who leverages .NET Framework technologies to passionately create customized software solutions for the desktop and the Web. He may be reached at jonathan.waldman@live.com.

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

The advertisement for TeaChart software features a dark background with several product icons on the left: 'TeaChart Charts' (blue), 'TeaChart Maps' (orange), and 'TeaChart Gauges' (red). To the right, a screenshot of the software interface shows a dashboard with a bar chart, a gauge, and a world map. At the bottom, the text reads 'Charting for Developers' followed by 'Charts, Maps and Gauges for Microsoft.NET (WinForms, Pocket, WebForms and Ajax), for Delphi/C++Builder, as native Java, ActiveX or PHP.' A circular logo for 'steema software' with the website 'www.steema.com' is also present.

Create a Silverlight 4 Web Part for SharePoint 2010

Paul Stubbs

Microsoft SharePoint 2010 provides a business collaboration platform that's easy to customize into tools that organizations can depend on and grow with. And when building custom SharePoint solutions, it's best to take advantage of Silverlight on the front end.

Web Parts are a central part of most SharePoint solutions. When most developers think about using Silverlight in their SharePoint app, using it in Web Parts seems like the obvious direction, and Web Parts are the most common way to create Silverlight applications in SharePoint. But Web Parts aren't your only option.

You can also use Silverlight in menus, navigation, dialogs, page layouts, master pages—anywhere you can put an object tag. This gives designers a lot of flexibility to create great experiences that feel integrated into SharePoint. With this approach, Silverlight should feel like a natural extension to SharePoint.

This article discusses:

- Creating sample data
- From Silverlight to Web Part
- Deploying the Web Part
- Using the client object model

Technologies discussed:

Silverlight 4, SharePoint 2010, Visual Studio 2010, Expression Blend

Code download available at:

code.msdn.microsoft.com/mag201009Silverlight

The sheer breadth and depth of both the SharePoint platform and Silverlight can be daunting for developers. In fact, it's common for developers to focus entirely on SharePoint, and for designers to be more familiar with Silverlight. To build a useful SharePoint application, you'll need to understand both technologies. In this article, I'll give you an overview of Silverlight integration with SharePoint 2010 and walk you through the basics of using Silverlight in the front end of a SharePoint solution.

Silverlight and SharePoint Workflow

The first step in working with Silverlight and SharePoint together is using the right tools. Visual Studio and Expression Blend are designed to work together, and by using them a Silverlight developer can create a great application using sample data from SharePoint without ever seeing or installing SharePoint. Likewise, a SharePoint developer can integrate the Silverlight application into SharePoint without having to understand or delve into the XAML code.

In this article, I'll show you how to create a simple Silverlight Web Part that will use the SharePoint client object model for Silverlight to surface a SharePoint list. The application will also be a SharePoint sandboxed application that will enable Site Collection administrators to install and manage the application. The application will also work on SharePoint Online standard accounts.

Figure 1 shows what the final application will look like when it's running on SharePoint.

WE ARE SPREADSHEETS

Act now and save up to \$300!

 GrapeCity PowerTools

SPREAD⁵

Windows Forms & ASP.NET

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

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



GCPowerTools.com/ActNow



WE ARE
GRAPECITY
Excel  Report  Analyze

1-800-645-5913 / 1-919-460-4551

WE ARE

REPORTING

Act now and save up to \$300!

 GrapeCity PowerTools

ACTIVE REPORTS 6

Powerful Award-winning .NET reporting tool for Microsoft Visual Studio

- Unmatched customization, performance and quality
- Rich collection of report designing and rendering components
- Medium Trust environment support in ASP.NET
- Wide range of cross-platform export and preview formats
- Powerful PDF reporting with digital signatures, timestamps and multilanguage support

GCPowerTools.com/ActNow



WE ARE
GRAPECITY
Excel  Report  Analyze

1-800-645-5913 / 1-919-460-4551

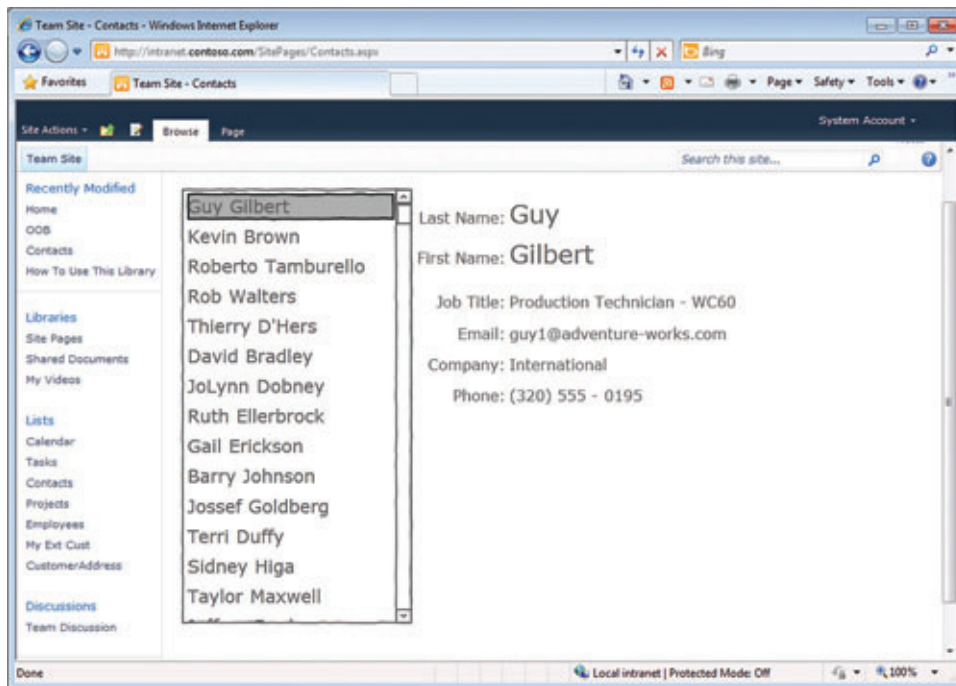


Figure 1 The Completed Silverlight/SharePoint Application

SharePoint Sample Data

In order for Silverlight and SharePoint developers to work independently, the Silverlight developer will need sample data based on SharePoint data. This will enable the Silverlight developer to be completely free to create an application that will work correctly when plugged into SharePoint.

Expression Blend has a number of features for supporting design time data and sample data. In this example, I will use the XML-formatted sample data. This means the SharePoint developer must create an XML file that represents the SharePoint List that the application will use. You can make this sample data file in any shape you want, but everything becomes much easier if you make it the same shape as what will be returned from SharePoint. For example, the shapes returned using the WCF Data Services entities and the RESTful ListData services of SharePoint are different from the shape of the data returned using the Silverlight client object model.

In this application, I use some of the new data-binding features of Silverlight 4 and the SharePoint client object model for Silverlight. There's no built-in way to generate this sample data, so you'll need to create a simple console application to generate the SharePoint sample data. In this case, I created a Visual Basic console application that uses the Microsoft .NET Framework client object model to generate an XML document from the top five list items (see **Figure 2**).

Once you have the SharePointSampleData.xml file, you'll be able to use this in Expression Blend as the design time data source.

Designing with Sample Data

With the sample data XML file in hand, you can open Expression Blend and create a new Silverlight 4 application.

On the Data panel, click the Create Sample Data icon in the top-right corner and choose Import Sample Data from XML,

then browse to the SharePoint-SampleData.xml file. You can also leave the block checked to see the sample data while the application is running. Otherwise, you'll only see the sample data in Blend or in the Visual Studio designer.

Figure 3 shows the Contacts List. The client object model actually returns a dictionary of 45 fields, so you won't be able to see them all in the figure without scrolling.

The next step is to create a data-bound list box. In the Data panel, make sure that the List Mode icon is set. You'll find this in the top-left of the Data panel. Now select the fields you want to appear in the list box. In this case I selected Title, which actually is the first name, and I selected LastName. Drag the fields from the list and drop them onto the design surface. Expression Blend creates a list box and data

template data-bound to the fields. Arrange the list box to the left side of the design surface to make space for the details.

Creating the data-bound details is just as easy. Select Details Mode from the icon on the top left of the Data panel window. Select the fields you want to appear as the detail fields. Drag the fields onto the design surface. Expression Blend will create each field bound to the selected item in the list box. If you press F5 and run the application, you'll see the list box populated with the sample data. As you change the selected item in the list box, you'll see the details change on the right (see **Figure 4**).

Web Parts are a central part of most SharePoint solutions.

You now have a fully functional application data-bound to the SharePoint sample data. As a Silverlight designer, you can continue to test and refine your application without ever knowing anything about SharePoint. In this example, I have done a little more design work and arranged all of the fields and added the Sketch theme from the Silverlight Toolkit.

The Silverlight Web Part

At this point, the Silverlight developer can hand the project file over to the SharePoint developer. This is possible because Expression Blend and Visual Studio share the same project files. The SharePoint developer opens the project in Visual Studio 2010 and can see all of the parts of the application so he can work on the application in the Visual Studio designer (see **Figure 5**).

Figure 2 Console App to Return Data XML

<pre>Imports Microsoft.SharePoint.Client Imports System.Text Module Module1 Sub Main() Dim context As New _ ClientContext("http://intranet.contoso.com") Dim sampleListItems As ListItemCollection Dim sampleList As List = _ context.Web.Lists.GetByTitle("Contacts") Dim qry As New Microsoft.SharePoint.Client.CamlQuery qry.ViewXml = "<View><RowLimit>5</RowLimit></View>" sampleListItems = sampleList.GetItems(qry) context.Load(sampleListItems) context.ExecuteQuery()</pre>	<pre>'Build the Sample XML Data Dim SampleData = _ <SharePointListItems></SharePointListItems> 'Iterate through each row using Linq 'and XML Literals For Each item In sampleListItems Dim sampleItem = <ListItem> <%= From fieldValue In item.FieldValues _ Select <<%= fieldValue.Key %>> <%= fieldValue.Value %></> %> </ListItem> SampleData.Add(sampleItem) Next 'Write the file to disk System.IO.File.AppendAllText(_ "C:\SharePointSampleData.xml", _ SampleData.ToString()) End Sub End Module</pre>
---	--

The first thing the SharePoint developer will need to do is add an empty SharePoint project to the existing solution that contains the Silverlight project. He'll use this SharePoint project to deploy the Silverlight application file (.xap) and to create a Silverlight Web Part to host the Silverlight application.

Visual Studio 2010 makes doing both of these tasks easy compared to building SharePoint solutions in the past. Right-click on the solution and choose Add New Project. Browse to the SharePoint 2010 templates and select Empty SharePoint Project. In the New Project wizard, be sure to keep the default selection for creating the project as a sandboxed solution. This is a best practice and gives you the most flexible and secure deployment options. Silverlight also works well with sandboxed solutions because Silverlight runs on the client and isn't limited to many of the restrictions as server-side code running in the sandbox.

Although the order really doesn't matter, it's better to create the Web Part first. This will enable you to nest the module that will deploy the Silverlight .xap file under the Web Part. This just keeps the project structure well organized and easier to follow as your solutions get larger.

Add the Web Part to your SharePoint project by right-clicking on the project and choosing Add New Item. Select Web Part from the SharePoint 2010 template folder. This will create a new empty Web Part in your project.

A Web Part consists of just three items. The first is the Elements.xml file. This is a SharePoint solution file that describes the items in a feature.

The second is the Web Part definition file, which has a .webpart extension. This file is deployed to the SharePoint Web Parts gallery and defines the code for the Web Part and all of the properties of the Web Part.

The third file is the code file. All three of these files are placed under a folder in the project structure.

So this is where it gets interesting. All of the files that are added to the project when you create a new Web Part are needed if you're creating a Web Part from scratch. In this

case, you don't need to use all of the auto-generated files because SharePoint already ships with a Silverlight Web Part. What you want to do is to build on this existing Web Part. To do that you simply need to make the .webpart file point to the code for the Silverlight Web Part and include all of the properties.

Expression Blend creates a list box and data template data-bound to the fields.

Create a copy of the built-in Silverlight Web Part .webpart file. There's actually an automatic way to do this. In SharePoint add a Silverlight Web Part to any page. While in design mode, click on the drop-down menu of the Web Part and choose Export. This lets you make a copy of the .webpart file that you can add to your project. Now just change a few properties such as the path to the Silverlight application, title, description, and so on. Also, because you're now using the code for the Web Part from the built-in Microsoft.SharePoint.dll, you no longer need the .cs file for the Web Part. You can simply delete this file. **Figure 6** shows an example of what your .webpart file will look like once you add it to your project.

Deploying Silverlight to SharePoint

At this point, you have a fully functional Silverlight application and a Web Part to host the Silverlight application. Now you need to create a SharePoint module to deploy the actual Silverlight .xap file to SharePoint.

Visual Studio 2010 has built-in functionality to do this, so hooking everything up is straightforward. Right-click on your Web Part folder and choose Add New Item. Select Module template from the SharePoint 2010 templates.

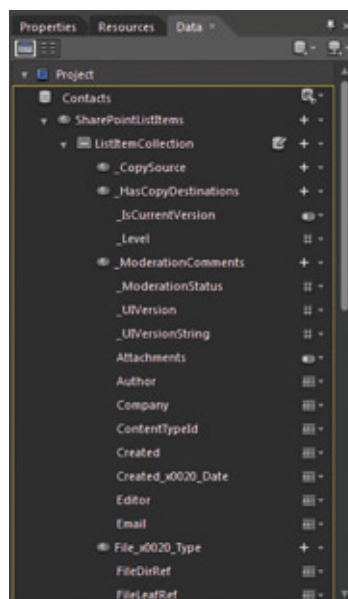


Figure 3 The Sample Contacts List

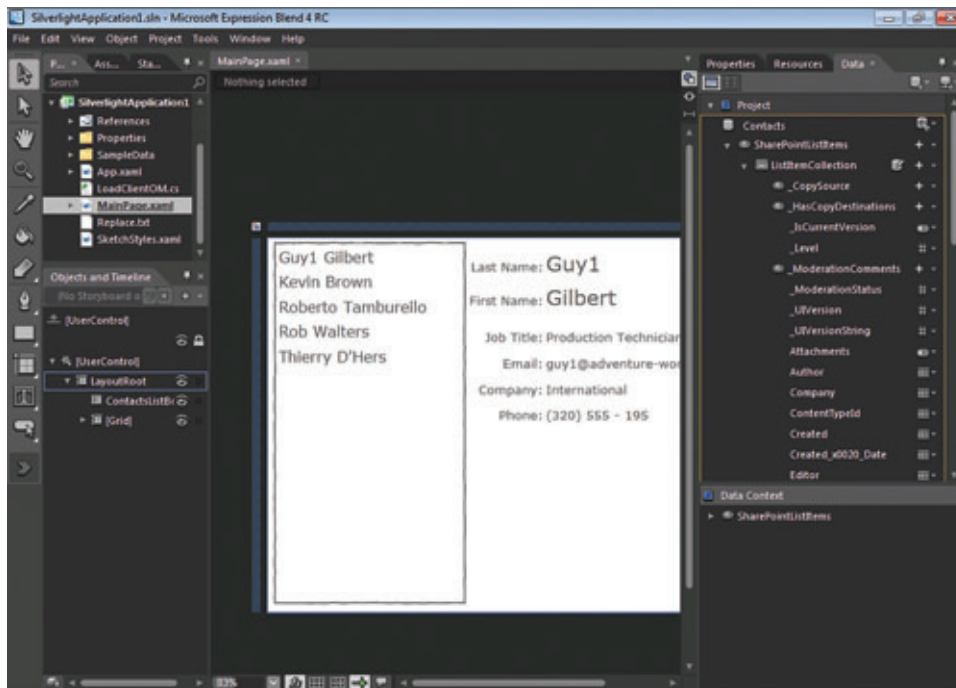


Figure 4 Working on the App in Expression Blend

A new module, by default, contains an elements.xml file, just like the one you added to the Web Part feature. It also contains a sample.txt file. Delete the sample.txt file, as you will not need it.

There's a special way to add a reference to the Silverlight application in your project. Select the Module folder in the Solution Explorer and view the properties. Click on the Project Output References property to open the Project Output References dialog. In the dialog, click the Add button to add a new reference. In the Project Name drop-down menu, choose the Silverlight application project. Set the Deployment Type to Element File and close the dialog.

You've now added a reference to the Silverlight app. This works much like standard project references where the Silverlight project will build before the SharePoint project and the output of the Silverlight project (the .xap file) will be copied to the SharePoint project and built into the SharePoint solution file (the .wsp file).

The last step is to set the deployment path of the Silverlight application. This is the location to which the .xap file will be copied and the location from which the Web Part will load the application.

Open the elements.xml file under the Module folder. Set the URL property of the file element to be the same as the URL property in the .webpart

file. In this case you will deploy to the master page gallery so set the value to: `~site/_catalogs/masterpage/SilverlightApplication1.xap`

Note the tilde at the beginning of the path is a special SharePoint wildcard to indicate the root of the site.

Now deploy the solution to SharePoint to verify that everything is set up correctly. Set the SharePoint project as the default startup project and press F5. Visual Studio will build and package the solution. Although Visual Studio will open the SharePoint site and attach the debugger, you still need to add the Web Part to the page. I find the easiest way to do this while developing my solutions is to add the Web Part to a new Site Page.

Client Object Model

With everything in place, you're ready to wire up the Silverlight

application to actual SharePoint data. SharePoint 2010 includes client-side object models for the CLR, Silverlight and ECMAScript. These enable easy access using a familiar object model to SharePoint data, such as Lists and Libraries.

In this particular case you're building a Silverlight app, so you need to add a reference to the Silverlight client object model. The Silverlight client object model is made up of two files, Microsoft.SharePoint.Client.Silverlight.dll and

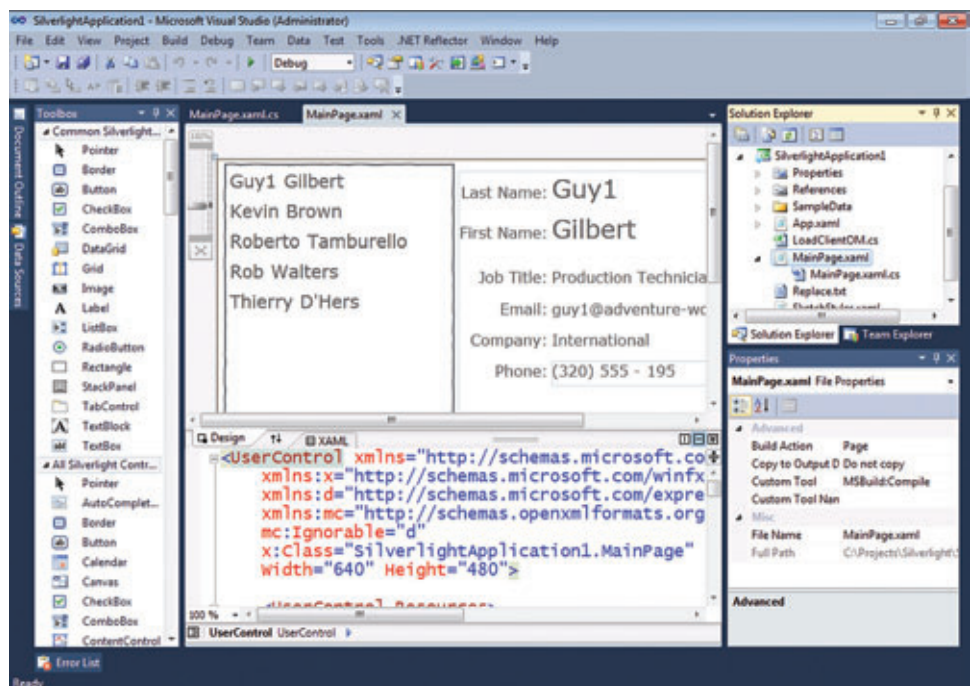


Figure 5 Working on the App in Visual Studio

Microsoft.SharePoint.Client.Silverlight.Runtime.dll. These files are located in C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\14\TEMPLATE\LAYOUTS\ClientBin\.

Once you have the references added, you can write the code to retrieve the contacts from the Contacts list. Start by adding a using statement for Microsoft.SharePoint.Client to the top of the MainPage.xaml.cs. Then you'll need to define a couple of class-level fields to hold the returned results. You also need to handle the page-loaded event, which is used to load the List data:

```
public partial class MainPage : UserControl {
    ClientContext ctx;
    ListItemCollection contactsListItems;

    public MainPage() {
        InitializeComponent();
        this.Loaded += MainPage_Loaded;
    }

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

Next, implement the LoadList method to retrieve the data from SharePoint. First, get a reference to the current client context. The static method ClientContext.Current returns a context to the site in which the Web Part is loaded. You then call GetByTitle to get a reference to the Contacts list. The Load method will add the request to the query. Then call ExecuteQueryAsync to make the call to the SharePoint Server:

```
void LoadList(){
    ctx = ClientContext.Current;

    if (ctx != null) { //null if not in SharePoint
        List contactsList =
            ctx.Web.Lists.GetByTitle("Contacts");
        contactsListItems =
            contactsList.GetItem(
                CamlQuery.CreateAllItemsQuery());

        ctx.Load(contactsListItems);
        ctx.ExecuteQueryAsync(
            ContactsLoaded, ContactsLoadedFail);
    }
}
```

Calls to the server in Silverlight are asynchronous. The ExecuteQueryAsync method takes two callback delegates, one for succeeded

and one for failed results. In the ContactsLoaded callback method, you data bind the results to the XAML that the Silverlight designer created in the beginning of the article. In the succeeded callback, all you need to do is set the ItemsSource property of the list box to the contacts collection returned by the client object model. The callback also occurs on a background thread, so you'll need to update the property inside of the Dispatcher's BeginInvoke method:

```
// ContactsLoaded
void ContactsLoaded(object sender,
    ClientRequestSucceededEventArgs args) {
    //call back on the UI thread
    this.Dispatcher.BeginInvoke(() => {
        ContactsListBox.ItemsSource = contactsListItems;
    });
}
```

I left the failed callback for you to implement.

SharePoint 2010 includes client-side object models for the CLR, Silverlight and ECMAScript.

There's one more thing you need to do before running the application: you need to change the way Silverlight binds to the data source. Remember, in the beginning of the article, I created an XML file that represented the data from SharePoint, but this isn't exactly the same data because the SharePoint client object model actually returns a collection of dictionary field values.

One of the new features of Silverlight 4 is the ability to bind to indexers. This means you can data bind to the dictionary values returned by the client object model using the name of the key value as a string to the binding command. Unfortunately, there isn't a good way to deal with this complex dictionary-based sample data, but it's easy enough to change the bindings back and forth using the Visual Studio find and replace tools. For example, you need to change the Text="{Binding Title}" to Text="{Binding Path=FieldValues[Title]}".

Figure 6 Excerpt from the Default .webpart File

<pre><webParts> <webPart xmlns="http://schemas.microsoft.com/WebPart/v3"> <metaData> <type name="Microsoft.SharePoint.WebPartPages.SilverlightWebPart, Microsoft.SharePoint, Version=14.0.0.0, Culture=neutral, PublicKeyToken=7 1e9bce11e9429c" /> <importErrorMessage\$Resources:core,ImportErrorMessage; </importErrorMessage> </metaData> <data> <properties> <property name="HelpUrl" type="string" /> <property name="AllowClose" type="bool">True</property> <property name="ExportMode" type="exportmode">All</property> <property name="Hidden" type="bool">False</property> <property name="AllowEdit" type="bool">True</property> <property name="Description" type="string">Contacts Web Part</property></pre>	<pre>... <property name="Height" type="unit">480px</property> <property name="ChromeType" type="chrometype">None</property> <property name="Width" type="unit">640px</property> <property name="Title" type="string">Contacts Demo</property> <property name="ChromeState" type="chromestate">Normal</property> <property name="TitleUrl" type="string" /> <property name="Url" type="string"> ~site/_catalogs/masterpage/SilverlightApplication1.xap </property> <property name="WindowlessMode" type="bool">True</property> </properties> </data> </webPart> </webParts></pre>
---	--

Figure 7 Loading Microsoft.SharePoint.Client.xap

```
using System;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Ink;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
using System.IO;
using System.Windows.Resources;
using System.Reflection;

namespace SilverlightApplication1 {
    public class LoadClientOM {
        private bool isDownloaded = false;
        private Action action;

        public LoadClientOM(Action action) {
            this.action = action;
        }

        public void Run() {
            WebClient client = new WebClient();
            client.OpenReadCompleted +=
                new OpenReadCompletedEventHandler(
                    client_OpenReadCompleted);
            client.OpenReadAsync(new Uri(
                "/_layouts/clientbin/Microsoft.SharePoint.Client.xap",
                UriKind.Relative));
        }

        void client_OpenReadCompleted(object sender,
            OpenReadCompletedEventArgs e) {
            Stream assemblyStream;
            AssemblyPart assemblyPart;

            assemblyStream = Application.GetResourceStream(
                new StreamResourceInfo(e.Result, "application/binary"),
                new Uri("Microsoft.SharePoint.Client.Silverlight.Runtime.dll",
                    UriKind.Relative)).Stream;
            assemblyPart = new AssemblyPart();
            Assembly b = assemblyPart.Load(assemblyStream);

            assemblyStream = Application.GetResourceStream(
                new StreamResourceInfo(e.Result, "application/binary"),
                new Uri("Microsoft.SharePoint.Client.Silverlight.dll",
                    UriKind.Relative)).Stream;
            assemblyPart = new AssemblyPart();
            Assembly a = assemblyPart.Load(assemblyStream);

            this.isDownloaded = true;

            if (action != null) {
                action();
            }
        }
    }
}
```

Do this for all of the fields. I've included the regular expression code to switch back and forth in the sample code for this article.

Once you've changed all of the bindings to use the Silverlight 4 indexer binding, you're ready to run the application. Browse to the site page you created to host the Silverlight application. It should look like **Figure 1** that you saw in the beginning of the article.

Dynamically Loaded Client Object Model

At this point your application is complete, but there's one more thing I want to show you. This is a change to SharePoint since the beta version. When I created a reference to the Silverlight client object model by browsing to the files using the Visual Studio Add References dialog, by default, Visual Studio includes these two files in the .xap package that it builds. This adds an additional 407KB to your Silverlight package.

One of the new features of
Silverlight 4 is the ability to
bind to indexers.

A better option is to dynamically load these assemblies at run time. This enables the browser to cache these common files, reducing your application size and the load times while increasing the performance of your applications.

Since the SharePoint beta release, SharePoint packaged the two client object model files into a single .xap file called Microsoft.SharePoint.Client.xap. This file is in the same location as the other client object model files, C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\14\TEMPLATE\LAYOUTS\ClientBin. You still need to add a reference to the client object model files because this will give you the IntelliSense and

compile support you need. But you'll need to select each file in the references folder of your Silverlight application and set the Copy Local property to false. Setting this to false will prevent Visual Studio from adding these files to your .xap package.

Next, you need to add code to dynamically load the client object model assemblies. This is very generic code and can be used in any Silverlight application on any .xap package with little modification. Start by changing the page-loaded event to call the code to download and load the assemblies. In this example, you pass a callback delegate to the LoadList method. This way, when the assemblies are loaded and ready to use, you'll then load the list data from SharePoint, like so:

```
void MainPage_Loaded(
    object sender, RoutedEventArgs e) {
    LoadClientOM loadClientOM =
        new LoadClientOM(delegate() { LoadList(); });
    loadClientOM.Run();
}
```

Add a new class file to your project called LoadClientOM.cs and add the code shown in **Figure 7**. This code uses the WebClient to download the .xap package from /_layouts/clientbin/Microsoft.SharePoint.Client.xap. Once the package is downloaded, you load each assembly from the package.

You can now run the application again. You'll see that the application looks exactly the same, but now you're loading the Silverlight client object model dynamically at run time from the server. It may be difficult to detect in this simple application, but you should also see better performance loading them dynamically. ■

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

THANKS to the following technical experts for reviewing this article:
Mike Morton, John Papa and Unni Ravindranathan

Making MapPoint 2010 and SQL Server Spatial Work Together

Eric Frost and Richard Marsden

After Bing Maps, two of the most visible geospatial technologies from Microsoft are Microsoft MapPoint 2010 and the spatial functionality in SQL Server 2008 R2. However, even though SQL Server is an ideal store for geospatial data, and MapPoint makes a good geospatial viewer, communicating between the two is not nearly as straightforward as it could be.

This article will demonstrate how to read point and polygon objects from SQL Server and render them in MapPoint. We'll also

demonstrate how to write points and polygons back to SQL Server using the Entity Framework 4.0 included with Visual Studio 2010.

For purposes of illustration, we'll be using the "Al's Beef" Chicago-based company's restaurant locations and hypothetical trade areas. In retail analysis and modeling, trade areas can be defined using various parameters and can be used for different aims. Typically, they're defined as the smallest region around a store that contains areas that meet a specific threshold—for example, where 50 percent or 75 percent of customers live or work. All of the trade areas used in this article were generated using the MapPoint Create Drivetime Zone feature, so they represent a hypothetical trade area based on driving times.

As a chain with fewer than a couple dozen locations, Al's Beef is a relatively small business, but the same concepts and techniques can be applied to large retailers with thousands of locations and in other industries and applications.

Both the sample code and the "Al's Beef" dataset (as a SQL script) are available for download at code.msdn.microsoft.com/mag201009Spatial.

While this isn't an overly technical article covering arcane aspects of the latest language or technology, it serves as a practical how-to for a useful marriage of common Microsoft technologies. A couple of hurdles include the inability of the Entity Framework to directly understand geography objects, and the SQL Server Spatial requirement for polygons to be counterclockwise, which isn't required by MapPoint. Hopefully, this article will help even seasoned developers who may be reluctant to jump into the geospatial arena for lack of prior experience, and at the same time show MapPoint developers how to successfully leverage SQL Server 2008 R2.

This article discusses:

- Setting up a geospatial database
- Using the MapPoint control in a C# Windows Form app
- Using Entity Data Objects
- Adding a MapPoint COM control to the Visual Studio toolbox
- Initializing a map form
- Adding pushpins to a map
- Adding polygons to a map
- Handling MapPoint events
- Expanding the app

Technologies discussed:

SQL Server 2008 R2, Microsoft MapPoint 2010, COM, Entity Framework 4.0, Visual Studio 2010

Code download available at:

code.msdn.microsoft.com/mag201009Spatial

Setting up the Database

In order to follow along with the code samples included here, download the SQL script and run it against SQL Server to set up the database and objects. The data is stored in a SQL Server database called “Corporate” and includes one table, one view and one stored procedure. All Beef locations are stored in a table called “Locations” (see **Figure 1**).

This includes the stores’ addresses, attributes (for example, is it a drive-in?) and a location point geography datatype. The hypothetical trade area polygons are also stored in the “Locations” table, in a field called TradeArea using the geography datatype.

The vLocations view exposes the point and polygon geography fields into datatypes that can be understood and read into the Entity Framework.

The point geography field is decomposed into latitude and longitude fields and passed back to the client as a varbinary field. This is because the Entity Framework can’t deal with geography datatypes directly, but it can handle varbinary fields. The application can later convert these back into geography objects.

Here’s the stored procedure uspAddLocation, which, as the name suggests, is used to insert new locations from MapPoint back into SQL Server:

```
CREATE VIEW [dbo].[vLocations]
AS
SELECT LocID, Location.Long As Longitude,
       Location.Lat As Latitude,
       CAST(Location AS VARBINARY(MAX)) AS Location,
       Locations.TradeArea.STAsText() As TradeAreaWKT,
       CAST(TradeArea AS VARBINARY(MAX)) AS TradeArea
FROM Locations
```

We’ll come back to this later.

Setting up the Application

Our project is a C# Windows Form application that incorporates the MapPoint Control. The control is included with MapPoint 2010, and the full version of MapPoint 2010 must be installed for it to be available. Records can be navigated using buttons to walk through the records and to display the store and its trade area. Stores can also be selected by clicking on the store’s pushpin icon. The form also has a checkbox to display the trade area as a convex hull and a button to add new locations. By default, the application shows the polygon as it is stored in the database (see **Figure 2**).

If the *View Trade Area As Convex Hull* checkbox is set, a line (the convex hull) is wrapped around the trade area similar to wrapping a rubber band around the polygon (see **Figure 3**).

Before we can implement the map display, we need to add Entity Data Objects that point to the database table and view. To establish the Entity Data Objects, right-click on the application in Visual Studio Solution Explorer and go to Add | New Item | Visual C# Items | ADO.NET Entity Data Model. Click Add and choose Generate from Database. In the Choose Your Database Objects dialog, select the table Locations and the view vLocations. After you click Finish, the wizard will create the objects and generate the code necessary to connect to the database.

To add the MapPoint 2010 Control to the Windows Form, it’s necessary to first add the MapPoint COM control component to the Visual Studio toolbox. COM isn’t the most fashionable technology, but it continues to be an important part of the Windows ecosystem. Many applications, including MapPoint, only imple-

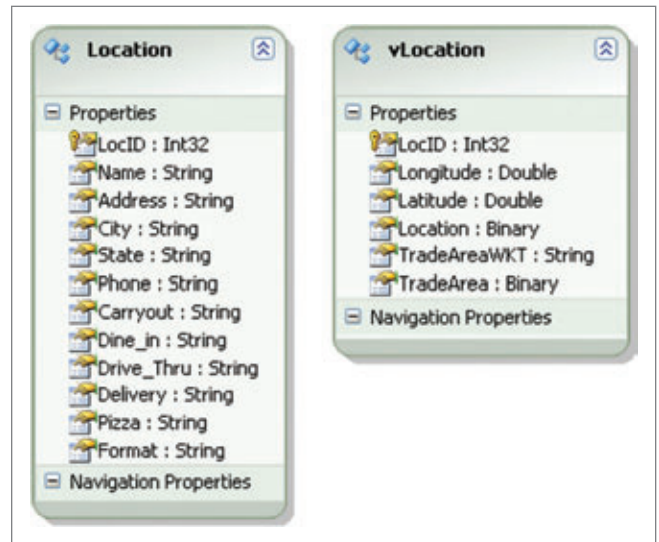


Figure 1 The Table and View Included in the Sample Database

ment an API through a COM interface, and COM support from within Visual Studio isn’t going away anytime soon.

Open the Visual Studio toolbox and in the general section, right-click and select Choose Items. Go to the COM Components tab and select Microsoft MapPoint Control 17.0. The MapPoint Control 17.0 refers to MapPoint 2010 (North America or Europe). Older versions of MapPoint (2002 onward) can also be used, but minor name changes (for example, the toolbar name and symbol identifier) will be required.

On both AxInterop.MapPoint and Interop.MapPoint assemblies, set the Embed Interop Type property to False and the Copy Local to True. The MapPoint Control can now be dragged onto the form and used within the application.

COM isn’t the most fashionable technology, but it continues to be an important part of the Windows ecosystem.

Initializing the Map Form: Loading MapPoint

The Map form declares several member variables, including the database connection to the Entity Framework, a list of the store information and a parallel list of the stores’ geographical information that will be read from the view. The variable curLoc keeps track of the current store ID within the application, and objMap is used to reference the MapPoint control’s map object, as shown here:

```
namespace AlsBeef
{
    public partial class Map : Form
    {
        CorporateEntities db;
        List<Location> locationList;
        List<vLocation> vlocationList;
        int curLoc = -1; // <0 indicates 'not set'
        MapPoint.Map objMap;
        MapPoint.Symbol objSymbol;
        ...
    }
}
```

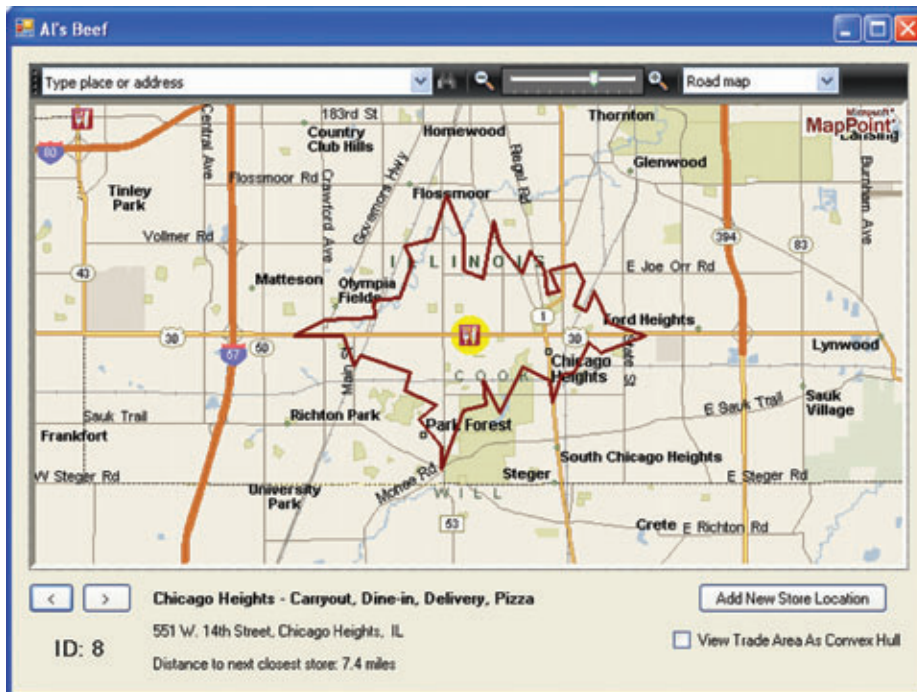


Figure 2 The AI's Beef App, Showing the Chicago Heights Store and Territory as Defined in the Database

When the form is created, the method `CreateNewMapObject` is called to initialize the map control and open a new map using the default North America map template. The toolbars are set, `objMap` is defined and Point of Interest is turned off so as not to clutter the map (see Figure 4). "Points of Interest" are `MapPoint` predefined places, restaurants and theaters, for example.

The form's `Load` method populates both lists of store information. The `locationList` contains all the regular non-geographic information, and `vlocationList` reads the geographic fields as transformed by the database view:

```
private void Map_Load(object sender, EventArgs e)
{
    db = new CorporateEntities();
    locationList = new List<Location>();
    vlocationList = new List<VLocation>();

    ObjectQuery<Location> locationQuery =
        db.Locations;
    ObjectQuery<VLocation> vlocationQuery =
        db.vLocations;

    locationList = locationQuery.ToList();
    vlocationList = vlocationQuery.ToList();

    InitMapSymb();
    InitMapPins();
    SetLocation(0);
}
```

The last two lines effectively start the application by initializing the map. They add a pushpin for each store location (`InitMapPins`), and position the map and form controls to point to the data for the first store location (`SetLocation`).

Adding Pushpins to the Map

Things become more interesting in the `InitMapPins` method:

```
private void InitMapPins()
{
    MapPoint.Pushpin objPin = null;
    for (int i = 0; i < locationList.Count; i++)
    {
        MapPoint.Location objLoc =
            objMap.GetLocation(vlocationList[i].
                Latitude.Value,
                vlocationList[i].Longitude.Value);
        objPin = objMap.AddPushpin(objLoc,
            locationList[i].Name);
        objPin.Symbol = 145; // Red fork and
                                // knife
    }
}
```

Looping over the `locationList`, we retrieve the latitude and longitude values that were calculated and exposed by the view. These are used to create `MapPoint Location` objects, which are then used to create map pushpins (`MapPoint Pushpin` objects). The store name is used for the `PushpinName` property, which will be later used to search and position the

map. The pushpins are depicted with the `MapPoint` built-in red restaurant pushpin symbol (symbol #145). A complete list of `MapPoint` 2010 built-in symbols can be found at mapping-tools.com/info/pushpins/pushpins_2010.shtml. This page also has links to pushpin listings for earlier versions of `MapPoint`.

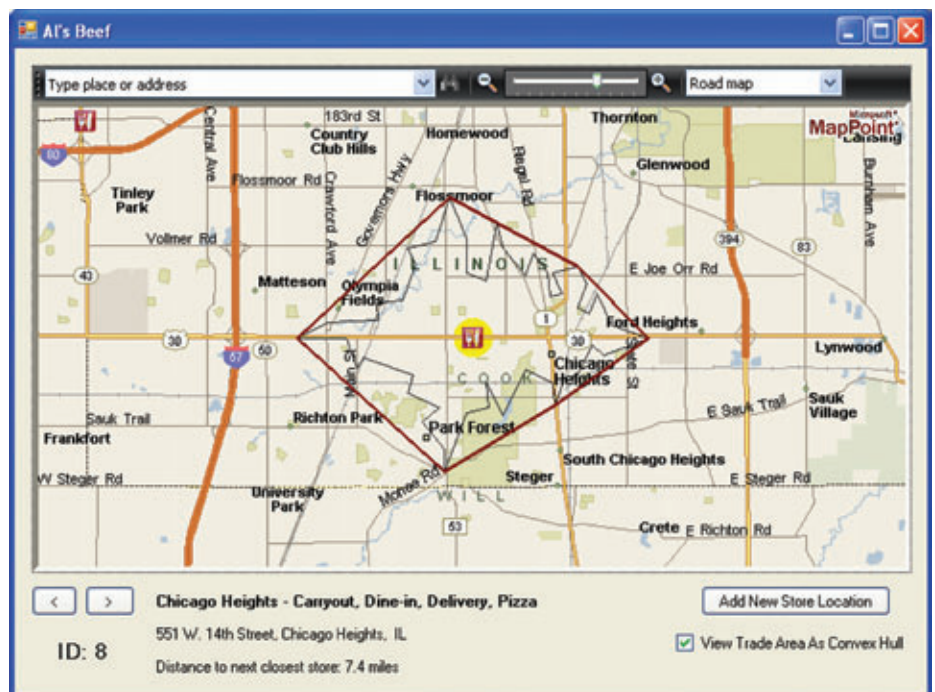


Figure 3 The Chicago Heights Store, with a Convex Hull Wrapped Around the Territory Shown in Figure 2

Positioning to Current Record and Adding Polygons to the Map

New records are selected and displayed using the `IncDecLocation` and `SetLocation` methods. `IncDecLocation` simply applies an increment (`cnt`) to the current position (`curLoc`) and passes this new record position to `SetLocation`:

```
private void IncDecLocation(int cnt = 0)
{
    // Apply the increment/decrement, wrapping around if necessary
    int newLoc = (curLoc + cnt + locationList.Count) % locationList.Count;

    SetLocation(newLoc);
}
```

The `SetLocation` routine is the workhorse of the application. `SetLocation` selects a new record position and displays it on the map. `SetLocation` also removes the highlight from the previous pushpin (if any) and clears all previous trade area polygons from the map (see **Figure 5**).

Geometry data is defined in a flat (2D Euclidean) Cartesian plane, whereas geography data is projected onto the spheroidal Earth surface using a spherical coordinate system.

The next section becomes a little tricky. First, the application checks the status of the *View Trade Area As Convex Hull* checkbox. If it hasn't been set, it takes the Well-Known Text (WKT) string that defines the polygon and passes it to the custom `RenderPolygon` method to be parsed and rendered as a polygon on the map.

If the checkbox has been set, it pulls the territory polygon's varbinary object and converts it to a geography object using the `System.IO.MemoryStream` class and `BinaryReader` method. `STConvexHull` is one of the methods included with SQL Server 2008; it allows you to modify instances of geography or geometry data. `STConvexHull`, in particular, only works with geometry datatypes. The differences between SQL Server's geometry and geography datatypes is covered extensively elsewhere, but for now consider that geometry data is defined in a flat (2D Euclidean) Cartesian plane, whereas geography data is projected onto the spheroidal Earth surface using a spherical coordinate system (datum, projection, prime meridian and unit of measurement).

The trade area is stored in the database with a geography field type and is then cast into a varbinary by the view. This needs to be read into a geography object, which can then be converted into a geometry object in order to run the `STConvexHull` method.

Because of the small areas being covered, calculations performed by `STConvexHull` on the (planar) geometry object are practically the same as would have resulted had the convex hull been calculated for the actual spheroidal geography object.

Figure 4 Creating a Form

```
public Map()
{
    InitializeComponent();
    CreateNewMapObject();
}

private void CreateNewMapObject()
{
    MPctrl1.NewMap(GeoMapRegion.geoMapNorthAmerica);
    object barObj = "advanced";
    MPctrl1.Toolbars.get_Item(refbarObj).Visible = true;
    MPctrl1.Toolbars.LargeToolBarButtons = false;
    objMap = MPctrl1.ActiveMap;
    // Make sure all points of interest are turned off
    objMap.PlaceCategories.Visible = MapPoint.GeoTriState.geoFalse;
}
```

Figure 5 The `SetLocation` Routine Is the App Workhorse

```
private void SetLocation(int newLoc)
{
    MapPoint.Pushpin objPin = null;

    // Unhighlight previous pushpin
    If (curLoc >= 0)
    {
        objPin = (MapPoint.Pushpin)
        objMap.FindPushpin(locationList[curLoc].Name);
        objPin.Highlight = false;
    }

    // Clear all previous shapes
    while(objMap.Shapes.Count > 0)
    {
        objMap.Shapes[1].Delete();
    }

    // Set the new location
    curLoc = Math.Min( Math.Max(newLoc,0), locationList.Count-1);

    objPin = (MapPoint.Pushpin)
    objMap.FindPushpin(locationList[curLoc].Name);
    objMap.Location = objPin.Location;

    ...
}
```

Figure 6 Drawing the Original Trade Area and the Convex Hull

```
...
// Draw trade area
if (checkBox1.Checked == false)
{
    RenderPolygon(vlocationList[curLoc].TradeAreaWKT);
}
else
{
    // Need to add C:\Program Files\Microsoft SQL
    // Server\100\SDK\Assemblies\Microsoft.SqlServer.Types.dll
    // to references
    SqlGeographyTradeAreaGeog = new SqlGeography();
    using (var stream = new
        System.IO.MemoryStream(vlocationList[curLoc].TradeArea))
    {
        using (var rdr = new System.IO.BinaryReader(stream))
        {
            TradeAreaGeog.Read(rdr);
        }
    }
    SqlGeometry TAConvexHullGeom = new SqlGeometry();
    TAConvexHullGeom =
        SqlGeometry.STPolyFromText(TradeAreaGeog.STAsText(), 4326);
    TAConvexHullGeom = TAConvexHullGeom.STConvexHull();
    RenderPolygon(TradeAreaGeog.ToString(), 3355443, 0); // Gray80
    RenderPolygon(TAConvexHullGeom.ToString());
}
...
```


Figure 7 TheRenderPolygon Method

```
private void RenderPolygon(string polystring,
    int forecolor = 128, int weight = 3)
{
    polystring = polystring.Replace("POLYGON (", "");
    polystring = polystring.Replace(")", "");
    string[] stringList = polystring.Split(',');
    MapPoint.Location[] objLoc =
        new MapPoint.Location[stringList.Count()];
    for (int i = 0; i < stringList.Count(); i++)
    {
        string[] coords = stringList[i].Trim().Split(' ');
        objLoc[i] = objMap.GetLocation(Convert.ToDouble(coords[1]),
            Convert.ToDouble(coords[0]), 0);
    }
    MapPoint.Shape objShape;
    objShape = objMap.Shapes.AddPolyline(objLoc);
    objShape.Line.ForeColor = forecolor;
    objShape.Line.Weight = weight;
}
```

In the next part of SetLocation, the original trade area is drawn as a thin black line and the convex hull is rendered as a thicker red line. The code is shown in **Figure 6**.

So what does this WKT string look like and what does RenderPolygon do with it? You've already seen the results (in **Figures 2** and **3**). Let's dive into the internals.

WKT is an Open Geospatial Consortium (OGC) standard format for formatting geospatial vector data in the form of text. A WKT polygon string looks like this (much abbreviated):

```
POLYGON ((-111.918823979795 33.6180476378649, -111.91810682416
33.6096635553986, -111.911686453968 33.6078672297299, -111.907403888181
33.599476357922, -111.907403888181 33.6060674674809, -111.903121406212
33.6060674674809))
```

The word "POLYGON" precedes a list of coordinates that are surrounded by two sets of parentheses. Individual coordinate pairs are separated by commas. We use the MapPoint AddPolyLine method to draw the polygon on the map and to add it to the MapPoint Shapes collection. This takes an array of MapPoint Location objects as a parameter. Converting the WKT string to an array of Location objects requires half a dozen lines of code. RenderPolygon performs this by stripping the "POLYGON" prefix and parentheses before splitting the string into coordinates using

the comma separator. Individual coordinates are then parsed into pairs of doubles (longitude, latitude) that are used to create MapPoint Location objects. The resulting array of Location objects is then passed to AddPolyline to create the new polygon.

RenderPolygon takes additional parameters for the color and line thickness (see **Figure 7**).

A more complete RenderPolygon could take additional parameters for whether or not the shape is filled in, the fill color, a shape name (an internal string that can be assigned to shapes) and the zOrder (to position the shape in front of or behind roads and other shapes).

Drawing and annotation can be placed on a MapPoint map by both the user and a program. MapPoint supports a total of 40 different colors for this annotation. Although the programming interface appears to support the standard 3-byte RGB (16,777,216 different) colors, in reality these numbers merely provide a useful way to specify the color to use. The 40 colors supported by MapPoint can be seen at mapping-tools.com/info/pushpins/colors.shtml.

Historically, this limitation helped with image update efficiency, but today it primarily aids map clarity by helping to ensure colors are different.

We now come to the final part of SetLocation (see **Figure 8**).

WKT is an Open Geospatial Consortium standard format for formatting geospatial vector data in the form of text.

This highlights the new pushpin, sets the zoom level (using the Map object Altitude property), reports the store information (from the locationList array) and finds the distance to the nearest store location.

This distance is calculated by NearestLocation. This loops through the locations and uses the SQL Server Spatial STUnion method to

Figure 8 The Final Part of SetLocation

```
...
// Reset zoom level
objMap.Altitude = 30;
objPin.Highlight = true;

Double distance;
distance =
    NearestLocation(curLoc) * 0.000621371192; //convert to miles

label1.Text = "ID: " + locationList[curLoc].LocID.ToString();
label2.Text = locationList[curLoc].Name + " - " +
    locationList[curLoc].Format;
label3.Text = locationList[curLoc].Address + ", " +
    locationList[curLoc].City + ", " + locationList[curLoc].State;
label4.Text = "Distance to next closest store: " +
    String.Format("{0:#,0.0}", distance) + " miles";
}

private double NearestLocation(int curLoc)
{
    SqlGeography AllLocations = new SqlGeography();
```

```
SqlGeography CurLocation = new SqlGeography();
for (int i = 0; i < locationList.Count; i++)
{
    SqlGeography TempLocation = new SqlGeography();
    using (var stream = new
        System.IO.MemoryStream(vlocationList[i].Location))
    {
        using (var rdr = new System.IO.BinaryReader(stream))
        {
            TempLocation.Read(rdr);
        }
    }
    if (i == curLoc)
    {
        CurLocation = TempLocation;
    }
    else
    {
        AllLocations = AllLocations.STUnion(TempLocation);
    }
}
return (Double)AllLocations.STDistance(CurLocation); //meters
}
```

Figure 9 Adding Handlers to MapPoint Events

```
//  
// MPctrl  
//  
this.MPctrl.Enabled = true;  
this.MPctrl.Location = new System.Drawing.Point(13, 13);  
this.MPctrl.Name = "MPctrl";  
this.MPctrl.OcxState =  
    ((System.Windows.Forms.AxHost.State)  
    (resources.GetObject("MPctrl.OcxState")));  
this.MPctrl.Size = new System.Drawing.Size(674, 389);  
this.MPctrl.TabIndex = 0;  
this.MPctrl.SelectionChange +=  
    new AxMapPoint._IMappointCtrlEvents_SelectionChangeEventHandler  
    (this.MPctrl_SelectionChange);  
this.MPctrl.BeforeClick +=  
    new AxMapPoint._IMappointCtrlEvents_BeforeClickEventHandler  
    (this.MPctrl_BeforeClick);
```

combine the Location geography points into a MultiPoint geography instance. The exception is the current store location, which is skipped—otherwise the distance would always be zero miles! The application then uses the `STDistance` method to calculate the distance in meters between the current store location and the MultiPoint geography instance. `STDistance` reports the distance to a MultiPoint as the shortest distance to any component point within the MultiPoint.

The button to add a new site location removes any trade-area polygons from the map and then simply changes the mouse pointer to a crosshair:

```
private void button1_Click(object sender, EventArgs e)  
{  
    // Clear all previous shapes  
    while(objMap.Shapes.Count > 0)  
    {  
        objMap.Shapes[1].Delete();  
    }  
    MPctrl.MousePointer = MapPoint.GeoPointer.geoPointerCrosshair;  
}
```

In order to deal with MapPoint events, the form requires an event handler defined in the form designer. The events can be added using the form designer or added manually to `Map.Designer.cs`. Handlers are added to two of the MapPoint events: `SelectionChange` and `BeforeClick`, as shown in **Figure 9**.

The `SelectionChange` event is used to detect if the user has selected a pushpin. This is then used to move the current record to this pushpin's record. **Figure 10** shows the event handler's implementation.

This checks that the newly selected object is, indeed, a pushpin. Then we perform a simple search through the local `locationList` for a matching record. MapPoint pushpins can have duplicate names, so this code assumes that all location records (and hence pushpins) have unique names. You should also compare geographic coordinates for situations where this can't be relied upon.

The map's `BeforeClick` event handler is used in the "add new store location" functionality. The handler checks to see if the mouse pointer is a crosshair—that is, the user is trying to insert a new site location. It lets MapPoint handle the click event if the pointer isn't a crosshair. If the mouse pointer is a crosshair, the program traps the click action and adds a new pushpin at the mouse pointer location using the red restaurant symbol. At this point, to simplify things, rather than have the user draw the trade area, the program uses the MapPoint `AddDrivetimeZone` method to generate a hypothetical (travel-time-based) trade area around the new site.

In the interest of getting this shape into SQL Server, the shape is first decomposed into vertices, which are then converted into a polygon WKT (text) definition. This will then be written to SQL Server.

To pass the point and polygon back into SQL Server and update the geography columns, we can't use the normal Entity Framework-supported stored procedures, because the geography datatype isn't supported. However, because execution of arbitrary stored procedures is now supported by Entity Framework 4.0, we are able to import the stored procedure and execute it like a normal function.

This code sets up the parameters and then executes the `uspAddLocation` stored procedure:

```
object[] parameters =  
{  
    new SqlParameter("Latitude",objLoc.Latitude),  
    new SqlParameter("Longitude",objLoc.Longitude),  
    new SqlParameter("PolyWKT",PolyWKT)  
};  
  
var retValue = db.uspAddLocation(objLoc.Longitude,  
    objLoc.Latitude, PolyWKT);
```

Drawing and annotation can be placed on a MapPoint map by both the user and a program.

Last, this routine resets the map (`CreateNewMapObject`), requeries the list of locations from the database (`InitMapPins`) and selects the new store as the current record (`SetLocation`):

```
// Re-query and re-initialize map  
ObjectQuery<Location> locationQuery = db.Locations;  
ObjectQuery<vLocation> vlocationQuery = db.vLocations;  
locationList = locationQuery.ToList();  
vlocationList = vlocationQuery.ToList();  
objMap.Saved = true;  
CreateNewMapObject();  
InitMapSymb();  
InitMapPins();  
SetLocation( locationList.Count - 1 );  
e.cancel = true;  
}
```

The line `e.cancel=true`; prevents MapPoint from further processing the click event. The stored procedure `uspAddLocation` follows (see **Figure 11**).

Figure 10 Implementing a SelectionChange Event Handler

```
private void MPctrl_SelectionChange(object sender,  
    AxMapPoint._IMappointCtrlEvents_SelectionChangeEvent e)  
{  
    // Has the user just selected a pushpin?  
    if (e.pNewSelection is MapPoint.Pushpin)  
    {  
        MapPoint.Pushpin ppin = e.pNewSelection as MapPoint.Pushpin;  
  
        // Find the corresponding location object, and select it  
        for (int iloc = 0; iloc < locationList.Count; iloc++)  
        {  
            if (locationList[iloc].Name == ppin.Name)  
            { // Found it: select, and move to it  
                SetLocation(iloc);  
                break;  
            }  
        }  
    }  
}
```

Figure 11 The `uspAddLocation` Stored Procedure

```
CREATE PROCEDURE [dbo].[uspAddLocation]
@Longitude FLOAT,
@Latitude FLOAT,
@PolyWKT NVARCHAR(MAX)
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    DECLARE @NewLocID int = 0
    SELECT @NewLocID = MAX(LocID)+1 FROM Locations

    DECLARE @NewPoly geography
    SET @NewPoly = geography::STPolyFromText(@PolyWKT,4326)

    INSERT INTO Locations(LocID,Name,Address,City,State,Format,Location,TradeArea)
    VALUES(@NewLocID, 'New Location ' + CAST(@NewLocID As varchar(3)), '123 Main',
    'Anywhere', 'ST', 'Food', geography::Point(@Latitude,@Longitude,4326),
    @NewPoly)

    SELECT @NewLocID AS NewLocID

END
```

You can see a new geography instance and variable being created for the polygon before the INSERT statement, whereas the Point location is created via the Point method inside the INSERT method. Either approach is valid.

The event-handling code in **Figure 12** handles the “previous” and “next” buttons, the convex hull checkbox and form closing—this completes the application.

Viewing and Visually Editing SQL Server Geospatial Data

This article covered a lot of ground, but it demonstrates a complete end-to-end application with SQL Server, showing how to pass data in and out using both SQL Server Spatial and MapPoint methods to display and edit real-world information.

The principles shown here can be taken further. Keeping a local copy of the data ensures fast map updates but wouldn't be practical for huge datasets. For these situations, the data should be fetched on a record-by-record basis, ideally with a caching mechanism.

MapPoint is capable of some useful geospatial operations (for example, the drive-time zone calculation), but it lacks many of the

geospatial operations expected of a full GIS. Here we leverage two such operations, `STConvexHull` and `STDistance`, from the SQL Server Spatial extensions. Other advanced functions available in the spatial extensions include the ability to measure the length and extent of geographic features, as well as finding unions and intersections of polygons. These functions could be used to create a sophisticated territory-management application. This could combine territories or find overlaps where one store is cannibalizing the business of another.

Similarly, MapPoint's strengths could be leveraged. MapPoint is capable of offline geocoding. Our example uses existing coordinates, but the MapPoint geocoder could be used to locate street addresses instead. MapPoint also ships with a number of demographic databases at the county, ZIP code and census tract levels. This data could be plotted on a store map, allowing easy comparisons to be made—for example, how store sales compare to local populations and income levels.

To pass the point and polygon back into SQL Server and update the geography columns, we can't use the normal Entity Framework-supported stored procedures, because the geography datatype isn't supported.

Looking ahead, SQL Server Spatial is likely to have a generational leap forward with the next version of SQL Server, and the MapPoint product has been enjoying a renaissance of new development and features with the last two versions, and this is set to continue. Furthermore, the Entity Framework is likely to continue to add support for new field types, including spatial datatypes, which should improve communication between SQL Server and MapPoint. Taken together, these technologies form a robust and powerful evolving platform for mapping application development. ■

ERIC FROST is a Microsoft MVP and business application developer specializing in GIS/mapping applications. He manages the active Microsoft mapping technology forum mapforums.com and can be reached at eric.frost@mp2kmag.com.

RICHARD MARSDEN is a Microsoft MVP and freelance software developer. He sells a number of MapPoint extensions at mapping-tools.com and operates the GeoWeb Guru Web site at geowebguru.com.

THANKS to the following technical experts for reviewing this article:
Bob Beauchemin, Ed Katibah and Amar Nityananda

Figure 12 Completing the App

```
private void prev_Click(object sender, EventArgs e)
{
    IncDecLocation(-1);
}

private void next_Click(object sender, EventArgs e)
{
    IncDecLocation(1);
}

private void checkBox1_CheckedChanged(object sender, EventArgs e)
{
    IncDecLocation();
}

private void Map_FormClosing(object sender, FormClosingEventArgs e)
{
    db.Dispose();
    MPctrl.ActiveMap.Saved = true;
}
```


JUST
HOW **EPIC**

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



It's so powerful it makes the Kraken look like a gimp sea monkey.
It's so fast it makes warp drive look like a hobbit running backwards.

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

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

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

Windward Reports hasn't simply shifted the reporting paradigm,

IT HAS NINJA ROUND HOUSE KICKED IT ALL THE WAY INTO AN ENTIRELY DIFFERENT DIMENSION.

IF YOU FIND ALL OF THIS HARD TO SWALLOW,
D O W N L O A D T H E F R E E T R I A L A T
www.WindwardReports.com/msdn.aspx
AND SEE FOR YOURSELF. **

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

the ONLY **EPIC** REPORTING & DOCUMENT GENERATING SOLUTION



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

(303) 499-2544

A Visual Studio 2010 Q&A with David Thielen of Windward Reports



Windward Reports makes enterprise reporting software that empowers companies to deliver better reports, faster. Using MS Word, Excel or PowerPoint as its design tool, Windward enables the user to create custom reports exactly as they envisioned. Windward delivers powerful reporting for Corporate or OEM Use.

A It needs to be simple, so that you don't have to do a lot of work to integrate it into a solution. It needs to be flexible and powerful enough so that you can rely on it for any project and any kind of report. Windward has a very simple API and a small but expressive set of tags. Designing in Office reduces complexity in sophisticated layouts.

A At an earlier job, I was given a set of sample reports by a Program Manager. Looking at the sample output in Word and Excel, I thought, "why can't I just push a

A We use Office (Word, Excel, PowerPoint) as the design tool. The layout, formatting, design—that's all done in Office. Windward benefits because we don't have to create a report designer, users benefit because they have the most powerful and easy to use document designer on the planet—that they already know how to use. We also have an Office Add-In called AutoTag. It provides a straightforward way to select data and place it in the template. Because the tags, including the selects, are placed right in the template, there is no programming involved.

A Actually, we have fewer limits than other reporting systems. A template is a free-form document where tags can be inserted anywhere to work in any manner. So N rows of data are not limited to a table, they can actually be a combination of text, charts, imported sub-reports, and/or tables, repeating once for each row of data.

A Age of Kings. Speed with the mouse is everything in Starcraft and so I'll never beat our interns at it. Age of Kings has a significant strategic component and I love the look on the face of some of the younger employees here when they realize someone my age just creamed them.

[illegible]



Request-Response Testing Using IronPython

I'm a big fan of using the Python programming language for several types of lightweight test automation tasks. In this month's column, I show you how to use IronPython—a Microsoft .NET Framework-compliant implementation of Python—to perform HTTP request-response testing for ASP.NET Web applications.

Specifically, I create a short test harness script that simulates a user exercising an ASP.NET application. The IronPython harness programmatically posts HTTP request information to the application on a Web server. It then fetches the HTTP response stream and examines the HTML text for an expected value of some sort to determine a pass/fail result. In addition to being a useful testing technique in its own right, learning how to perform HTTP request-response testing with IronPython is an excellent way to learn about the IronPython language.

This column assumes you have basic familiarity with ASP.NET technology and intermediate scripting skills with a language such as JavaScript, Windows PowerShell, VBScript, Perl, PHP or Ruby, but I don't assume you have any experience with Python. However, even if you're new to ASP.NET and scripting, you should still be able to follow the column without too much difficulty. The best way to see where I'm headed is to examine the screenshots in **Figures 1** and **2**.

Figure 1 illustrates the example ASP.NET Web application under test. The system under test is a simple but representative Web application named MiniCalc. I deliberately kept my ASP.NET Web application under test as simple as possible so that I don't obscure the key points in the IronPython test automation. Realistic Web applications are significantly more complex than the dummy MiniCalc application shown in **Figure 1**, but the IronPython testing technique I describe here easily generalizes to complex applications. The MiniCalc Web application accepts two numeric values, an indication to add or multiply the values, and the number of decimals to display the answer to. The app then sends the values to a Web server, where the result is computed. The server creates the HTML response and sends it back to the client browser, where the result is displayed to the number of decimal places specified by the user.

Figure 2 shows an IronPython test harness in action. My script is named `harness.py` and doesn't accept any command-line arguments. For simplicity, I have hard-coded information, including the URL of the Web application and the name of the test case input file. My harness begins by echoing the target URL of `http://localhost/MiniCalc/Default.aspx`. In test case 001, my harness programmatically posts information that corresponds to a user typing 1.23 into control `TextBox1`, typing 4.56 into `TextBox2`,

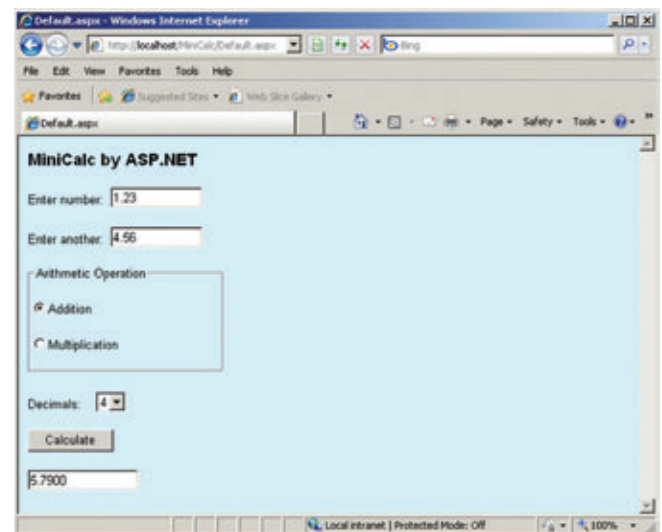


Figure 1 MiniCalc Web App Under Test

selecting `RadioButton1` (to indicate an addition operation), selecting 4 from the `DropDownList1` control and clicking on `Button1` (to calculate). Behind the scenes, the harness captures the HTTP response from the Web server and then searches the response for an indication that 5.7900 (the correct sum of 1.23 and 4.56 to 4 decimals) is in the `TextBox3` result control. The harness tracks the number of test cases that pass and the number that fail and displays those results after all test cases have been processed.

In the sections that follow, I briefly describe the Web application under test so you'll know exactly what's being tested. Next, I walk you through the details of creating lightweight HTTP request-response automation using IronPython. I wrap up by presenting a few opinions about when the use of IronPython is suitable and when other techniques are more appropriate. I think you'll find the information interesting, and a useful addition to your testing toolset.

The Application Under Test

Let's take a look at the code for the MiniCalc ASP.NET Web application, which is the target of my test automation. I created the app using Visual Studio 2008. After launching Visual Studio with administrator privileges, I clicked on `File | New | Web Site`. In order

Code download available at code.msdn.microsoft.com/mag201009TestRun.


```

C:\RequestResponseTestingUsingPython>ipy.exe harness.py
Begin IronPython Request-Response testing
URL under test = http://localhost/MiniCalc/Default.aspx

=====
Test case: 001
Input   : TextBox1=1.23&TextBox2=4.56&Operation=RadioButton1&DropDownList1=4&Button1=c clicked
Expected: value="5.7900" id="TextBox3"
Pass
=====

Test case: 002
Input   : TextBox1=1.23&TextBox2=4.56&Operation=RadioButton2&DropDownList1=4&Button1=c clicked
Expected: value="5.7900" id="TextBox3"
**Fail**
=====

Test case: 003
Input   : TextBox1=2.00&TextBox2=3.00&Operation=RadioButton1&DropDownList1=4&Button1=c clicked
Expected: value="5.0000" id="TextBox3"
Pass
=====

Number pass = 2
Number fail = 1

End test run

C:\RequestResponseTestingUsingPython>

```

Figure 2 Testing the App Using IronPython

to avoid the ASP.NET code-behind mechanism and keep all the code for my Web application in a single file, I selected the Empty Web Site option. I specified `http://localhost/MiniCalc` in the location field. I decided to use C# for the MiniCalc application, but the test harness I'm presenting in this column works with ASP.NET applications written in VB.NET, and with slight modifications the harness can target Web applications written using technologies such as classic ASP, CGI, PHP, JSP and Ruby. I clicked OK on the New Web Site dialog to generate the structure of my Web application. Next, I went to the Solution Explorer window, right-clicked on the MiniCalc project name and selected Add New Item from the context menu. I then selected Web Form from the list of installed templates and accepted the Default.aspx file name. I cleared the "Place code in separate file" option and then clicked the Add button. Next, I double-clicked on the Default.aspx file name in Solution Explorer to edit the template-generated code. I deleted all the template code and replaced it with the code shown in **Figure 3**.

To keep my source code small in size and easy to understand, I'm taking shortcuts such as not performing error-checking and combining server-side controls (such as `<asp:TextBox>`) with plain HTML (such as `<fieldset>`). The most important parts of the code in **Figure 3** are the IDs of the ASP.NET server-side controls. I use default IDs Label1 (user prompt), TextBox1 and TextBox2 (input for two numbers), RadioButton1 and RadioButton2 (choice of addition or multiplication), DropDownList1 (number of decimals for the result), Button1 (calculate) and TextBox3 (result). To perform automated HTTP request-response testing for an ASP.NET application using the technique I present here, you must know the IDs of the application's controls. In this situation, I have the source code available because I'm creating the application myself; but even if you're testing a Web application you didn't write, you can always examine the application by using a Web browser's View Source functionality.

To verify that my Web application under test was built correctly, I hit the <F5> key. I clicked OK on the resulting Debugging Not

Enabled dialog to instruct Visual Studio to modify the Web application's Web.config file. Visual Studio then launched Internet Explorer and loaded MiniCalc. Notice that the action attribute of my <form> element is set to Default.aspx. In other words, every time a user submits a request, the same Default.aspx page code is executed. This gives my MiniCalc Web application the feel of a single application rather than a sequence of different Web pages. Because HTTP is a stateless protocol, ASP.NET accomplishes the application effect by maintaining the Web application's state in a special hidden value type, called the ViewState. As you'll see shortly, dealing with an ASP.NET appli-

cation's ViewState is one of the keys to programmatically posting data to the application.

ASP.NET Request-Response Testing with IronPython

Let's go over the IronPython test harness program that produced the screenshot in **Figure 2**. IronPython is a free download available from CodePlex, the Microsoft-sponsored open source project, at ironpython.codeplex.com. I'm using version 2.6.1, which runs on version 2.0 of the .NET Framework and runs on any machine that supports that version of the Framework. The overall structure of my test harness script is presented in **Figure 4**.

As you can see, my harness script is simple and is driven by an external test case data file. That test case data file is named `testCases.txt` and consists of:

```

001|1.23|4.56|RadioButton1|4|clicked|5.7900
002|1.23|4.56|RadioButton2|4|clicked|5.7900
003|2.00|3.00|RadioButton1|4|clicked|5.0000

```

Each line represents one test case and has seven fields delimited by a "|" character. The first field is a test case ID. The second and third fields are inputs to TextBox1 and TextBox2. The fourth field encodes whether to request Addition or Multiplication. The fifth field is the value for the Decimals DropDownList control. The sixth field ("clicked") is the Button1 event. The seventh field is the expected result, which should appear in TextBox3. The second test case is deliberately incorrect just to demonstrate a test case failure. For the lightweight testing approach I'm describing here, a simple text file to hold test case data is often a good choice. If I had wanted to embed my test case data directly in the harness script, I could have done so using an array of strings like:

```

testCases = ['001|1.23|4.56|RadioButton1|4|clicked|5.7900',
             '002|1.23|4.56|RadioButton2|4|clicked|5.7900',
             '003|2.00|3.00|RadioButton1|4|clicked|5.0000']

```

and then iterated through each test case like:

```

for line in testCases:
    ...

```

Python also has a list type that can be used to store test case data.

The first three lines of my IronPython test harness are:

```
# harness.py
import sys
import clr
```

Comments in Python begin with the “#” character and extend to end of line. The “import sys” statement allows my script to access resources in the special IronPython sys module.

The locations of these resources can be listed by issuing a sys.path command from the IronPython interactive console. The “import clr” statement allows my script to access and use core .NET CLR functionality.

Figure 3 MiniCalc Code

```
<%@ Page Language="C#" %>
<script runat="server">
    private void Button1_Click(object sender, System.EventArgs e)
    {
        double alpha = double.Parse(TextBox1.Text.Trim());
        double beta = double.Parse(TextBox2.Text.Trim());

        string formatting = "F" + DropDownList1.SelectedValue;

        if (RadioButton1.Checked)
            TextBox3.Text = Sum(alpha, beta).ToString(formatting);
        else if (RadioButton2.Checked)
            TextBox3.Text = Product(alpha, beta).ToString(formatting);
        else
            TextBox3.Text = "Select method";
    }
    private static double Sum(double a, double b)
    {
        return a + b;
    }
    private static double Product(double a, double b)
    {
        return a * b;
    }
</script>

<html>
<head>
    <style type="text/css">
        fieldset { width: 16em }
        body { font-size: 10pt; font-family: Arial }
    </style>
    <title>Default.aspx</title>
</head>
<body bgColor="#ccffff">
    <h3>MiniCalc by ASP.NET</h3>
    <form method="post" name="theForm" id="theForm" runat="server"
        action="Default.aspx">
        <p><asp:Label id="Label1" runat="server">Enter number:</asp:Label>
        <asp:TextBox id="TextBox1" width="100" runat="server" /></p>
        <p><asp:Label id="Label2" runat="server">Enter another:</asp:Label>
        <asp:TextBox id="TextBox2" width="100" runat="server" /></p>
        <p></p>
        <fieldset>
            <legend>Arithmetic Operation</legend>
            <p><asp:RadioButton id="RadioButton1" GroupName="Operation"
                runat="server"/>Addition</p>
            <p><asp:RadioButton id="RadioButton2" GroupName="Operation"
                runat="server"/>Multiplication</p>
            <p></p>
        </fieldset>
        <p>Decimals: &nbsp;<asp:DropDownList ID="DropDownList1" runat="server">
            <asp:ListItem>3</asp:ListItem>
            <asp:ListItem>4</asp:ListItem>
        </asp:DropDownList>
        <p></p>
        <p><asp:Button id="Button1" runat="server" text=" Calculate "
            onclick="Button1_Click" /></p>
        <p><asp:TextBox id="TextBox3" width="120" runat="server" />
        </p>
    </form>
</body>
</html>
```

My next six statements explicitly enable the .NET functionality my harness uses:

```
from System import *
from System.IO import *
from System.Text import *
from System.Net import *
clr.AddReference('System.Web')
from System.Web import *
```

The first line here imports System and is similar to the “using System” statement in a C# program. The “import clr” statement includes the System namespace, so I could omit the “from System import*” statement but I prefer to leave it in as a form of documentation. The next three statements bring the System.IO (for file operations), System.Text (for byte conversion) and System.Net (for request-response functionality) namespaces into scope. The “clr.AddReference(‘System.Web’)” statement brings the System.Web namespace into scope. This namespace isn’t directly accessible by default, so I must use the AddReference method before I issue the “from System.Web import*” statement so I can access URL-encoding methods.

Next, I define a helper method to fetch the ViewState information for the Web application under test:

```
def getVS(url):
    wc = WebClient()
    bytes = wc.DownloadData(url)
    html = Encoding.ASCII.GetString(bytes)
    start = html.IndexOf('id="__VIEWSTATE"', 0) + 24
    end = html.IndexOf('\"', start)
    vs = html.Substring(start, end-start)
    return vs
```

Remember that because HTTP is a stateless protocol, ASP.NET gives the effect of being a stateful application by maintaining the application’s state in a hidden value called ViewState. A ViewState value is a Base64-encoded string that maintains the state of an ASP.NET page through multiple request/response roundtrips. Similarly, an EventValidation value was added in ASP.NET 2.0 and is used for security purposes to help prevent script-insertion attacks. These two mechanisms are key to programmatically posting data to an ASP.NET Web application.

In Python, you must define functions before you call them in a script. Functions are defined using the def keyword. The helper function first instantiates a WebClient object. Next the DownloadData method sends an HTTP request to the Web application given by parameter url, and fetches the HTTP response as an array of byte values. I use the GetString method to convert bytes to a string named html. A ViewState element looks like this:

```
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
value="/wEPDwULLTEwNjU=" />
```

So, to extract the value, I first determine the location of the substring id="__VIEWSTATE" and then add 24 characters. This approach is brittle in the sense that the technique will break if ASP.NET changes the format of the ViewState string, but because this is lightweight automation, simplicity trumps robustness. I determine where the terminating double-quote character is and then I can use the Substring method to extract the ViewState value. Unlike most languages, which use tokens such as begin...end or {...} to delimit code blocks, Python uses indentation. If you are new to Python programming, this may seem odd at first, but most engineers I’ve talked to say they quickly become used to the syntax. Python is supported by a collection of modules, so an alternative to using .NET methods to retrieve the ViewState value is to use functions in the native Python urllib module.

After defining the getVS helper function, I define a helper function to get the EventValidation value:

```
def getEV(url):
    wc = WebClient()
    bytes = wc.DownloadData(url)
    html = Encoding.ASCII.GetString(bytes)
    start = html.IndexOf('id="__EVENTVALIDATION"', 0) + 30
    end = html.IndexOf("'", start)
    ev = html.Substring(start, end-start)
    return ev
```

I use the same technique to extract EventValidation as I do to extract ViewState. Notice that Python is a dynamically typed language, so I don't specify the data types of my parameters, variables and objects. For example, DownloadData returns a byte array and IndexOf returns an int, and the IronPython interpreter will figure these types out for me. Defining two functions, getVS and getEV, requires two roundtrip calls to the Web application under test, so you may want to combine the two helper functions into a single function and name the helper something like getVSEV(url).

After defining my helper functions, my actual automation begins:

```
try:
    print '\nBegin IronPython Request-Response testing'
    url = 'http://localhost/MiniCalc/Default.aspx'
    print '\nURL under test = ' + url + '\n'
    testCases = 'testCases.txt'
    ...
```

Unlike some languages that require an entry point, such as a Main method, Python script execution simply begins with the first executable statement. I use the try keyword to catch any exceptions, then I print a startup message. Python allows the use of either single quotes or double quotes to define string literals, and escape sequences such as \n can be embedded with either delimiter. To disable escape sequence evaluation, you can pre-pend string literals with lowercase r ("raw"), for example: file = r'\newFile.txt'. I hard-code the URL of the application under test and display that value to the shell. If I wanted to read the URL from the command line, I could have used the built-in sys.argv array, for example: url = sys.argv[1]. Python uses the "+" character for string concatenation. I also hard-code the name of my test case data file, and because I don't include file path information, I make the assumption that the file is in the same directory as the IronPython script.

Figure 4 Test Harness Structure

```
set up imports
define helper functions

try:
    initialize variables
    open test case data file
    loop
        read a test case from file
        parse test case data
        determine ViewState
        determine EventValidation
        construct request string
        send request string to app
        fetch response from app
        determine if response has expected result
        print pass or fail
    end loop
    close test case data file
    print summary results
except:
    handle exceptions
```

Next, I set up counters, open my test case data file and start iterating through the file:

```
...
numPass = numFail = 0
fin = open(testCases, 'r')
for line in fin:
    print '-----'
    (caseid,value1,value2,operation,decimals,action,expected) =
        line.split('|')
    ...
```

Python has several idioms I really like, including multiple variable assignment and concise file operations syntax. The "r" argument in the call to the open function means to open the file for reading. The "for line in fin" statement enumerates through the file one line at a time, assigning the current line of input to variable "line." Another neat Python construct is the tuple idiom. Tuples are denoted using left and right parentheses and the tuple values are delimited by "," characters. Here, I call the split method and assign each resulting token to variables caseid, value1, value2, operation, decimals, action and expected, all in a single statement. Very pretty.

Next, I begin to build up the data to post to the Web application under test:

```
...
expected = 'value=' + expected.Trim() + ' id="TextBox3"'
data = 'TextBox1=' + value1 + ' & TextBox2=' + value2 + '&0peration=' +
    operation + '&DropDownList1=' + decimals + '&Button1=' + action

print 'Test case: ' + caseid
print 'Input   : ' + data
print 'Expected: ' + expected
...
```

I slightly modify the expected variable to resemble something like: value="5.7900" id="TextBox3"

So when I search the HTTP response, I'll be more specific than simply searching for "5.7900." I associate my test case input values to their respective controls as name-value pairs connected by the "&" character. The first two name-value pairs of the post string simply set TextBox1 and TextBox2 to value1 and value2 from the test case data. The third name-value pair (for example, Operation=RadioButton1) is how I simulate a user selecting a RadioButton control, in this case, the control that corresponds to addition. You might have guessed incorrectly (as I originally did) that the way to set the radio button would be to use a syntax like RadioButton1=checked. But RadioButton1 is a value of the Operation control, not a control itself. The fifth name-value pair, Button1=clicked, is somewhat misleading. I need to supply a value for Button1 to indicate that the control has been clicked, but any value will work, so I could've used Button1=foo (or even just Button1=) but Button1=clicked is more descriptive, in my opinion. I echo the values I've parsed from the test case data to the command shell, making use of the "+" string concatenation operator.

Next, I deal with the ViewState and EventValidation values:

```
...
vs = getVS(url)
ev = getEV(url)
vs = HttpUtility.UrlEncode(vs)
ev = HttpUtility.UrlEncode(ev)
data = data + "&__VIEWSTATE=" + vs + "&__EVENTVALIDATION=" + ev
...
```

I call the getVS and getEV helper functions defined earlier. The ViewState and EventValidation values are Base64-encoded strings. Base64 encoding uses 64 characters: uppercase A-Z, lowercase a-z,

digits 0-9, the “+” character and the “/” character. The “=” character is used in Base64 for padding. Because some of the characters used in Base64 aren’t permitted in an HTTP request stream, I use the `HttpUtility.UrlEncode` method from the `System.Web` namespace to convert troublesome characters into a three-character sequence beginning with the “%” character.

For example, a raw “>” character is encoded as `%3D` and a blank space is encoded as `%20`. When a Web server receives an HTTP request containing any of these special three-character sequences, the server decodes the sequences back to raw input. After encoding, I append the `ViewState` and `EventValidation` values onto the `Post` data.

Next, I process the `Post` data to prepare it for an HTTP request:

```
...
buffer = Encoding.ASCII.GetBytes(data)
req = HttpRequest.Create(url)
req.Method = 'POST'
req.ContentType = 'application/x-www-form-urlencoded'
req.ContentLength = buffer.Length
...
```

I use the `GetBytes` method of the `System.Text` namespace to convert my `Post` data into a byte array named `buffer`. Then I create a new `HttpRequest` object using an explicit `Create` method. I supply values to the `Method`, `ContentType` and `ContentLength` properties of the `HttpRequest` object. You can think of the value of the `ContentType` as a magic string that’s necessary to `Post` an HTTP Web request.

Next, I send the HTTP request:

```
...
reqst = req.GetRequestStream()
reqst.Write(buffer, 0, buffer.Length)
reqst.Flush()
reqst.Close()
...
```

The programming pattern for sending a request may seem a bit odd if you’re new to the technique. Rather than use an explicit `Send` method of some sort, you create a `Stream` object and then use a `Write` method.

The `Write` method requires a byte array, the index in the array to begin writing and the number of bytes to write. By using 0 and `buffer.Length`, I write all bytes in the buffer. The `Write` method doesn’t actually send the `Post` to the Web server; you must force a send by calling the `Flush` method. The `Close` method actually calls the `Flush` method, so my call to `Flush` isn’t required in this situation, but I include the call for clarity.

After sending the HTTP request, I fetch the associated response:

```
...
res = req.GetResponse()
resst = res.GetResponseStream()
sr = StreamReader(resst)
html = sr.ReadToEnd()
sr.Close()
resst.Close()
...
```

The `GetResponse` method returns the `HttpWebResponse` object associated with an `HttpRequest` object. The response object can be used to create a `Stream`, and then I associate the `Stream` to a `StreamReader` object and use the `ReadToEnd` method to fetch the entire response stream as a single string. Although the underlying .NET Framework cleanup mechanisms would eventually close the `StreamReader` and `Stream` objects for you, I prefer to explicitly close them.

Next, I examine the HTTP response for the test case expected value:

```
...
if html.IndexOf(expected) >= 0:
    print 'Pass'
    numPass = numPass + 1
else:
    print '**FAIL**'
    numFail = numFail + 1
...
```

I use the `IndexOf` method to search the HTTP response. Because `IndexOf` returns the location of the beginning of the search string within the reference string, a return value `>= 0` means the search string is in the reference string. Note that unlike many languages, Python doesn’t have increment or decrement operators such as `++numPass`.

Next, I finish my script:

```
...
print '=====\\n'
# end main processing loop
fin.close()
print '\\nNumber pass = ' + str(numPass)
print 'Number fail = ' + str(numFail)
print '\\nEnd test run\\n'
except:
    print 'Fatal: ', sys.exc_info()[0]
# end script
```

I place a comment at the end of the “for” loop that iterates through each line of the test case data file as an aid to making sure my indentation is correct. Once outside the loop, I can close the test case data file and print the pass and fail counters. Notice that because `numPass` and `numFail` were inferred to be type `int`, I must cast them to type “string” using the Python `str` function so I can concatenate them. My harness finishes by handling any exceptions thrown in the `try` block simply by printing the generic exception message stored in the built-in `sys.exec_info` array.

Quick Automation with a Short Lifetime

The example I presented here should give you enough information to write IronPython test automation for your own Web applications. There are several alternatives to using IronPython. In my opinion, IronPython is best suited for lightweight test automation where you want to create the automation quickly, and the automation has a short (a few days or weeks) intended lifetime. My technique does have some limitations—in particular it can’t easily deal with Web applications that generate popup dialog boxes.

Compared to other scripting languages, one of the advantages of using IronPython for lightweight test automation is that you can use the interactive console to issue commands to help write your script. Additionally, there are several nice editors available, and you can even find IronPython SDKs that let you integrate IronPython into Visual Studio. Personally, when I’m going to write Web application test automation, if I want to develop a relatively short harness, I consider using IronPython and Notepad. If I’m going to write a harness that has more than three pages of code, however, I generally use C# and Visual Studio. ■

DR. JAMES MCCAFFREY works for Volt Information Sciences Inc. where he manages technical training for software engineers working at the Microsoft Redmond, Wash., campus. He has worked on several Microsoft products, including Internet Explorer and MSN Search. McCaffrey is the author of “*.NET Test Automation Recipes: A Problem-Solution Approach*” (Apress, 2006). He can be reached at jammc@microsoft.com.

THANKS to the following technical experts for reviewing this article:
Dave Fugate and Paul Newson



The MSF-Agile+SDL Process Template for TFS 2010

Anyone who is a regular reader of this magazine will be familiar with Microsoft Team Foundation Server (TFS) and the productivity benefits development teams can get from using it. (If you're not familiar with TFS, check out the article by Chris Menegay from our Visual Studio 2005 Guided Tour at msdn.microsoft.com/magazine/cc163686. While it covers an older version, it does give a useful overview of the kinds of features you can leverage in TFS.)

If you use TFS yourself, you're probably also familiar with the Microsoft Solutions Framework (MSF) for Agile Software Development process template—better known as MSF-Agile—that ships with TFS. The topic of this month's column is the new MSF-Agile plus Security Development Lifecycle (SDL) process template. MSF-Agile+SDL builds on the MSF-Agile template and adds SDL security and privacy features to the development process.

You can download the MSF-Agile+SDL template for Visual Studio Team System 2008 or 2010 from microsoft.com/sdl. Before you download it, however, you'll probably want to know what's been built into it. So let's get started.

SDL Tasks

The core of the SDL is its security requirements and recommendations—activities that dev teams must perform throughout the development lifecycle in order to ensure better security and privacy in the final product. These requirements include policy activities such as creating a security incident response plan, as well as technical activities such as threat modeling and performing static vulnerability analysis. All of these activities are represented in the MSF-Agile+SDL template as SDL task work items.

Probably the biggest difference between SDL tasks and the standard work items that represent functional tasks is that project team members are not meant to directly create SDL tasks themselves. Some SDL tasks are automatically created when the Team Project is first created. These are relatively straightforward, one-time-only security tasks, such as identifying the member of your team who will serve as the primary security contact. Other SDL tasks are automatically created by the process template in response to user actions. (More specifically, they're automatically created by the SDL-Agile controller Web service that gets deployed to the TFS application tier.)

Whenever a user adds a new iteration to the project, the template adds new SDL tasks to the project that represent the security tasks to be performed during that iteration. A good example of a per-iteration SDL task is threat modeling: The team must assess

the changes it makes over the course of the iteration to identify potential new threats and mitigations.

Finally, whenever a user checks a new Visual Studio project or Web site into the Team Source Control repository, the template adds SDL tasks to reflect the security work that must be done specifically for that project. For example, whenever a new C or C++ project is added, SDL tasks are created to ensure the use of buffer overflow defense compiler and linker settings, such as the `/dynamicbase` flag for address space layout randomization and the `/gs` flag for buffer security check.

The template is sophisticated enough to recognize the difference between native C/C++ projects and Microsoft .NET Framework projects, and it won't add invalid requirements. The `/dynamicbase` and `/gs` flags are meaningless to C# code and those SDL tasks won't be created for C# projects. Instead, C# projects get .NET-specific security tasks such as reviewing any use of `AllowPartiallyTrustedCallersAttribute`. Likewise, the template can also distinguish client/server and stand-alone desktop applications from Web sites and Web services, and will add only the appropriate set of SDL tasks accordingly.

SDL Task Workflow and Exceptions

The state and reason workflow transitions for SDL tasks are also different from those of functional tasks. A task can be marked as closed for several different reasons: completed, cut from the project, deferred to a later iteration, or even obsolete and no longer relevant to the project. Of these reasons, only completed is applicable to SDL tasks.

Teams that follow the SDL can't simply cut security and privacy requirements from their projects. Functional requirements can be horse-traded in and out of projects for technical or business reasons, but security requirements must be held to a higher standard. It's not impossible to ever skip an SDL task, but a higher level of process must be followed in order to make this happen.

If, for whatever reason, a team can't complete a required SDL task, it must petition its security advisor for an exception to the task. The team or its management chooses the team's security advisor at the project's start. This individual should have experience in application security and privacy, and ideally should not be working directly on the project—he should not be one of the project's developers, program managers or testers.

At Microsoft, there's a centralized group of security advisors who work in the Trustworthy Computing Security division. These security advisors then work with the individual product teams directly. If your organization has the resources to create a dedicated pool of

Still Struggling with Visual Basic® 6.0?

ArtinSoft's Visual Basic Upgrade Companion (VBUC) allows you to take advantage of the technology renewal cycle to Windows 7 and 64-bit computing, ensuring your Visual Basic 6 applications will not become an issue in the future.

- Convert your applications to .NET using ArtinSoft's next generation migration technology
- Save about 80% compared to a manual rewrite
- Ensure compatibility with the latest platforms
- Obtain maintainable, native code

From Fortune 100 companies to independent developers worldwide-thousands of applications successfully migrated to VB.NET and C#®

ArtinSoft is the only official MSDN partner for VB6 to .NET migrations



Visit www.artinsoft.com/msdn and learn how to

OBTAIN A FREE VBUC LICENSE



The world's leading software migration company



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

security advisors, great. If not, it's best to select the individual with the strongest background in security.

It's up to the team's security advisor to approve or reject any exception request for an SDL task. The team creates an exception request by setting the SDL task state to Exception Requested and filling out the Justification, Resolution Plan, and Resolution Timeframe fields. Each SDL task also has a read-only Exception Rating field that represents the inherent subjective security risk of not completing the requirement, which ranges from 4 (least risk) to 1 (critical; most risk). The security advisor weighs the team's rationale against the exception rating and either closes the SDL task with a Reason of Approved, or reactivates the SDL task with a Reason of Denied.

However, even if the request is approved, most exceptions don't last forever. This is where the Resolution Timeframe field comes into play. Teams generally request exceptions for a set number of iterations—usually just one iteration, but sometimes as many as three. Once the specified number of iterations has elapsed, the process template will expire the exception and reactivate the SDL task.

Security Bugs

After ensuring that security and privacy requirements are met, the next most important function of the SDL is to ensure that products don't ship with known security bugs. Tracking security bugs separately from functional bugs is critical to ensuring the security health of your product.

Unlike with SDL tasks, the MSF-Agile+SDL template doesn't add a second SDL Bug work item type to distinguish security bugs from functional bugs. Instead, it adds the fields Security Cause and Security Effect to the existing Bug work item type. Whenever a team member files a new bug, if the bug is a strict functional bug with no security implications, the finder simply leaves these fields at their default values of Not a Security Bug. However, if the bug does represent a potential security vulnerability, the finder sets the Security Cause to one of the following values:

- Arithmetic error
- Buffer overflow/underflow
- Cross-Site Scripting
- Cryptographic weakness
- Directory traversal
- Incorrect/no error messages
- Incorrect/no pathname canonicalization
- Ineffective secret hiding
- Race condition
- SQL/script injection
- Unlimited resource consumption (denial of service)
- Weak authentication
- Weak authorization/inappropriate permission or ACL
- Other

The finder also sets the Security Effect to one of the STRIDE values:

- Spoofing
- Tampering
- Repudiation
- Information Disclosure
- Denial of Service
- Elevation of Privilege

Finally, the finder can also choose to set the Scope value for the bug. In a nutshell, Scope defines some additional subjective information about the bug that is then used to determine severity. The allowed values for Scope vary based on the selected Security Effect. For example, if you choose Elevation of Privilege for the Security Effect, the possible choices for Scope include:

- (Client) Remote user has the ability to execute arbitrary code or obtain more privilege than intended.
- (Client) Remote user has the ability to execute arbitrary code with extensive user action.
- (Client) Local low-privilege user can elevate himself to another user, administrator or local system.
- (Server) Remote anonymous user has the ability to execute arbitrary code or obtain more privilege than intended.
- (Server) Remote authenticated user has the ability to execute arbitrary code or obtain more privilege than intended.
- (Server) Local authenticated user has the ability to execute arbitrary code or obtain more privilege than intended.

Tracking security bugs separately from functional bugs is critical.

You can see that the axes of severity for Elevation of Privilege vulnerabilities—that is, what characteristics make one Elevation of Privilege worse than another—deal with conditions such as the site of the attack (the client or the server) and the authentication level of the attacker (anonymous or authenticated). However, if you choose a different Security Effect, such as Denial of Service, the Scope choices change to reflect the axes of severity for that particular effect:

- (Client) Denial of service that requires reinstallation of system and/or components
- (Client) Denial of service that requires reboot or causes blue screen/bug check
- (Client) Denial of service that requires restart of application
- (Server) Denial of service by anonymous user with small amount of data
- (Server) Denial of service by anonymous user without amplification in default/common install
- (Server) Denial of service by authenticated user that requires system reboot or reinstallation
- (Server) Denial of service by authenticated user in default/common install

Once the values for Security Cause, Security Effect, and Scope have all been entered, the template uses this data to calculate a minimum severity for the bug. The user can choose to set the actual bug severity higher than the minimum bar—for example, to set the bug as a “1–Critical” rather than a “2–High”—but never the other way around. This may seem overly strict, but it avoids the temptation to downgrade bug severity in order to meet ship dates or sprint deadlines.

If you'd like to learn more about the rationale for setting up a more objective bug bar methodology for triaging bugs, read the March 2010 Security Briefs column, “Add a Security Bug Bar to Microsoft

LAS VEGAS • OCTOBER 18 - 20, 2010

**SAVE \$200
WHEN YOU
REGISTER BY
SEPTEMBER 15!**

Three Days of Design & Development Nirvana

We get it—web design is your world. That's why **Web Design World** is bringing you the top minds in web design this October. Oh, and did we mention it's in Vegas?

NEW: Exclusive 1-on-1 Consulting Time!

We've negotiated **20 minutes of individual face time with an expert** who will review your website, answer questions and offer helpful tips. all you have to do is check the box when you register, and pick the area you'd like to focus on:

- **Mobility**
- **Design**
- **eCommerce**
- **Development**
- **Usability**
- **Accessibility**

The best part? **We're not charging you one extra dime!** It's all part of your registration. There is one tiny catch: **seating is limited, so it's first come, first served!** Don't miss out on your chance for individual, one-on-one expert consulting - register early to save your spot!

Register By September 15th to SAVE \$200!

Use Priority Code **NQW7**

WebDesignWorld.com

NOVEMBER 14-17, 2010

ORLANDO, FL | HILTON WALT DISNEY WORLD RESORT

1105 MEDIA

Visual Studio® **LIVE!**

EXPERT SOLUTIONS FOR .NET DEVELOPERS

FOUR DAYS THAT WILL ROCK YOUR CODE.

Are you ready to take your code to the next level? Expand your skillset and maximize the development capabilities of Visual Studio during the four action-packed days of Visual Studio Live! Orlando! With pre-conference workshops, 60+ sessions by expert instructors, and keynotes by industry heavyweights, you'll walk away with knowledge and skills you can put to use today.



HARD-HITTING, REAL-WORLD TRAINING ON:

- **PROGRAMING WITH WCF**
- **ARCHITECTING FOR AZURE**
- **WPF & SILVERLIGHT**
- **ASP.NET 4**
- **JQUERY FOR ASP.NET**
- **WHAT'S NEW IN VISUAL STUDIO 2010**
- **SHAREPOINT 2010 FOR ASP.NET DEVELOPERS**

DETAILS AND REGISTRATION AT

© VSLIVE.COM/ORLANDO

USE PRIORITY CODE NQZF9

Supported by:

Microsoft

msdn

Microsoft Visual Studio

**Visual Studio
MAGAZINE**

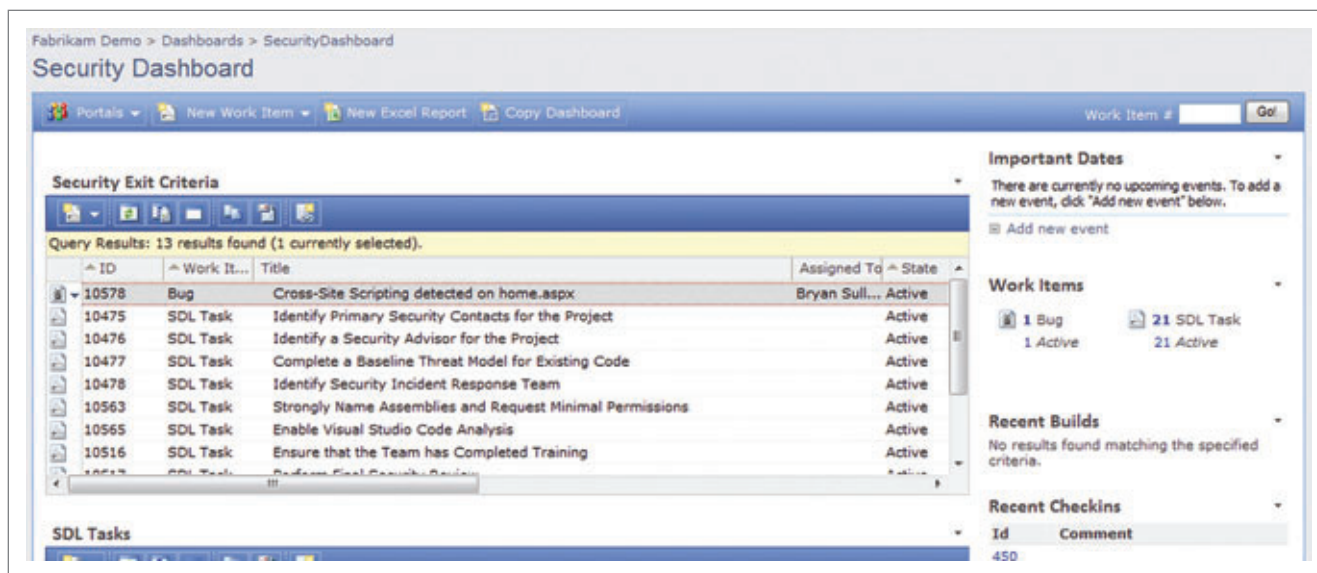


Figure 1 MSF-Agile+SDL Security Dashboard

Team Foundation Server 2010” (msdn.microsoft.com/magazine/ee336031). The process for adding a bug bar to TFS that I detailed in that article has already been built into the MSF-Agile+SDL template.

Finally, there’s one more optional field in the bug work item. You can use the Origin field to specify the name of the automated security tool that originally found the bug (if any) or you can leave the field at its default value of “User” if a user found the bug through manual code review or testing.

Over time, you’ll collect enough data to determine which of your testing tools are providing the biggest bang for the buck. To make this determination easier, the MSF-Agile+SDL template includes an Excel report called Bugs by Origin that displays a bar chart of vulnerabilities broken out by the Origin field.

You can customize this report to filter the data based on Severity, Security Cause or Security Effect. If you wanted to see which tools work best at finding cross-site scripting vulnerabilities or which tools have found the most critical severity Elevation of Privilege bugs, it’s easy to do so.

Bug Workflow

Just as you can’t defer SDL tasks, you can’t defer any bug with security implications (that is, any bug with its Security Effect set to a value other than Not a Security Bug). The team must request an exception in order to delay fixing any security bug with Severity of “3 – Moderate” or higher.

The process for this is identical to the exception request process for SDL Tasks: a team member sets the status to Exception Requested and enters details for the Justification, Exception Resolution and Exception Timeframe fields. The team’s security advisor then reviews the exception request and either approves it (setting the State to Closed with a Reason of Approved) or denies it (setting the State to Active with a Reason of Denied).

Security Queries and the Security Dashboard

In addition, the MSF-Agile+SDL template also includes several new team queries in order to simplify following the process. These new queries appear under the Security Queries folder in Team Explorer and include:

Figure 2 MSF-Agile+SDL Check-in Policies

SDL Check-in Policy	Description
SDL Banned APIs	Ensures that the compiler option to treat warning C4996 (use of a deprecated function) is treated as an error. Because most of the runtime library functions that can potentially lead to buffer overruns (for example, strcpy, strncpy and gets) have been deprecated in favor of more secure alternatives (strcpy_s, strncpy_s and gets_s, respectively), use of this check-in policy can significantly improve the application’s resistance to buffer overrun attacks.
SDL Buffer Security Check	Ensures that the compiler option Enable Buffer Security Check (/GS) is enabled. This option reorganizes the stack of the compiled program to include a security cookie or canary value that greatly increases the difficulty for an attacker to write a reliable exploit for any stack overflow vulnerability.
SDL DEP and ASLR	Ensures that the linker options Data Execution Prevention (/NXCOMPAT) and Randomized Base Address (/DYNAMICBASE) are enabled. These options randomize the address at which the application is loaded into memory and help to prevent code from executing in memory intended to be allocated as data. Especially when used in combination, these two options are strong defense-in-depth measures against buffer overrun attacks.
SDL Safe Exception Handlers	Ensures that the linker option /SAFESEH is enabled. This option helps prevent attackers from defining malicious exception handlers, which could lead to a compromise of the system. /SAFESEH creates a table of legitimate exception handlers at link time, and will not allow other exception handlers to run.
SDL Uninitialized Variables	Ensures that the compiler warning level is set at level 4 (/W4), the strictest level. Use of this option will flag code where variables may have been used without being initialized, which can lead to potential exploits.

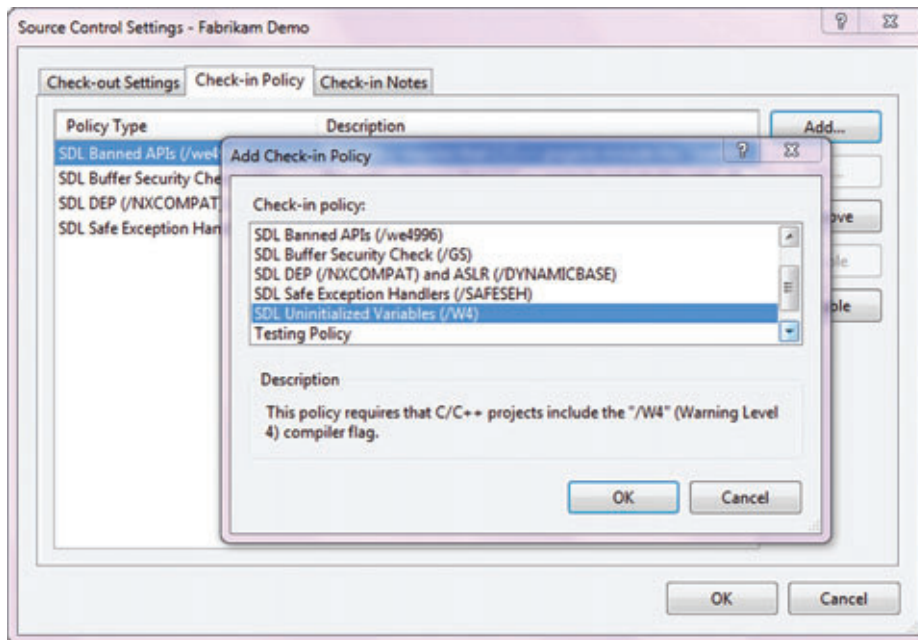


Figure 3 Adding Check-in Policies

- Active Security Bugs
- My Security Bugs
- Resolved Security Bugs
- Open SDL Tasks
- My SDL Tasks
- Open Exceptions (includes both tasks and bugs, and is especially useful for security advisors)
- Approved Exceptions
- Security Exit Criteria

Most of these queries are self-explanatory, but the Security Exit Criteria query needs a little more explaining. In order to meet their SDL commitment for a given iteration, the team must have completed all of the following activities:

- All every-sprint, recurring SDL task requirements for that iteration must be complete or have had an exception approved by the team's security advisor
- There must be no expired one-time or bucket SDL task requirements
- All bugs with security implications with Severity of "3 – Moderate" or higher must be closed or have had an exception approved by the team's security advisor

The terms every-sprint, one-time and bucket in this context refer to the SDL-Agile concept of organizing requirements based on the frequency with which they must be completed. Every-sprint requirements are recurring requirements and must be completed in every iteration. One-time requirements are non-recurring and only need to be completed once. Bucket requirements are recurring requirements, but only need to be completed once every six months.

A detailed discussion of this classification system is beyond the scope of this article; but if you'd like to understand more about this system, please read the *MSDN Magazine* article "Agile SDL: Streamline Security Practices for Agile Development" from the November 2008 issue (msdn.microsoft.com/magazine/dd153756).

The intent of the Security Exit Criteria query is to provide team members with an easy way to check how much more work they have left in order to complete their SDL commitment. If you configure a SharePoint site for your MSF-Agile+SDL team project when you create it (normally this is done for you automatically), you'll also see the Security Exit Criteria query results on the team project's Security Dashboard.

The new Security Dashboard is available only for MSF-Agile+SDL projects (see **Figure 1**). By default, it includes the Security Exit Criteria, Open SDL Tasks, Open Exceptions, and Security Bugs queries, but these can be customized if you like. The Security Dashboard is also set as the default project portal page for all MSF-Agile+SDL projects, but if you'd like to change to a different default dashboard, simply open

the Dashboards document library, select the dashboard you want to use, and choose the "Set as Default Page" option.

Check-in Policies

The final feature of the MSF-Agile+SDL process template is the set of SDL check-in policies. These policies help prevent developers from checking in code that violates certain SDL requirements and could therefore lead to security vulnerabilities. The SDL check-in policies available are shown in **Figure 2**.

It's simple to enable any or all of the SDL check-in policies. From Team Explorer, right-click a Team Project and select the Source Control option from the context menu. Choose the Check-in Policy tab and add the SDL policies you want to enforce (see **Figure 3**). It's important to note that check-in policy enforcement is performed on the client machine, not on the TFS server, so you'll need to install the SDL check-in policies on each developer's machine.

Wrapping Up

For any secure development methodology to be effective, it has to be easy to automate and easy to manage. The MSF-Agile+SDL process template helps significantly with both of these requirements. If you're already using the MSF-Agile process template that ships with TFS, you already know how to use the MSF-Agile+SDL template—it's a strict superset of the MSF-Agile template you're already familiar with. Download it from microsoft.com/sdl and start creating more secure and more privacy-aware products today. ■

BRYAN SULLIVAN is a security program manager for the Microsoft Security Development Lifecycle team, where he specializes in Web application and .NET security issues. He's the author of "Ajax Security" (Addison-Wesley, 2007).

THANKS to the following technical expert for reviewing this article:
Michael Howard

Know it all.

We've got all the answers.

- Thousands of code samples
- Over 28,000 articles
- Practical answers in active forums
- MFC/Microsoft® Visual C++®, Visual C#®, Visual Basic® .NET, ASP.NET, SQL Server®, Windows® Phone, and more!
- QuickAnswers
- Over 7.2 million members
- Start your FREE membership TODAY!



THE CODE PROJECT™
WWW.CODEPROJECT.COM



Touch and Response

Programming is an engineering discipline rather than a science or a branch of mathematics, so rarely does there exist a single correct solution to a problem. Varieties and variations are the norm, and often it's illuminating to explore these alternatives rather than focus on one particular approach.

In my article "Multi-Touch Manipulation Events in WPF" in the August issue of *MSDN Magazine* (msdn.microsoft.com/magazine/ff898416), I began exploring the exciting multi-touch support introduced into version 4 of the Windows Presentation Foundation (WPF). The Manipulation events serve primarily to consolidate multi-touch input into useful geometric transforms, and to assist in implementing inertia.

In that article, I showed two related approaches to handling Manipulation events on a collection of Image elements. In both cases, the actual events were processed by the Window class. One program defined handlers for the Manipulation events of the manipulated elements. The other approach showed how to override the `OnManipulation` methods to get the same events routed through the visual tree.

The Custom Class Approach

A third approach also makes sense: A custom class can be defined for the manipulated elements that overrides its own `OnManipulation` methods rather than leaving this job to a container element. The advantage of this approach is that you can make the custom class a little more attractive by decorating it with a Border or other element; these decorations can also be used to provide visual feedback when the user touches a manipulable element.

When veteran WPF programmers determine they need to make visual changes to a control based on events, they probably think of `EventTrigger`, but WPF programmers need to start transitioning to the Visual State Manager. Even when deriving from `UserControl` (the strategy I'll be using), it's fairly easy to implement.

An application using the Manipulation events should probably base visual feedback on those same events rather than the low-level `TouchDown` and `TouchUp` events. When using the Manipulation events, you'll want to begin the visual feedback with either the `ManipulationStarting` or `ManipulationStarted` event. (It really doesn't make a difference which you choose for this job.)

However, when experimenting with this feedback, one of the first things you'll discover is that the `ManipulationStarting` and `ManipulationStarted` events are not fired when an element is first touched, but only when it starts moving. This behavior is a holdover

from the stylus interface, and you'll want to change it by setting the following attached property on the manipulated element:

```
Stylus.IsPressAndHoldEnabled="False"
```

Now, the `ManipulationStarting` and `ManipulationStarted` events are fired when an element is first touched. You'll want to turn off the visual feedback with either the `ManipulationInertiaStarting` or `ManipulationCompleted` event, depending on whether you want the feedback to end when the user's finger lifts from the screen or after the element has stopped moving due to inertia. If you're not using inertia (as I won't be in this article), it doesn't matter which event you use.

A custom class can be defined for the manipulated elements that overrides its own `OnManipulation` methods rather than leaving this job to a container element.

The downloadable code for this article is in a single Visual Studio solution named `TouchAndResponseDemos` with two projects. The first project is named `FeedbackAndSmoothZ`, which includes a custom `UserControl` derivative named `ManipulablePictureFrame` that implements the manipulation logic.

`ManipulablePictureFrame` defines a single property of type `Child` and uses its static constructor to redefine defaults for three properties: `HorizontalAlignment`, `VerticalAlignment` and the all-important `IsManipulationEnabled`. The instance constructor calls `InitializeComponent` (as usual), but then sets the control's `RenderTransform` to a `MatrixTransform` if that's not the case.

During the `OnManipulationStarting` event, the `ManipulablePictureFrame` class calls:

```
VisualStateManager.GoToElementState(this, "Touched", false);
```

and during the `OnManipulationCompleted` event, it calls:

```
VisualStateManager.GoToElementState(this, "Untouched", false);
```

Code download available at code.msdn.microsoft.com/mag201009UIFrontiers.

Figure 1 The ManipulablePictureFrame.xaml File

```
<UserControl x:Class="FeedbackAndSmoothZ.ManipulablePictureFrame"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Stylus.IsPressAndHoldEnabled="False"
    Name="this">

    <VisualStateManager.VisualStateGroups>
        <VisualStateGroup x:Name="TouchStates">
            <VisualState x:Name="Touched">
                <Storyboard>
                    <DoubleAnimation Storyboard.TargetName="maskBorder"
                        Storyboard.TargetProperty="Opacity"
                        To="0.33" Duration="0:0:0.25" />

                    <DoubleAnimation Storyboard.TargetName="dropShadow"
                        Storyboard.TargetProperty="ShadowDepth"
                        To="20" Duration="0:0:0.25" />
                </Storyboard>
            </VisualState>
            <VisualState x:Name="Untouched">
                <Storyboard>
                    <DoubleAnimation Storyboard.TargetName="maskBorder"
                        Storyboard.TargetProperty="Opacity"
                        To="0" Duration="0:0:0.1" />

                    <DoubleAnimation Storyboard.TargetName="dropShadow"
                        Storyboard.TargetProperty="ShadowDepth"
                        To="5" Duration="0:0:0.1" />
                </Storyboard>
            </VisualState>
        </VisualStateGroup>
    </VisualStateManager.VisualStateGroups>

    <Grid>
        <Grid.Effect>
            <DropShadowEffect x:Name="dropShadow" />
        </Grid.Effect>

        <!-- Holds the photo (or other element) -->
        <Border x:Name="border"
            Margin="24" />

        <!-- Provides visual feedback -->
        <Border x:Name="maskBorder"
            Margin="24"
            Background="White"
            Opacity="0" />

        <!-- Draws the frame -->
        <Rectangle Stroke="{Binding ElementName=this, Path=Foreground}"
            StrokeThickness="24"
            StrokeDashArray="0 0.9"
            StrokeDashCap="Round"
            RadiusX="24"
            RadiusY="24" />

        <Rectangle Stroke="{Binding ElementName=this, Path=Foreground}"
            StrokeThickness="8"
            Margin="16"
            RadiusX="24"
            RadiusY="24" />
    </Grid>
</UserControl>
```

This is my code file's sole contribution to implementing visual states. The code performing the actual manipulations will be familiar from the code in last month's column—with two significant changes:

- In the `OnManipulationStarting` method, the `ManipulationContainer` is set to the element's parent.
- The `OnManipulationDelta` method is just a little simpler because the element being manipulated is the `ManipulablePictureFrame` object itself.

Figure 1 shows the complete `ManipulablePictureFrame.xaml` file.

Pretty much any kind of simple highlighting can provide visual feedback during touch events.

The `Border` named "border" is used to host the child of the `ManipulablePictureFrame` class. This will probably be an `Image` element, but it doesn't have to be. The two `Rectangle` elements draw a type of "scalloped" frame around the `Border`, and the second `Border` is used for visual feedback.

While an element is being moved, the animations in `ManipulablePictureFrame.xaml` "lighten" the picture a bit—actually, it's more of a "washing out" effect—and increase the drop shadow, as shown in Figure 2.

Pretty much any kind of simple highlighting can provide visual feedback during touch events. However, if you're working with small elements that can be touched and manipulated, you'll want to make the elements larger when they're touched so they won't be

entirely hidden by the user's finger. (On the other hand, you don't want to make an element larger for visual feedback if you're also allowing the user to resize the element. It's very disconcerting to manipulate an element into a desired size and then have it shrink a little when you lift your fingers from the screen!)

You'll notice that as you make the images smaller and larger, the frame shrinks or expands accordingly. Is this correct behavior? Perhaps. Perhaps not. I'll show an alternative to this behavior toward the end of this article.

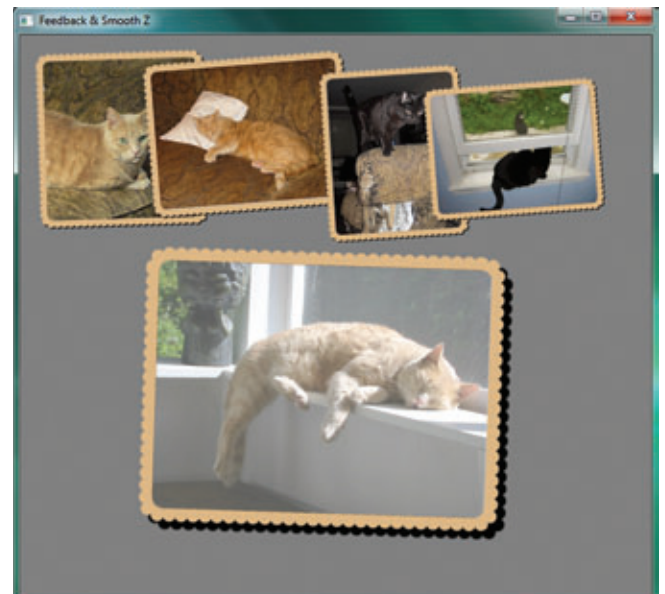


Figure 2 A Highlighted Element in the FeedbackAndSmoothZ Program

Smooth Z Transitions

In the programs I showed last month, touching a photo would cause it to jump to the foreground. This was just about the simplest approach I could think of and required setting new `Panel.ZIndex` attached properties for all the Image elements.

A brief refresher: Normally when children of a Panel overlap, they are arranged from background to foreground based on their position in the Children collection of the Panel. However, the Panel class defines an attached property named `ZIndex` that effectively supersedes the child index. (The name alludes to the Z-axis orthogonal to the conventional XY plane of the screen, which conceptually comes out of the screen.) Elements with a lower `ZIndex` value are in the background; higher `ZIndex` values put an element in the foreground. If two or more overlapping elements have the same `ZIndex` setting (which is the case by default), their child indices in the Children collection are used instead to determine which is on top of the other.

In the earlier programs, I used the following code to set new `Panel.ZIndex` values, where the variable `element` is the element being touched and `pnl` (of type `Panel`) is the parent of that element and its siblings:

```
for (int i = 0; i < pnl.Children.Count; i++)
    Panel.SetZIndex(pnl.Children[i],
        pnl.Children[i] == element ? pnl.Children.Count : i);
```

This code ensures that the touched element gets the highest `ZIndex` and appears in the foreground.

Unfortunately, the touched element jumps to the foreground in a sudden, rather unnatural, movement. Sometimes other elements switch places at the same time. (If you have four overlapping elements and touch the first, it gets a `ZIndex` of 4 and the others have `ZIndex`

values of 1, 2 and 3. Now if you touch the fourth, the first goes back to a `ZIndex` of 0 and will suddenly go behind all the others.)

My goal was to avoid the sudden snapping of elements to the foreground and background. I wanted a smoother effect that mimicked the process of slipping a photo from underneath a pile and then slipping it back on top. In my mind, I started thinking of these transitions as “smooth Z.” Nothing would jump to the foreground or background, but as you moved an element around, eventually it would find itself on top of all the others. (An alternative approach is implemented in the `ScatterView` control available for download from CodePlex at scatterview.codeplex.com/releases/view/24159. `ScatterView` is certainly preferable when dealing with large numbers of items.)

My goal was to avoid the sudden snapping of elements to the foreground and background.

In implementing this algorithm, I set a few criteria for myself. First, I didn't want to maintain state information from one move event to the next. In other words, I didn't want to analyze whether the manipulated element was intersecting another element previously but was no longer. Second, I didn't want to perform memory allocations during the `ManipulationDelta` events because there might be many of them. Third, to avoid too much complexity, I wanted to restrict changes of the relative `ZIndex` to only the manipulated element.

Figure 3 The Smooth Z Algorithm

```
// BumpUpZIndex with reusable SortedDictionary object
SortedDictionary<int, UIElement> childrenByZIndex = new
SortedDictionary<int, UIElement>();

void BumpUpZIndex(FrameworkElement touchedElement, UIElementCollection siblings)
{
    // Make sure everybody has a unique even ZIndex
    for (int childIndex = 0; childIndex < siblings.Count; childIndex++)
    {
        UIElement child = siblings[childIndex];
        int zIndex = Panel.GetZIndex(child);
        Panel.SetZIndex(child, 2 * (zIndex * siblings.Count + childIndex));
    }

    int zIndexNew = Panel.GetZIndex(touchedElement);
    int zIndexCantGoBeyond = Int32.MaxValue;

    // Don't want to jump ahead of any intersecting elements that are on top
    foreach (UIElement child in siblings)
    {
        if (child != touchedElement &&
            AreElementsIntersecting(touchedElement, (FrameworkElement)child))
        {
            int zIndexChild = Panel.GetZIndex(child);

            if (zIndexChild > Panel.GetZIndex(touchedElement))
                zIndexCantGoBeyond = Math.Min(zIndexCantGoBeyond, zIndexChild);
        }

        // But want to be in front of non-intersecting elements
        foreach (UIElement child in siblings)
        {
            if (child != touchedElement &&
                !AreElementsIntersecting(touchedElement, (FrameworkElement)child))
            {
                // This ZIndex is odd, hence unique
                int zIndexNextHigher = 1 + Panel.GetZIndex(child);

                if (zIndexNextHigher < zIndexCantGoBeyond)
                    zIndexNew = Math.Max(zIndexNew, zIndexNextHigher);
            }
        }

        // Now give all elements indices from 0 to (siblings.Count - 1)
        Panel.SetZIndex(touchedElement, zIndexNew);
        childrenByZIndex.Clear();
        int index = 0;

        foreach (UIElement child in siblings)
            childrenByZIndex.Add(Panel.GetZIndex(child), child);

        foreach (UIElement child in childrenByZIndex.Values)
            Panel.SetZIndex(child, index++);
    }

    // Test if elements are intersecting with reusable //
    RectangleGeometry objects
    RectangleGeometry rectGeo1 = new RectangleGeometry();
    RectangleGeometry rectGeo2 = new RectangleGeometry();

    bool AreElementsIntersecting(FrameworkElement element1, FrameworkElement
    element2)
    {
        rectGeo1.Rect = new
        Rect(new Size(element1.ActualWidth, element1.ActualHeight));
        rectGeo1.Transform = element1.RenderTransform;

        rectGeo2.Rect = new
        Rect(new Size(element2.ActualWidth, element2.ActualHeight));
        rectGeo2.Transform = element2.RenderTransform;

        return rectGeo1.FillContainsWithDetail(rectGeo2) != IntersectionDetail.
        Empty;
    }
}
```


Figure 4 Alternative Smooth Z Logic for Manipulation Without Transforms

```
bool AreElementsIntersecting(FrameworkElement element1, FrameworkElement element2)
{
    rectGeo1.Rect = new Rect(Canvas.GetLeft(element1), Canvas.GetTop(element1),
        element1.ActualWidth, element1.ActualHeight);

    rectGeo2.Rect = new Rect(Canvas.GetLeft(element2), Canvas.GetTop(element2),
        element2.ActualWidth, element2.ActualHeight);

    return rectGeo1.FillContainsWithDetail(rectGeo2) != IntersectionDetail.Empty;
}
```

The complete algorithm is shown in **Figure 3**. Crucial to the approach is determining whether two sibling elements visually intersect. There are several ways to go about this, but the code I used (in the `AreElementsIntersecting` method) seemed the simplest. It reuses two `RectangleGeometry` objects stored as fields.

The `BumpUpZIndex` method performs the bulk of the work. It begins by making sure all the siblings have unique `ZIndex` values, and that all these values are even numbers. The new `ZIndex` for the manipulated element can't be higher than any `ZIndex` value of any element that's intersecting and currently on top of the manipulated element. Taking this limit into account, the code attempts to assign a new `ZIndex` that's higher than the `ZIndex` values of all non-intersecting elements.

The code I've discussed so far will normally have the effect of progressively increasing `ZIndex` values without limit, eventually exceeding the maximum positive integer value and becoming negative. This situation is avoided using a `SortedDictionary`. All the siblings are put into the dictionary with their `ZIndex` values as keys. Then the elements can be given new `ZIndex` values based on their indices in the dictionary.

When resizing Image elements,
it's usually best to specify just one
dimension and let the element
choose its other dimension to
maintain the proper aspect ratio.

The Smooth Z algorithm has a quirk or two. If the manipulated element is intersecting element A but not element B, then it can't be slipped on top of B if B has a higher `ZIndex` than A. Also, there's been no special accommodation for manipulating two or more elements at the same time.

Manipulation Without Transforms

In all the examples I've shown so far, I've used information delivered with the `ManipulationDelta` event to alter the `RenderTransform` of the manipulated element. That's not the only option. In fact, if you don't need rotation, you can implement multi-touch manipulation without any transforms at all.

This "no transform" approach involves using a `Canvas` as a container for the manipulated elements. You can then move the

elements on the `Canvas` by setting the `Canvas.Left` and `Canvas.Top` attached properties. Changing the size of the elements requires manipulating the `Height` and `Width` properties, either with the same percentage `Scale` values used previously or with the absolute `Expansion` values.

One distinct advantage of this approach is that you can decorate the manipulated elements with a border that won't itself become larger and smaller as you change the size of the element.

This technique is demonstrated in the `NoTransformManipulation` project, which includes a `UserControl` derivative named `NoTransformPictureFrame` that implements the manipulation logic.

The picture frame in this new class isn't nearly as fancy as the one in `ManipulablePictureFrame`. The earlier picture frame used a dotted line for a scalloped effect. If you make such a frame larger to accommodate a larger child but without applying a transform, the line thickness will remain the same and the number of dots in the dotted line will increase! This looks very peculiar and is probably too distracting for a real-life program. The picture frame in the new file is just a simple `Border` with rounded corners.

In the `MainPage.xaml` file in the `NoTransformManipulation` project, five `NoTransformPictureFrame` objects are assembled on a `Canvas`, all containing `Image` elements and all with unique `Canvas.Left` and `Canvas.Top` attached properties. Also, I've given each `NoTransformPictureFrame` a `Width` of 200 but no `Height`. When resizing `Image` elements, it's usually best to specify just one dimension and let the element choose its other dimension to maintain the proper aspect ratio.

The `NoTransformPictureFrame.xaml.cs` file is similar in structure to the `ManipulablePictureFrame` code except that no transform code is required. The `OnManipulationDelta` override adjusts the `Canvas.Left` and `Canvas.Top` attached properties and uses the `Expansion` values to increase the `Width` property of the element. Just a little bit of trickiness is required when scaling is in effect, because the translation factors need to be adjusted to accommodate the center of scaling.

A change was also required in the `AreElementsIntersecting` method that plays a crucial role in the smooth Z transitions. The earlier method constructed two `RectangleGeometry` objects reflecting the untransformed dimensions of the two elements and then applied the two `RenderTransform` settings. The replacement method is shown in **Figure 4**. These `RectangleGeometry` objects are based solely on the actual size of the element offset by the `Canvas.Left` and `Canvas.Top` attached properties.

Remaining Issues

As I've been discussing the Manipulation events, I've been ignoring an important feature, and the elephant in the room has become larger and larger. That feature is inertia, which I'll tackle in the next issue. ■

CHARLES PETZOLD is a longtime contributing editor to MSDN Magazine. He's currently writing "Programming Windows Phone 7," which will be published as a free downloadable e-book in the fall of 2010. A preview edition is currently available through his Web site, charlespetzold.com.

THANKS to the following technical experts for reviewing this column:
Doug Kramer and Robert Levy



Weasel Words

My e-mail service went down last month for a full day. When it finally came back up, I received an apology from its administrator, saying: “users experienced e-mail connectivity issues.” Bullhockey, Mr. Administrator. I did not experience “an issue.” I experienced the lack of e-mail because your servers were down. I experienced the waste of my time, the delay of my projects and the loss of my income. I experienced anger at your enterprise, which promised reliability but didn’t deliver. And I experienced even greater anger at your attempt to downplay your malpractice by using that worst of all weasel words: “issue.”

Don’t get me started on the “I” word. I have no problem with its meaning of distribution, as in “the issue of food and blankets to flood victims.” Nor do I mind its meaning of offspring, as in “my issue is two daughters, on whom the sun rises and sets,” nor for designating a specific month’s magazine, as in “the September issue of *MSDN Magazine*,” which you are now reading. But I hereby fling scorn and disdain at weasels who use this term to mean “software malfunction,” hoping that the users whom that malfunction harms will somehow be less angry at them than if they had said, “Gosh, we know you had no e-mail because we screwed up, and we know how we hate it when that happens to us, so we’re really, really sorry and we’ll give you a free month of service for your trouble—maybe two months if you squawk really loudly.”

The apology’s author uses the “I” word five times in four paragraphs, including the memorable phrase, “Once the configuration issues were resolved and all the servers were online, we discovered that some users were still experiencing issues...” Please, somebody, put this guy out of his misery.

Weasel words aren’t harmless. They try to hide a problem that needs to be solved—Johnny has “a drinking issue.” No he doesn’t. Johnny’s a drunk.

As any recovering alcoholic will tell you, the very first word of the very first step to recovery is “Admit.” Johnny won’t get better until he stops hiding behind weasel words, until he can stand up in public and say: “My name is Johnny, and I am an alcoholic, but I don’t want to be a drunk anymore.” His loved ones hope he does that before he kills himself or someone else. Using the “I” word only postpones that day of realization.

Developers don’t talk in weasel words, and we don’t like hearing them. We’re engineers; solving problems is what we do. Before we can solve a problem, we need to recognize its existence and call it by its correct name. You can always tell a developer who’s starting to drink the manager’s Kool-Aid, bucking for a raise. He goes away on a training program retreat, comes back with a tie and a lobotomy scar, and starts referring to bugs as issues. And then, like any zombie, he tries to eat your brain so you’ll be a zombie too: “Bob, can I have your list of issues by Friday?”

Weasel words aren’t harmless.
They try to hide a problem that
needs to be solved—Johnny has
“a drinking issue.” No he doesn’t.
Johnny’s a drunk.

At Tech•Ed some years ago, I exhorted my listeners: “It’s not an issue, it’s a bug. Say the word. Say it loudly: Bug. B as in Bad. U as in Ugly. G as in Gol-dangit, I’ve got a bug.” I got a standing ovation.

If you want to issue supplies, read a magazine issue, or even take issue with my writing here, fine. But don’t use the “I” word to mean “software malfunction.” That tells your users that you don’t share their concerns, that you don’t really give a darn about them, that you think they’re stupid enough to believe your twaddle. It’s a blatant form of disrespect toward the people who pay your salary, who put bread into your children’s mouths and a roof over their heads. And I have a serious problem—not an issue—with that.

Request to readers: Do you have any favorite examples of weasel words? Send them to me via rollthunder.com, and I’ll use the best of them in a future column. ■

DAVID S. PLATT teaches *Programming .NET* at Harvard University Extension School and at companies all over the world. He is the author of 11 programming books, including “*Why Software Sucks*” (Addison-Wesley Professional, 2006) and “*Introducing Microsoft .NET*” (Microsoft Press, 2002). Microsoft named him a Software Legend in 2002. He wonders whether he should tape down two of his daughter’s fingers so she learns how to count in octal. You can contact him at rollthunder.com.



New Version

GrapeCity PowerTools



for Windows Forms
for ASP.NET

SPREAD 5

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

Once Again,
**The Best Grid is
a Spreadsheet.**



Spread is the Winner of this year's CodeProject Members Choice Award for Best Grid Controls

"As before, The Code Project Members Choice Awards reflect the diversity and depth of the tools available to professional developers around the world. This past year has brought even more user interface (UI) tools but also more tools to support all aspects of the development process. Congratulations ..."

Jeff Hadfield,
President of The Code Project (USA)

Benefits

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

**Are you using the Best
~~Grid~~ Spreadsheet?**

Act now and save up to \$300!

GCPowerTools.com/ActNow

1-800-645-5913 / 1-919-460-4551



WE ARE
GrapeCity
Excel  Report  Analyze

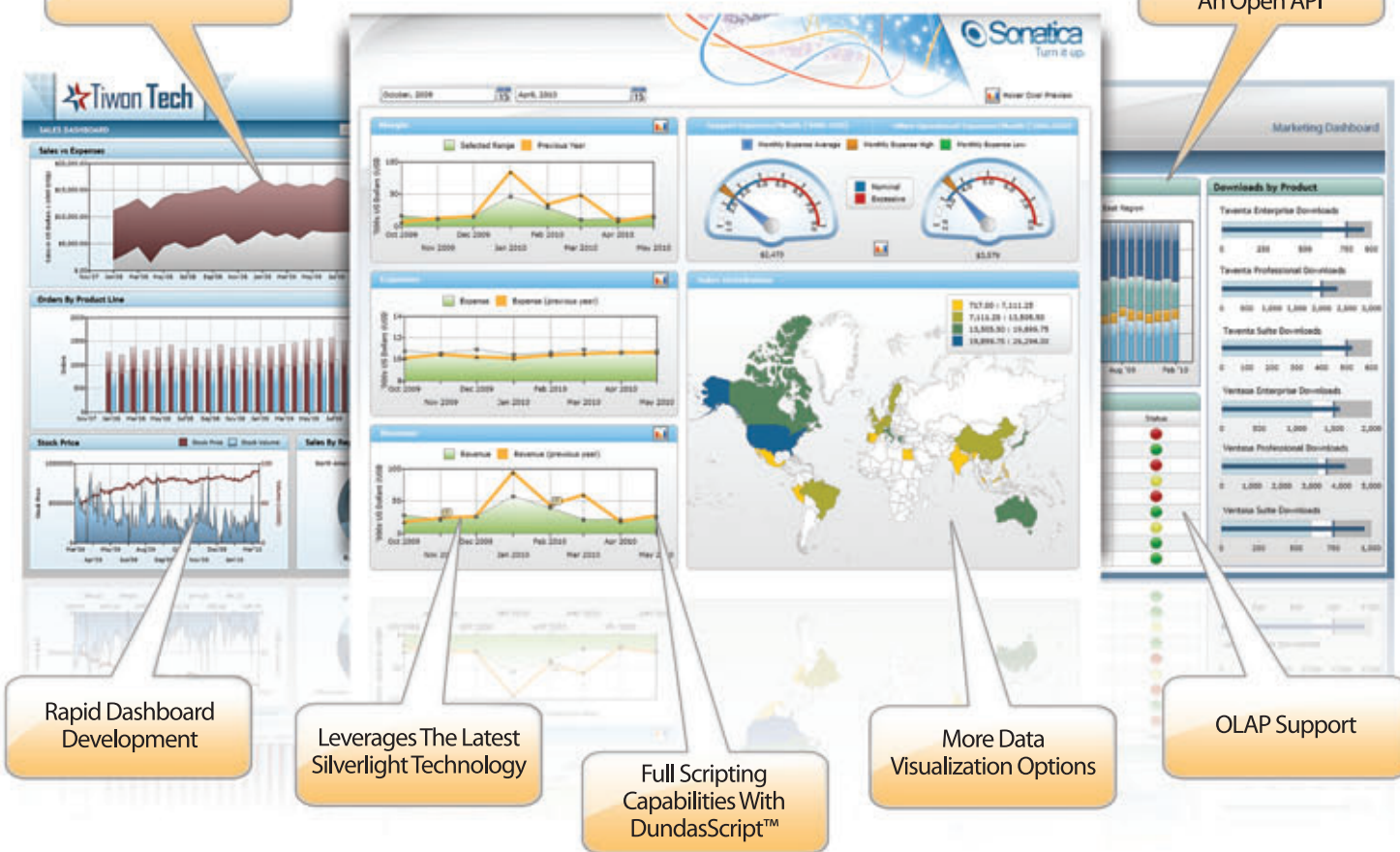


The Dashboard Platform for Developers like you

Build more **powerful** and **effective**
dashboards, **faster**.

Support For Numerous
Data Sources

Extensible And
Customizable With
An Open API



Dundas Dashboard is a ground-breaking, extensible solution that uses a revolutionary approach to dashboard creation, providing you with a unified view of key metrics and a new level of strategic insight and decision-making.



Powered by
Microsoft Silverlight



www.dundas.com/dashboard

(416) 467-5100 • (800) 463-1492

Silverlight is a trademark of Microsoft Corporation in the United States and/or other countries.