THE MICROSOFT JOURNAL FOR DEVELOPERS

DECEMBER 2011 VOL 26 NO 12

msdn magazine

COLUMNS

Microsoft

Write Once, Experience Many

check out infragistics.com/jquery

NetAdvantage® for jQuery

TREE

Simplify the look of hierarchical data, while offering the experience, design and functionality your users will love!

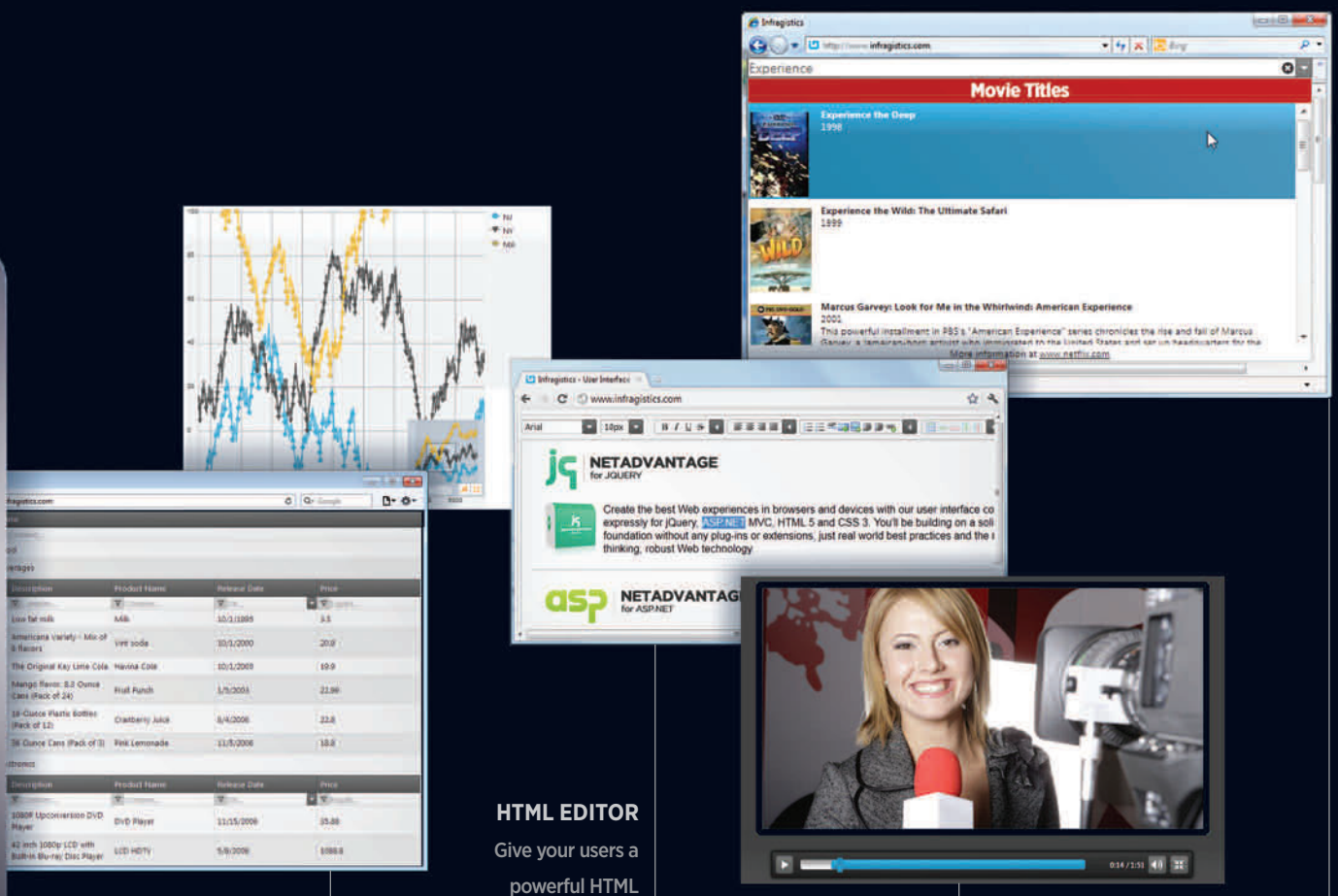
BUSINESS CHARTING

Combine interactive Outlook style grids with rich business charting to deliver a complete portable solution.

INFRAGISTICS

DESIGN / DEVELOP / EXPERIENCE

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



HIERARCHICAL GRID

An expandable data grid that presents multiple parent-child relationships is the backbone of your data application.

HTML EDITOR

Give your users a powerful HTML editing experience by incorporating the jQuery WYSIWYG editing tool.

VIDEO PLAYER

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

COMBO

The fully featured combo box control offers intuitive auto-suggest, auto-complete and auto-filtering built in.

msdn® magazine

DECEMBER 2011 VOLUME 26 NUMBER 12

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

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

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

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

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

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

Microsoft

BPA WORLDWIDE

Printed in the USA

techxtend.com
866-719-1528

TechXtend | programmer's paradise

NEW VERSION!

Embarcadero RAD Studio XE2

The ultimate application development suite for Windows, Mac, mobile and Web

by Embarcadero

Embarcadero® RAD Studio XE2 is the ultimate application development suite and the fastest way to build data-rich, visually engaging applications for Windows, Mac, mobile, .NET, PHP and the Web. RAD Studio includes Delphi®, C++Builder® and RadPHP™, enabling developers to deliver applications up to 5x faster across multiple desktop, mobile, Web, and database platforms including Delphi applications for 64-bit Windows.

techxtend.com/embarcadero

Professional Ed.
TechXtend #
CGI 15501A01

$1,383.99

Intel Visual Fortran Composer XE

by Intel

Intel® Visual Fortran Composer XE 2011 includes the latest generation of Intel Fortran compilers, Intel® Visual Fortran Compiler XE 12.0 for Windows*. Intel® Fortran Composer XE is available for Linux* and Mac OS* X. This package delivers advanced capabilities for development of application parallelism and winning performance for the full range of Intel® processor-based platforms and other compatible platforms. It includes the compiler's breadth of advanced optimization, multithreading, and processor support, as well as automatic processor dispatch, vectorization, and loop unrolling.

techxtend.com/intel

for Windows
Single User
TechXtend #
I23 86101E01

$659.99

VMware vSphere 5 Essentials Kit

VMware vSphere is the industry-leading virtualization platform for building cloud infrastructures that enables users to run business critical applications with confidence and respond faster to their business.

NEW VERSION 5!

vSphere accelerates the shift to cloud computing for existing datacenters, while also underpinning compatible public cloud offerings paving the way for the only hybrid cloud model. With over 250,000 customers worldwide and the support of over 2500 applications from more than 1400 ISV partners, VMware vSphere is the trusted platform for any application.

techxtend.com/vSphere

for 3 hosts
TechXtend #
V55 85201A01

$446.99

NEW VERSION 6!

ActiveReports 6

by GrapeCity PowerTools

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

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

techxtend.com/grapecity

Professional Ed.
TechXtend #
D03 04301A01

$1,310.99

Lenovo ThinkPad Tablet

by Lenovo

From the makers of the ThinkPad® comes an exciting Android 3.1 Tablet that offers a superior multimedia and entertainment experience. Unlike other tablets, the Lenovo® ThinkPad Tablet comes with additional features to help business professionals to do more on the go—and also allows IT managers support them easily, no matter where they go. This means, better management of multiple devices, more customization options, higher productivity, and more importantly, enhanced security. So, if you are looking for a tablet built for business, then it has to be the ThinkPad Tablet.

techxtend.com/lenovo

TechXtend #
ZHI KW3343

$589.00

TX Text Control 16.0

Word Processing Components

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

New Version Released!

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

Download a demo today.

techxtend.com/textcontrol

Professional Edition
TechXtend #
T79 12101A01

$1,109.99

Microsoft Visual Studio Professional 2010

by Microsoft

Microsoft Visual Studio 2010 Professional with MSDN Essentials Subscription is an integrated environment that simplifies creating, debugging and deploying applications. Unleash your creativity and bring your vision to life with powerful design surfaces and innovative collaboration methods for developers and designers. Work within a personalized environment, targeting a growing number of platforms, including Microsoft SharePoint and cloud applications and accelerate the coding process by using your existing skills.

techxtend.com/microsoft

Upgrade
TechXtend #
M47 40201B02

$479.99

HP ProCurve M110 IEEE

by Hewlett Packard

HP E-M110 Access Point is an entry-level, single IEEE 802.11 a/b/g radio device that can be configured to operate as an access point, a wireless distribution system (WDS) bridge, or a full-spectrum IEEE 802.11 wireless LAN (WLAN) monitor. The E-M110 Access Point supports up to two virtual service communities, each with independent VLAN and wireless security profiles. It is plenum-rated and powered by an IEEE 802.3af (PoE) source. The M110 access point is Wi-Fi Certified for WPA2, WPA, and WEP security and has hardware-assisted AES and RC4 encryption.

techxtend.com/hp

TechXtend #
ZHI CV0496

$166.99

VMware Workstation 8

Your On-Ramp to the Cloud

by VMware

NEW VERSION!

With more than 50 new features, it will dramatically change the way you work with virtual machines.

Winner of more than 50 industry awards, VMware Workstation is recognized for its broad operating system support, rich user experience, comprehensive feature set and high performance.

It is the perfect companion for technical professionals who want to save time, collaborate more effectively and do more with their PCs than they ever thought possible.

techxtend.com/vmware

Upgrade for
Linux & Windows
TechXtend #
V55 22401B01

$78.99

Bluebeam PDF Revu Standard

Simple. Reliable. Affordable.

by Bluebeam Software

Bluebeam PDF Revu was designed to make your life easier with simple PDF creation, markup, editing and access features. When using Bluebeam you will soon come to realize the ease of use and functionality provided with our software. Bluebeam PDF Revu integrates with MS Office programs for one button file creation and includes the Bluebeam PDF printer to create a PDF from just about anything!

techxtend.com/bluebeam

1-4 Seats
TechXtend #
BB4 02401A01

$139.22

HP Compaq 6000 Pro All-in-One PC

by Hewlett Packard

Capable. Integrated. Designed with enterprise class performance, manageability, and stability in an integrated form. Get a desktop computer, high-definition LCD monitor, webcam and more; the tools you need for your twenty-first century business. Features unique to the HP Compaq 6000 Pro, you get integrated collaboration tools, DisplayPort output to create a dual display setup, and an integrated 6-in-1 media card reader to easily transfer files, videos, and more.

techxtend.com/hp

TechXtend #
ZHI DJ1901

$729.99

top tech deals

$AVE, $AVE, $AVE!

TechXtend Top Tech Deals!

Check out our web site for special-priced offers!

Software, Systems, & Solutions – Since 1982

Prices subject to change. Not responsible for typographical errors.

EDITOR'S NOTE

MICHAEL DESMOND

31 Days of Mango

Jeffrey Blakenburg is a Microsoft senior developer evangelist who last month kicked off a month-long series of daily blog posts titled "31 Days of Mango" (bit.ly/suibvf). The idea: To introduce developers to the features and capabilities of the updated Windows Phone 7 platform and tooling, known widely by the code name "Mango," in a way that was both compelling and valuable.

It's a process Blakenburg has been through before. In December 2009, he published "31 Days of Silverlight" (bit.ly/tDvxFN), a month-long dive into the Microsoft rich Internet application platform, and last year published "31 Days of Windows Phone" (bit.ly/sQomr7). While Blakenburg was concerned about being able to produce a full month of new content based on the updated platform, he needn't have been. As he told me, it quickly became clear that Mango was going to be a significant upgrade. In the end, he said, "I actually had to decide what *wasn't* going to be included."

The project helped Blakenburg come to terms with some underappreciated aspects of the platform, including the concept in Windows Phone 7 Mango of Launchers and Choosers.

"These are tasks that allow a developer to grab the e-mail address of a user-selected contact, for example, or to pre-create an e-mail message for a user so that all they need to do is press the 'Send' button," Blakenburg explains. "My message about these tasks was that we didn't want to give developers direct access to the contact list, because malicious devs will exploit that access. When I discovered that there was a UserData namespace in Mango that delivered all of the user's contact data, I was surprised and delighted."

He also singled out in Mango the new emulator as one of the biggest improvements in the tooling, with its built-in ability to emulate the Accelerometer and GPS sensor.

Learn by Doing

What's perhaps most interesting about the project is the unique dynamic of having to punch out a coherent piece of developer how-to content day in and day out for a month. As Blankenburg told me, the self-imposed rigor is a source of both stress and inspiration. A lot of the challenge is simple time management, because each post takes six hours or more to research and produce, but the payoff is impressive.

"In previous development roles, I found myself constantly referring to the MSDN forums and documentation to understand how a specific concept works. I don't do that anymore with Windows Phone development," Blankenburg says. "I can sit down to work, and it's nothing but building an awesome app. I literally know how to do everything I need to do."

As Blankenburg told me, the self-imposed rigor is a source of both stress and inspiration.

The 31 Days series illustrates the increasingly diverse ecosystem of developer support, which spans the spectrum from blog posts and forum conversations to structured courses and full-length books. Blankenburg says the serial nature of his blog projects enables him to build out concepts, while still respecting the need to make each article stand well on its own.

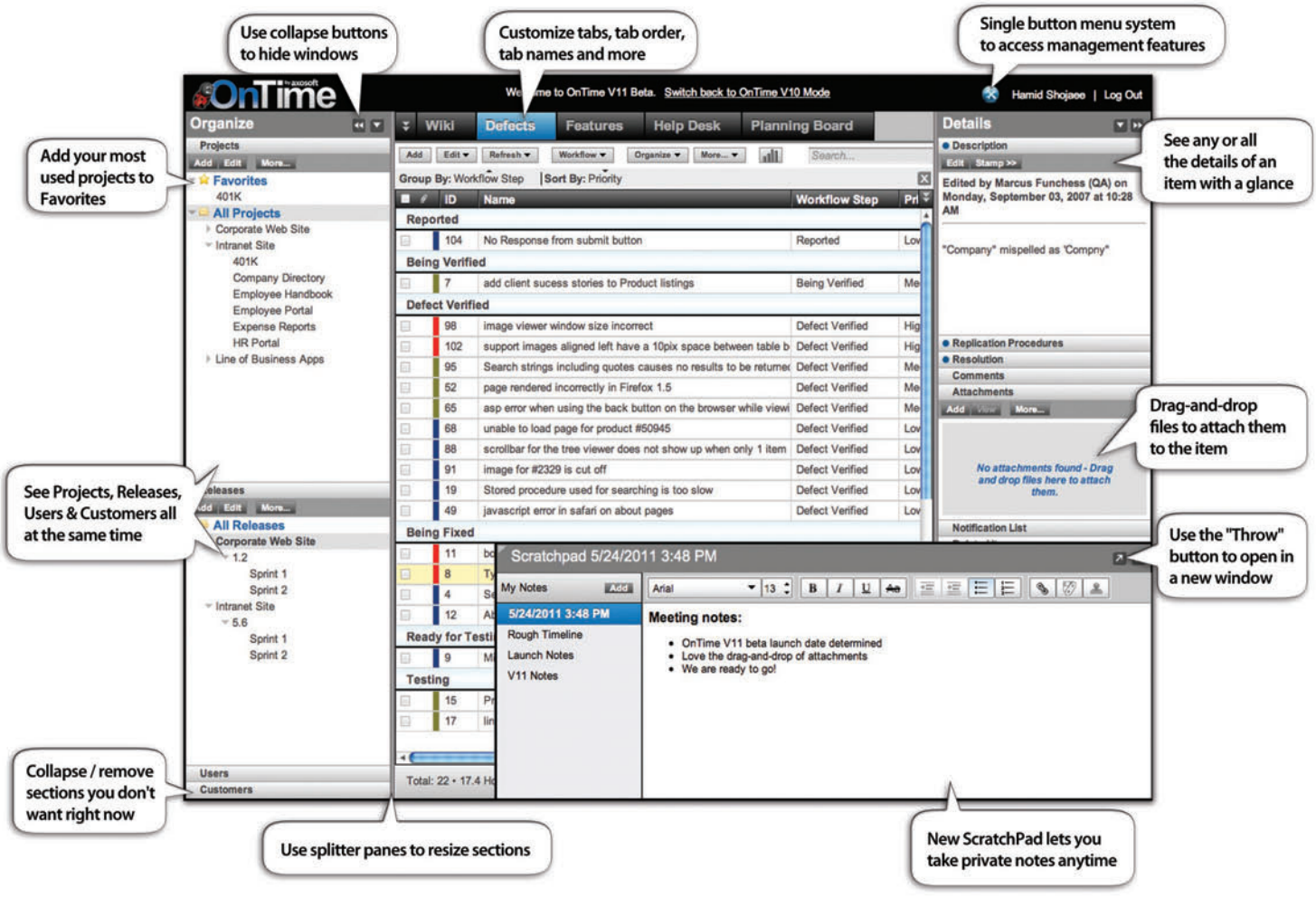
"I think that there are tons of resources out there for developers, but many of them are one-off blog articles that rank well in the search engines. My future vision of the tools Microsoft offers to developers will include not only links to specific parts of the MSDN documentation, but also those articles that solve those specific one-off problems," Blankenburg explains. "By curating a list of articles for each topic, much like they do for MSDN, it would encourage more developers to share their knowledge, and make solving problems significantly easier for those that need it."

OnTime11

axosoft.com
800.653.0024

Everything You Need to Ship Software OnTime

Agile Project Management • Bug Tracking • Backlogs • Burndowns • All in a Beautiful UI



Visit www.axosoft.com for the following:

FREE FOR 2 Users

Never expires, no credit card required. 2 users are totally free forever. Install on your servers or sign up for an account.

Making of OnTime 11

Watch the key principles, the design philosophy and what separates OnTime from other competing products.

Join a Live Demo

Refreshingly informative, and certainly not a sales pitch. We offer 3 live demos a week, so reserve your spot today!

DINO ESPOSITO

A Context-Sensitive Progress Bar for ASP.NET MVC

Most users of a computer application want to receive appropriate feedback from the application whenever it embarks on a potentially lengthy operation. Implementing this pattern in a Windows application is no big deal, although it isn't as straightforward as you might expect. Implementing the same pattern over the Web raises some additional difficulties, which are mostly related to the inherently stateless nature of the Web.

Particularly for Web developers, displaying some static feedback at the beginning of the operation is easy. So the focus of this column isn't on how to display a simple string of text saying "Please wait" or an animated image. Instead, this column tackles the issue of reporting the status of remote operations, providing context-sensitive feedback that faithfully represents the status of the operation for a given session. In Web applications, lengthy tasks occur on the server, and there are no built-in facilities to push state information to the client browser. To achieve this, you need to build your own framework with a mix of client and server code. Libraries such as jQuery are helpful but don't provide a built-in solution—not even in the wide selection of jQuery plug-ins.

In this column—the first of a short series—I'll discuss the most common AJAX pattern you'll find behind any context-sensitive progress framework, and build a solution tailor-made for ASP.NET MVC applications. If you're a Web Forms developer, you might want to take a look at a couple of columns I wrote in in the summer of 2007 targeting Web Forms and AJAX (see my column list at bit.ly/psNZAc).

The 'Progress Indicator' Pattern

The fundamental problem with an AJAX-based progress report is providing feedback about the status of the operation while the user is waiting for a server response. Put another way: The user starts an AJAX operation that takes a while to complete; meanwhile, the user receives updates about the progress made. The architectural problem is that the user isn't going to receive partial responses; the operation returns its response only when all the server-side work has been done. To bring partial responses back to the client, some sort of synchronization between the AJAX client and the server application must be established. The "Progress Indicator" pattern addresses this point.

The pattern suggests that you architect ad hoc server operations that write information about their status to a known location. The status is overwritten every time the operation makes a significant amount of progress. At the same time, the client opens a second channel and makes periodic reads from the same known location. In this way, any changes are promptly detected and reported to the client. More important, the feedback is context-sensitive and real. It represents effective progress made on the server and isn't based on guesses or estimates.

The implementation of the pattern requires that you write your server operations so they're aware of their roles. For example, suppose you implement a server operation based on a workflow. Before starting each significant step of the workflow, the operation will invoke some code that updates a durable store with some task-related information. The durable store can be a database table or a piece of shared memory. The task-related information can be

Figure 1 **A Controller Action Method Using the SimpleProgress Framework**

```
public String BookFlight(String from, String to)
{
  var taskId = GetTaskId();

  // Book first flight
  ProgressManager.SetCompleted(taskId,
    String.Format("Booking flight: {0}-{1} ...", from, to));
  Thread.Sleep(2000);

  // Book return flight
  ProgressManager.SetCompleted(taskId,
    String.Format("Booking flight: {0}-{1} ...", to, from));
  Thread.Sleep(1000);

  // Book return
  ProgressManager.SetCompleted(taskId,
    String.Format("Paying for the flight ..."));
  Thread.Sleep(2000);

  // Some return value
  return String.Format("Flight booked successfully");
}
```

Figure 2 **Invoking and Monitoring a Method via JavaScript**

```
<script type="text/javascript">
  $(document).ready(function () {
    $("#buttonStart").bind("click", buttonStartHandler);
  });

  function buttonStartHandler() {
    new SimpleProgress()
    .setInterval(600)
    .callback(
      function (status) { $("#progressbar2").text(status); },
      function (response) { $("#progressbar2").text(response); })
    .start("/task/bookflight?from=Rome&to=NewYork", "/task/progress");
  }
</script>
```

Code download available at code.msdn.microsoft.com/mag201112CuttingEdge.

LEADTOOLS® 17.5

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

- OCR/OMR
- BARCODE
- PDF & PDF/A
- MEDICAL 3D
- DICOM
- PACS
- MPEG-2 TRANSPORT STREAM
- ANNOTATIONS
- VIEWER CONTROLS
- FORMS RECOGNITION & PROCESSING
- FILE FORMATS
- VIRTUAL PRINTER
- DOCUMENT CLEANUP & PREPROCESSING
- SCANNING
- DVD & DVR
- DIRECTSHOW CODECS
- MULTIMEDIA PLAYBACK & CAPTURE
- IMAGE PROCESSING
- MEDICAL WORKSTATION FRAMEWORK
- MEDICAL WEB VIEWER FRAMEWORK

MAJOR ADDITIONS INCLUDE

PDF READER AND VIEWER WITH TEXT EXTRACTION, BOOKMARKS AND ANNOTATIONS. A FASTER OCR ENGINE WITH MORE ACCURACY AND LANGUAGES. A NEW HIGH LEVEL BARCODE INTERFACE WITH SUPPORT FOR .NET, SILVERLIGHT AND WINDOWS PHONE. A NEW FRAMEWORK FOR CREATING CLOUD APPLICATIONS.

.NET | **C/C++**

SILVERLIGHT/WINDOWS PHONE

ASP.NET | WPF | WF | WCF

FREE 60 DAY EVALUATION - TRY IT TODAY!!!

LEAD Technologies' newly released **LEADTOOLS Version 17.5** is packed with new features and enhancements delivering more speed, power and extensibility into the hands of application developers than ever before.

800 637-1840

WWW.LEADTOOLS.COM

Figure 3 **The Base Class for Controllers that Incorporate Monitored Actions**

```
public class SimpleProgressController : Controller
{
  protected readonly ProgressManager ProgressManager;

  public SimpleProgressController()
  {
    ProgressManager = new ProgressManager();
  }

  public String GetTaskId()
  {
    // Get the header with the task ID
    var id = Request.Headers[ProgressManager.HeaderNameTaskId];
    return id ?? String.Empty;
  }

  public String Progress()
  {
    var taskId = GetTaskId();
    return ProgressManager.GetStatus(taskId);
  }
}
```

a number indicating the percentage of work accomplished or a message that describes the ongoing task.

At the same time, you need a JavaScript-based service that concurrently reads the text from the durable store and brings it back to the client. Finally, the client will use some JavaScript code to merge the text to the existing UI. This can result in some simple text displayed in a DIV tag or in something more sophisticated such as an HTML-based progress bar.

Progress Indicator in ASP.NET MVC

Let's see what it takes to implement the Progress Indicator pattern in ASP.NET MVC. The server operation is essentially a controller action method. The controller can be either synchronous or asynchronous. Asynchrony in controllers is beneficial only to the health and responsiveness of the server application; it doesn't have any impact on the time the user has to wait to get a response. The Progress Indicator pattern works well with any type of controller.

To invoke and then monitor a server operation, you need a bit of AJAX. However, the AJAX library shouldn't be limited to placing the request and waiting for the response. The library should also be able to set up a timer that periodically fires a call to some endpoint that returns the current status of the operation. For this reason, jQuery or the native XMLHttpRequest object are necessary, but not sufficient. So I created a simple JavaScript object to hide most of the extra steps required with a monitored AJAX call. From within an ASP.NET MVC view, you invoke the operation via the JavaScript object.

The controller method responsible for the operation will use the server-side portion of the framework to store the current status in a known (and shared) location, such as the ASP.NET cache. Finally, the controller must expose a common endpoint for the timed requests to call in order to read status in real time. Let's first see the whole framework in action and then move on to explore the internals.

Introducing the SimpleProgress Framework

Written specifically for this article, the SimpleProgress Framework (SPF) consists of a JavaScript file and a class library. The class library defines one key class—the ProgressManager class—that governs the execution of the task and any monitoring activity. **Figure 1** shows a sample controller action method that uses the framework. Note that this (potentially long-running) code should actually go in an asynchronous controller to avoid blocking an ASP.NET thread for too long.

As you can see, the operation is based on three main steps. For the sake of simplicity, the actual action is omitted and is replaced with a Thread.Sleep call. More important, you can see three calls to SetCompleted that actually write the current status of the method to a shared location. The details of the location are buried in the ProgressManager class. **Figure 2** shows what's required to invoke and monitor a controller method.

Note that for the sake of readability, I kept the buttonStartHandler of **Figure 2** out of the document's ready handler. By doing so, however, I add a bit of pollution to the global JavaScript scope by defining a new globally visible function.

You first set a few parameters such as the URL to be called back to grab the current status and the callbacks to be invoked to update the progress bar and to refresh the UI once the lengthy task has completed. Finally, you start the task.

The controller class must incorporate some extra capabilities. Specifically, it must expose a method to be called back. This code is relatively standard, and I hardcoded it into a base class from which you can inherit your controller, as shown here:

```
public class TaskController : SimpleProgressController
{
  ...

  public String BookFlight(String from, String to)
  {
    ...
  }
}
```

The entire SimpleProgressController class is shown in **Figure 3**.

The class has two methods. GetTaskId retrieves the unique task ID that represents the key to retrieve the status of a particular call. As you'll see in more detail later, the task ID is generated by the JavaScript framework and sent over with each request using a custom HTTP

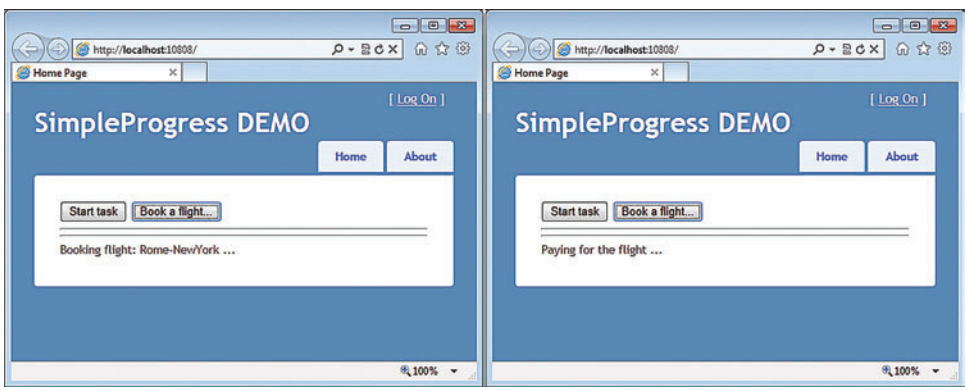


Figure 4 **The Sample Application in Action**

Less Pain, More Gain

check out infragistics.com/reporting

NetAdvantage® for Reporting

EXPORT TO EXCEL, WORD AND PDF

Export reports from the client and server side in the popular format of your choice!

DATA ACCESS SUPPORT

Create MVVM-friendly reports with data accessed from an SQL Server, Oracle or any Object Data Source.

DESIGN INTERFACE

Optimize your data presentation and build attractive reports with an integrated and easy-to-use design-time experience.

REPORT VIEWER

View pixel-perfect reports with vector graphics in our Silverlight, ASP.NET, WPF and Windows Forms report viewer.

INFRAGISTICS
DESIGN / DEVELOP / EXPERIENCE

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

Figure 5 **A Progress Data Provider Based On the ASP.NET Cache Object**

```
public class AspnetProgressProvider : IProgressDataProvider
{
  public void Set(String taskId, String progress, Int32 durationInSeconds=3)
  {
    HttpContext.Current.Cache.Insert(
      taskId,
      progress,
      null,
      DateTime.Now.AddSeconds(durationInSeconds),
      Cache.NoSlidingExpiration);
  }

  public String Get(String taskId)
  {
    var o = HttpContext.Current.Cache[taskId];
    if (o == null)
      return String.Empty;

    return (String) o;
  }
}
```

header. The other method you find on the SimpleProgressController class represents the public (and common) endpoint that the JavaScript framework will call back to get the status of a particular task instance.

Before I get into some implementation details, **Figure 4** will give you a concrete idea of the results the SPF allows you to achieve.

The ProgressManager Class

The ProgressManager class supplies the interface to read and write the current status to a shared store. The class is based on the following interface:

```
public interface IProgressManager
{
  void SetCompleted(String taskId, Int32 percentage);
  void SetCompleted(String taskId, String step);
  String GetStatus(String taskId);
}
```

The SetCompleted method stores the status as a percentage or a simple string. The GetStatus method reads any content back. The shared store is abstracted by the IProgressDataProvider interface:

```
public interface IProgressDataProvider
{
  void Set(String taskId, String progress, Int32 durationInSeconds=300);
  String Get(String taskId);
}
```

The current implementation of the SPF provides just one progress data provider that saves its content in the ASP.NET cache. The key to identify the status of each request is the task ID. **Figure 5** shows a sample progress data provider.

As mentioned, each request for a monitored task is associated with a unique ID. The ID is a random number generated by the JavaScript framework and passed from the client to the server through a request HTTP header.

The JavaScript Framework

One of the reasons the jQuery library became so popular is the AJAX API. The jQuery AJAX API is powerful and feature-rich, and it's endowed with a list of shorthand functions that make placing an AJAX call a breeze. However, the native jQuery AJAX API doesn't support progress monitoring. For this reason, you need a wrapper API that uses jQuery (or any other JavaScript framework you might like) to place the AJAX call while ensuring that a random task ID is generated for each call and the monitor service is activated. **Figure 6** shows an excerpt from the SimpleProgress-Fx.js file in the accompanying code download that illustrates the logic behind the start of a remote call.

Figure 6 **The Script Code to Invoke the SimpleProgress Framework**

```
var SimpleProgress = function() {
  ...
  that.start = function (url, progressUrl) {
    that._taskId = that.createTaskId();
    that._progressUrl = progressUrl;

    // Place the AJAX call
    $.ajax({
      url: url,
      cache: false,
      headers: { 'X-SimpleProgress-TaskId': that._taskId },
      success: function (data) {
        if (that._taskCompletedCallback != null)
          that._taskCompletedCallback(data);
        that.end();
      }
    });

    // Start the callback (if any)
    if (that._userDefinedProgressCallback == null)
      return this;
    that._timerId = window.setTimeout(
      that._internalProgressCallback, that._interval);
  };
}
```

Once the task ID is generated, the function adds the ID as a custom HTTP header to the AJAX caller. Right after triggering the AJAX call, the function sets up a timer that periodically invokes a callback. The callback reads the current status and passes the result to a user-defined function for updating the UI.

I'm using jQuery to perform the AJAX call; in this regard, it's important that you turn off browser caching for AJAX calls. In jQuery, caching is on by default and is automatically turned off for certain data types such as script and JSON With Padding (JSONP).

Not an Easy Task

Monitoring ongoing operations isn't an easy task in Web applications. Solutions based on polling are common but not inevitable. An interesting GitHub project that implements persistent connections in ASP.NET is SignalR (github.com/signalr). Using SignalR, you can solve the same problem discussed here without polling for changes.

In this column, I discussed a sample framework (both client and server) that attempts to simplify the implementation of a context-sensitive progress bar. While the code is optimized for ASP.NET MVC, the underlying pattern is absolutely general and can be employed in ASP.NET Web Forms applications as well. If you download and experiment with the source code, feel free to share your thoughts and feedback. ■

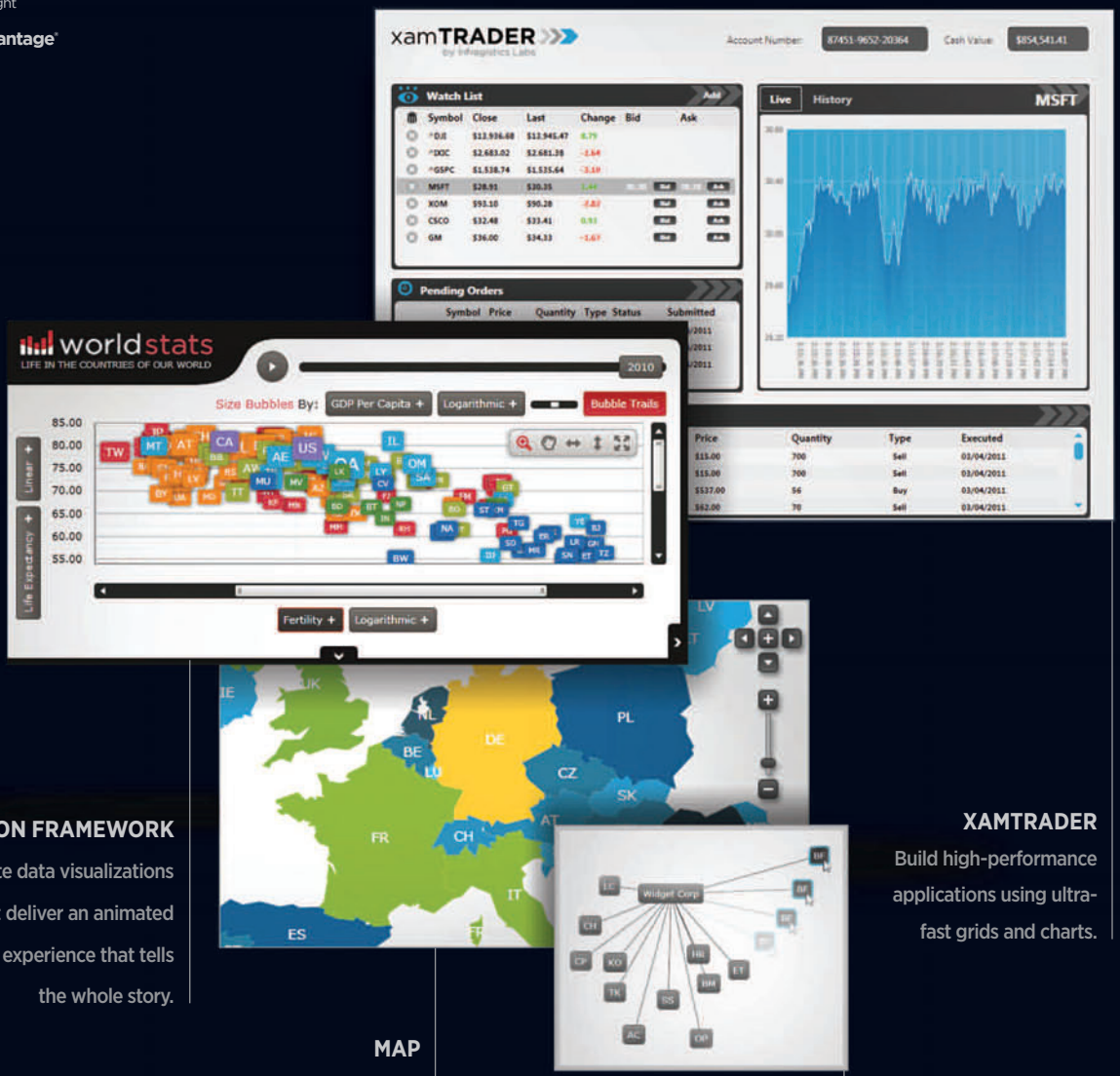
DINO ESPOSITO *is the author of "Programming Microsoft ASP.NET MVC3" (Microsoft Press, 2011) and coauthor of "Microsoft .NET: Architecting Applications for the Enterprise" (Microsoft Press, 2008). Based in Italy, Esposito is a frequent speaker at industry events worldwide. You can follow him on Twitter at twitter.com/despos.*

THANKS to the following technical experts for reviewing this column: *Damian Edwards, Phil Haack and Scott Hunter*

XAML-IFY YOUR APPS

check out infragistics.com/xaml

- **NetAdvantage®** for Silverlight Data Visualization
- **NetAdvantage®** for WPF Data Visualization
- **NetAdvantage®** for Silverlight
- **NetAdvantage®** for WPF



MOTION FRAMEWORK

Create data visualizations that deliver an animated user experience that tells the whole story.

MAP

Ensure your geospatial data really goes places with a feature-laden, interactive Map Control for your applications.

XAMTRADER

Build high-performance applications using ultra-fast grids and charts.

NETWORK NODE

Help your users make the connection with visual representations of simple or complex network relationships.

INFRAGISTICS
DESIGN / DEVELOP / EXPERIENCE

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

KENNY KERR

Thread Pool Timers and I/O

In this, my final installment on the Windows 7 thread pool, I'm going to cover the two remaining callback-generating objects provided by the API. There's even more I could write about the thread pool, but after five articles that cover virtually all of its features, you should be comfortable using it to power your applications effectively and efficiently.

In my August (msdn.microsoft.com/magazine/hh335066) and November (msdn.microsoft.com/magazine/hh547107) columns, I described work and wait objects respectively. A work object allows you to submit work, in the form of a function, directly to the thread pool for execution. The function will execute at the earliest opportunity. A wait object tells the thread pool to wait for a kernel synchronization object on your behalf, and queue a function when it's signaled. This is a scalable alternative to traditional synchronization primitives and an efficient alternative to polling. There are, however, many cases where timers are required to execute some code after a certain interval or at some regular period. This might be because of a lack of "push" support in some Web protocol or perhaps because you are implementing a UDP-style communications protocol and you need to handle retransmissions. Fortunately, the thread pool API provides a timer object to handle all of these scenarios in an efficient and now-familiar manner.

Timer Objects

The CreateThreadpoolTimer function creates a timer object. If the function succeeds, it returns an opaque pointer representing the timer object. If it fails, it returns a null pointer value and provides more information via the GetLastError function. Given a timer object, the CloseThreadpoolTimer function informs the thread pool that the object may be released. If you've been following along in the series, this should all sound quite familiar. Here's a traits class that can be used with the handy unique_handle class template I introduced in my July 2011 column (msdn.microsoft.com/magazine/hh288076):

```
struct timer_traits
{
  static PTP_TIMER invalid() throw()
  {
    return nullptr;
  }

  static void close(PTP_TIMER value) throw()
  {
    CloseThreadpoolTimer(value);
  }
};

typedef unique_handle<PTP_TIMER, timer_traits> timer;
```

I can now use the typedef and create a timer object as follows:

```
void * context = ...
timer t(CreateThreadpoolTimer(its_time, context, nullptr));
check_bool(t);
```

As usual, the final parameter optionally accepts a pointer to an environment so you can associate the timer object with an environment, as I described in my September 2011 column (msdn.microsoft.com/magazine/hh416747). The first parameter is the callback function that will be queued to the thread pool each time the timer expires. The timer callback is declared as follows:

```
void CALLBACK its_time(PTP_CALLBACK_INSTANCE, void * context, PTP_TIMER);
```

To control when and how often the timer expires, you use the SetThreadpoolTimer function. Naturally, its first parameter provides the timer object but the second parameter indicates the due time at which the timer should expire. It uses a FILETIME structure to describe either absolute or relative time. If you're not quite sure how this works, I encourage you to read last month's column, where I described the semantics around the FILETIME structure in detail. Here's a simple example where I set the timer to expire in five seconds:

```
union FILETIME64
{
  INT64 quad;
  FILETIME ft;
};

FILETIME relative_time(DWORD milliseconds)
{
  FILETIME64 ft = { -static_cast<INT64>(milliseconds) * 10000 };
  return ft.ft;
}

auto due_time = relative_time(5 * 1000);
SetThreadpoolTimer(t.get(), &due_time, 0, 0);
```

Again, if you're unsure about how the relative_time function works, please read my November 2011 column. In this example, the timer will expire after five seconds, at which point the thread pool will queue an instance of the its_time callback function. Unless action is taken, no further callbacks will be queued.

You can also use SetThreadpoolTimer to create a periodic timer that will queue a callback on some regular interval. Here's an example:

```
auto due_time = relative_time(5 * 1000);
SetThreadpoolTimer(t.get(), &due_time, 500, 0);
```

In this example, the timer's callback is first queued after five seconds and then every half-second after that until the timer object is reset or closed. Unlike the due time, the period is simply specified in milliseconds. Keep in mind that a periodic timer will queue a callback after the given period elapses, regardless of how long it takes the callback to execute. This means it's possible for multiple callbacks to run concurrently, or overlap, if the interval is small enough or the callbacks take a long enough time to execute.

If you need to ensure callbacks don't overlap, and the precise start time for each period isn't that important, then a different approach for creating a periodic timer might be appropriate. Instead of specifying a period in the call to SetThreadpoolTimer, simply reset the timer in the callback itself. In this way, you

Deliver the Ultimate User Experience

check out infragistics.com/ultimate

NetAdvantage® ULTIMATE

FINANCIAL CHARTING

With support for multiple chart styles, and technical indicators built in, financial charting capabilities are on the money.

TREEMAP

Communicate the relative differences in data weight more effectively, with customizable color and flexible layouts.

OLAP AXIS CHART

Take your data to new depths with the seemingly endless drilldown capability of the OLAP Axis Chart.

OLAP GRID

Provide highly-interactive pivot grid functionality in all of your applications.

INFRAGISTICS
DESIGN / DEVELOP / EXPERIENCE

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

can ensure the callbacks will never overlap. If nothing else, this simplifies debugging. Imagine stepping through a timer callback in the debugger only to find that the thread pool has already queued a few more instances while you were analyzing your code (or refilling your coffee). With this approach, that will never happen. Here's what it looks like:

```
void CALLBACK its_time(PTP_CALLBACK_INSTANCE, void *, PTP_TIMER timer)
{
  // Your code goes here

  auto due_time = relative_time(500);
  SetThreadpoolTimer(timer, &due_time, 0, 0);
}

auto due_time = relative_time(5 * 1000);
SetThreadpoolTimer(t.get(), &due_time, 0, 0);
```

As you can see, the initial due time is five seconds and then I reset the due time to 500 ms at the end of the callback. I have taken advantage of the fact that the callback signature provides a pointer to the originating timer object, making the job of resetting the timer very simple. You may also want to use RAII to ensure the call to SetThreadpoolTimer is reliably called before the callback returns.

You can call SetThreadpoolTimer with a null pointer value for the due time to stop any future timer expirations that may result in further callbacks. You'll also need to call WaitForThreadpoolTimerCallbacks to avoid any race conditions. Of course, timer objects work equally well with cleanup groups, as described in my October 2011 column.

SetThreadpoolTimer's final parameter can be a bit confusing because the documentation refers to a "window length" as well as a delay. What's that all about? This is actually a feature that affects energy efficiency and helps reduce overall power consumption. It's based on a technique called timer coalescing. Obviously, the best solution is to avoid timers altogether and use events instead. This allows the system's processors the greatest amount of idle time, thereby encouraging them to enter their low-power idle states as much as possible. Still, if timers are necessary, timer coalescing can reduce the overall power consumption by reducing the number of timer interrupts that are required. Timer coalescing is based on the idea of a "tolerable delay" for the timer expirations. Given some tolerable delay, the Windows kernel may adjust the actual expiration time to coincide with any existing timers. A good rule of thumb is to set the delay to one-tenth of the period in use. For example, if the timer should expire in 10 seconds, use a one-second delay, depending on what's appropriate for your application. The greater the delay, the more opportunity the kernel has to optimize its timer interrupts. On the other hand, anything less than 50 ms will not be of much use because it begins to encroach on the kernel's default clock interval.

I/O Completion Objects

Now it's time for me to introduce the gem of the thread pool API: the input/output (I/O) completion object, or simply the I/O object. Back when I first introduced the thread pool API, I mentioned that the thread pool is built on top of the I/O completion port API. Traditionally, implementing the most scalable I/O on Windows was possible only using the I/O completion port API. I have written about this API in the past. Although not particularly difficult to use, it was not always that easy to integrate with an application's other threading needs. Thanks to the thread pool API, though, you have the best of both worlds with a single API for work, synchronization, timers and now I/O, too. The other benefit is that performing overlapped I/O completion with the thread pool is actually more intuitive than using the I/O completion port API, especially when it comes to handling multiple file handles and multiple overlapped operations concurrently.

As you might have guessed, the CreateThreadpoolIo function creates an I/O object and the CloseThreadpoolIo function informs the thread pool that the object may be released. Here's a traits class for the unique_handle class template:

```
struct io_traits
{
  static PTP_IO invalid() throw()
  {
    return nullptr;
  }

  static void close(PTP_IO value) throw()
  {
    CloseThreadpoolIo(value);
  }
};

typedef unique_handle<PTP_IO, io_traits> io;
```

> Timer coalescing is based on the idea of a "tolerable delay" for the timer expirations.

The CreateThreadpoolIo function accepts a file handle, implying that an I/O object is able to control the I/O for a single object. Naturally, that object needs to support overlapped I/O, but this includes popular resource types such as file system files, named pipes, sockets and so on. Let me demonstrate with a simple example of waiting to receive a UDP packet using a socket. To manage the socket, I'll use unique_handle with the following traits class:

```
struct socket_traits
{
  static SOCKET invalid() throw()
  {
    return INVALID_SOCKET;
  }

  static void close(SOCKET value) throw()
  {
    closesocket(value);
  }
};

typedef unique_handle<SOCKET, socket_traits> socket;
```

Unlike the traits classes I've shown thus far, in this case the invalid function doesn't return a null pointer value. This is because the WSASocket function, like the CreateFile function, uses an unusual value to indicate an invalid handle. Given this traits class and typedef, I can create a socket and I/O object quite simply:

```
socket s(WSASocket( ... , WSA_FLAG_OVERLAPPED));
check_bool(s);

void * context = ...
io i(CreateThreadpoolIo(reinterpret_cast<HANDLE>(s.get()), io_
completion, context, nullptr));
check_bool(i);
```

The callback function that signals the completion of any I/O operation is declared as follows:

```
void CALLBACK io_completion(PTP_CALLBACK_INSTANCE, void * context, void
* overlapped,
  ULONG result, ULONG_PTR bytes_copied, PTP_IO)
```

The unique parameters for this callback should be familiar if you've used overlapped I/O before. Because overlapped I/O is by nature asynchronous and allows overlapping I/O operations—hence the name overlapped I/O—there needs to be a way to identify the particular I/O operation that has completed. This is the purpose of the overlapped parameter. This parameter provides a pointer to the OVERLAPPED or WSAOVERLAPPED structure that was specified when a particular I/O operation was first initiated. The traditional approach of packing an OVERLAPPED structure into a larger structure to hang more data off this parameter can still be used. The overlapped parameter provides a way to identify the particular I/O operation that has completed, while the context parameter—as usual—provides a context for the I/O endpoint, regardless of any particular operation. Given these two parameters, you should have no trouble coordinating the flow of data through your application. The result parameter tells you whether the overlapped operation succeeded with the usual ERROR_SUCCESS, or zero, indicating success. Finally, the bytes_copied parameter obviously tells you how many bytes were actually read or written. A common mistake is to assume that the number of bytes requested was actually copied. Don't make that mistake: it's the very reason for this parameter's existence.

The only part of the thread pool's I/O support that's slightly tricky is the handling of the I/O request itself. It takes care to code this properly. Before calling a function to initiate some asynchronous I/O operation, such as ReadFile or WSARecvFrom, you must call the StartThreadpoolIo function to let the thread pool know that an I/O operation is about to start. The trick is that if the I/O operation happens to complete synchronously, then you must notify the thread pool of this by calling the CancelThreadpoolIo function. Keep in mind that I/O completion doesn't necessarily equate to successful completion. An I/O operation might succeed or fail both synchronously or asynchronously. Either way, if the I/O operation will not notify the completion port of its completion, you need to let the thread pool know. Here's what this might look like in the context of receiving a UDP packet:

```
StartThreadpoolIo(i.get());

auto result = WSARecvFrom(s.get(), ...

if (!result)
{
  result = WSA_IO_PENDING;
}
else
{
  result = WSAGetLastError();
}

if (WSA_IO_PENDING != result)
{
  CancelThreadpoolIo(i.get());
}
```

As you can see, I begin the process by calling StartThreadpoolIo to tell the thread pool that an I/O operation is about to begin. I then call WSARecvFrom to get things going. Interpreting the result is the crucial part. The WSARecvFrom function returns zero if the operation completed successfully, but the completion port will still be notified, so I change the result to WSA_IO_PENDING. Any other result from WSARecvFrom indicates failure, with the exception, of course, of WSA_IO_PENDING itself, which simply means that the operation has been successfully initiated but it will be completed later. Now, I simply call CancelThreadpoolIo if the result is not pending to keep the thread pool up to speed. Different I/O endpoints may provide different semantics. For example, file I/O can be configured to avoid notifying the completion port on synchronous completion. You would then need to call CancelThreadpoolIo as appropriate.

Like the other callback-generating objects in the thread pool API, pending callbacks for I/O objects can be canceled using the WaitForThreadpoolIoCallbacks function. Just keep in mind that this will cancel any pending callbacks, but not cancel any pending I/O operations themselves. You will still need to use the appropriate function to cancel the operation to avoid any race conditions. This allows you to safely free any OVERLAPPED structures, and so forth.

And that's it for the thread pool API. As I said, there's more I could write about this powerful API, but given the detailed walk-through I've provided thus far, I'm sure you're well on your way to using it to power your next application. Join me next month as I continue to explore Windows with C++. ■

Kenny Kerr *is a software craftsman with a passion for native Windows development. Reach him at kennykerr.ca.*

JULIE LERMAN

Handling Entity Framework Validations in WCF Data Services

I'm writing this column on the heels of the Microsoft BUILD conference. The core of all of the excitement at BUILD was, of course, the new Metro UI for Windows 8 that sits on top of the new Windows Runtime (WinRT). If you're a data geek, you might've already looked to see what options exist for providing data to "Metro style" apps. In this early preview, you can provide data from file storage or from the Web. If you want to interact with relational data, Web-based options include XML or JSON over HTTP, sockets and services. On the services front, Metro-style apps will provide client libraries for consuming OData, which means that any experience you have today working with OData through the Microsoft .NET Framework, Silverlight or other client libraries will give you a big advantage when you're ready to consume OData in your Metro-style applications.

With that in mind, I'll devote this column to working with OData. The Entity Framework (EF) release that contains Code First and the DbContext introduced a new Validation API. I'll show you how to take advantage of built-in server-side validation when your EF Code First model is being exposed as OData through WCF Data Services.

Validation API Basics

You might already be familiar with configuring attributes such as Required or MaxLength to class properties using Data Annotations or the Fluent API. These attributes can be checked automatically by the new Validation API. "Entity Framework 4.1 Validation," an article in the MSDN Data Developer Center (msdn.microsoft.com/data/gg193959), demonstrates this, as well as how to apply rules with the IValidatableObject interface and the ValidateEntity method. While you might already be validating Data Annotations and IValidatableObject on the client side, their rules can also be checked on the server side along with any ValidateEntity logic that you've added. Alternatively, you can also choose to trigger validation on demand in your server code.

Here, for example, is a simple Person class that uses two Data Annotations (the first specifies that the LastName property is required and the other sets a maximum length for the IdentityCard string field):

```
public class Person
{
  public int PersonId { get; set; }
  public string FirstName { get; set; }
  [Required]
  public string LastName { get; set; }
  [MaxLength(10)]
  public string IdentityCardNumber { get; set; }
}
```

This column discusses the March 2011 WCF Data Services CTP. All information is subject to change.

Code download available at code.msdn.microsoft.com/mag201112DataPoints.

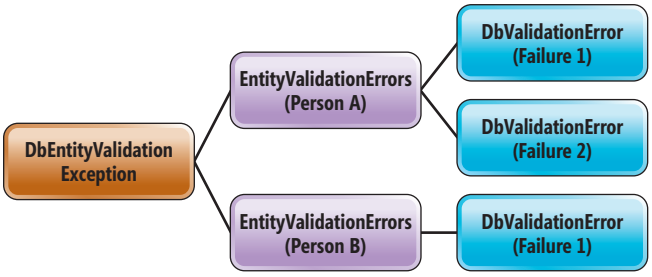


Figure 1 **DbEntityValidationException Contains Grouped Sets of Errors**

By default, EF will perform validation when SaveChanges is called. If either of these rules fails, EF will throw a System.Data.Entity.DbEntityValidationException—which has an interesting structure. Each validation error is described in a DbValidationError, and DbValidationErrors are grouped by object instance into sets of EntityValidationErrors.

For example, **Figure 1** shows a DbEntityValidationException that would be thrown if EF detected validation problems with two different Person instances. The first EntityValidationErrors object contains a set of DbValidationErrors for a single Person instance where there were two errors: no LastName and the IdentityCard had too many characters. The second Person instance had a single problem; therefore, there's only one DbValidationError in the second EntityValidationErrors object.

In the MSDN Data Developer Center article I mentioned, I showed the exception being passed back to a Model-View-Controller (MVC) application that knew how to discover and display the specific errors.

In a distributed application, however, the errors might not make it back to the client side to be used and reported so easily. While the top-level exception may be returned, the client application may

Figure 2 **Building a More Useful Exception Message**

```
protected override void HandleException(HandleExceptionArgs args)
{
  if (args.Exception.GetType()==
    typeof(DbEntityValidationException))
  {
    var ex=args.Exception as DbEntityValidationException;

    var errors = ex.EntityValidationErrors.First().ValidationErrors.ToList();
    var errorMessage=new StringBuilder();
    foreach (System.Data.Entity.Validation.DbValidationError e in errors)
    {
      errorMessage.AppendLine(e.ErrorMessage);
    }
    args.Exception = new DataServiceException(500, errorMessage.ToString());
  }
}
```

ComponentOne Ultimate™ delivers the tools and resources to build everything …everywhere. Whether you're a Windows, Web, or XAML developer, this ultimate dev tool collection delivers. Inside you'll find: 100s of .NET controls, OLAP data analysis controls, SharePoint Web Parts, documentation tools, LightSwitch extensions, and tools for ADO.NET Entity Framework and RIA Services. No job is too big. Bring speed, style, and functionality to your all your applications ...it is your destiny.

BUILD EVERYTHING EVERYWHERE

THE GALAXY IS YOURS: 100S OF .NET CONTROLS & TOOLS

COMPONENTONE
ULTIMATE

MEET YOUR DESTINY AT:
COMPONENTONE.COM/GALAXY

ComponentOne

Figure 3 **Parsing and Displaying the Error Message Returned from the Service**

```
try
{
  context.SaveChanges();
}

catch (Exception ex)
{
  var sr = new StringReader(ex.InnerException.Message);
  XElement root = XElement.Load(sr);
  IEnumerable<XElement> message =
    from el in root.Elements()
    where el.Name.LocalName == "message"
    select el;
  foreach (XElement el in message)
    Console.WriteLine(el.Value);

  Console.ReadKey();
}
```

have no idea how to drill into a DbEntityValidationException to find the errors. With many apps, you may not even have access to the System.Data.Entity namespace and therefore no knowledge of the DbEntityValidationException.

More problematic is how WCF Data Services transmits exceptions by default. On the client side, you only get a message telling you "An error occurred while processing this request." But the critical phrase here is "by default." You can customize your WCF Data Services to parse DbEntityValidationExceptions and return useful error information to the client. This is what I'll focus on for the rest of this column.

WCF Data Service Results Hide Validation Errors by Default

My model is hosted in a DbContext data layer I've called PersonModelContext:

```
public class PersonModelContext : DbContext
{
  public DbSet<Person> People { get; set; }
}
```

I have a simple data service that exposes the Person type from this context for reading and writing:

```
public class DataService : DataService<PersonModelContext>
{
  public static void InitializeService(DataServiceConfiguration config)
  {
    config.SetEntitySetAccessRule("People", EntitySetRights.All);
    config.DataServiceBehavior.MaxProtocolVersion =
      DataServiceProtocolVersion.V3;
  }
}
```

Because I'm using Code First, I would have to do some tweaks to get WCF Data Services to work with it. Instead of tweaking, I've replaced the Microsoft .NET Framework 4 System.Data.Services and System.Data.ClientServices with the Microsoft.Data.Services and Microsoft.Data.ClientServices libraries from the March 2011 WCF Data Services CTP (see bit.ly/mTI69m), which has those tweaks built in. That's why the DataServiceProtocolVersion is set to V3.

Figure 4 **Parsed Error Message Displayed in the Client**

Finally, I'm consuming the service with a simple console app that uses the following method to insert a Person:

```
private static void InsertPersonNoLastName()
{
  var person = new Person
  {
    FirstName = "Julie",
    IdentityCardNumber="123456789",
  };
  var context = new PersonModelContext
   (new Uri("http://localhost:43447/DataService.svc"));
 context.AddToPeople(person);
 context.SaveChanges();
}
```

Notice I've neglected to set the LastName property. Because LastName is configured to be required, the EF will throw an exception to the data service, but the console app will just receive a DataServiceRequestException with the message described earlier ("An error occurred while processing this request."). If you drill into the inner exception, you'll find that it contains the same message and no additional details.

WCF Data Services does have a setting to let you send back exception messages with more details by adding the following to the InitializeService method:

```
#if DEBUG
  config.UseVerboseErrors = true;
#endif
```

Now the inner message (contained in the XML response from the service) tells you: "Validation failed for one or more entities. See 'EntityValidationErrors' property for more details." But unfortunately, the EntityValidationErrors do not get passed back with the exception. So you know that the Validation API found one or more problems, but you can't discover anything more about the error. Note that I wrapped UseVerboseErrors in a compiler directive. UseVerboseErrors should only be used for debugging—you don't want it in your production code.

Overriding the HandleException Method

WCF Data Services exposes a virtual (Overrideable) method called HandleException. This lets you capture any exception that happens in the service, analyze it and construct your own DataServiceException to return to the caller. It is in this method that you can parse out any Validation errors and return more meaningful information to the calling application. The signature of the method is:

```
protected override void HandleException(HandleExceptionArgs args)
```

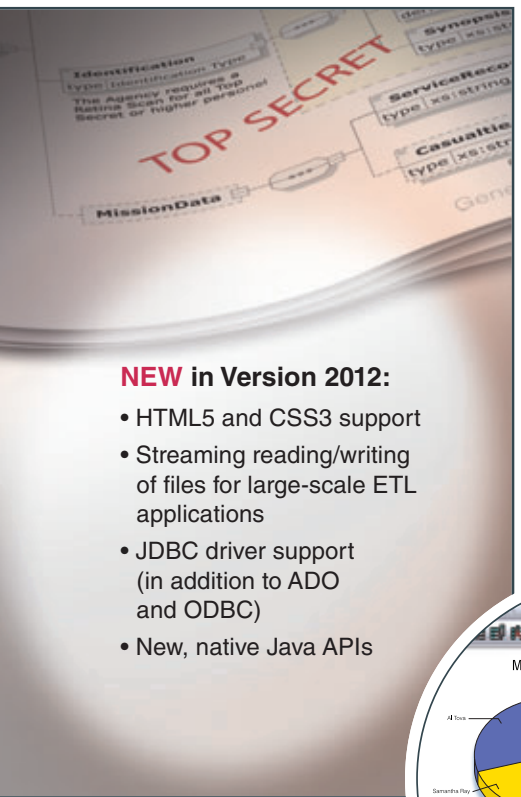
The HandleExceptionArgs type has a number of properties: Exception, ResponseContentType, esponseStatusCode, ResponseWritten, UseVerboseErrors

Of interest to me is the Exception property. This is where you can capture and identify exceptions thrown by the Validation API—DbEntityValidationException. You can also handle any other types of errors here, but I will focus on looking for and parsing the validation exceptions. I've got the System.Data.Entity.Validation namespace in my using statements at the top of the class so that I don't have to strongly type the exception.

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

ALTOVA® missionkit®

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



NEW in Version 2012:

- HTML5 and CSS3 support
- Streaming reading/writing of files for large-scale ETL applications
- JDBC driver support (in addition to ADO and ODBC)
- New, native Java APIs

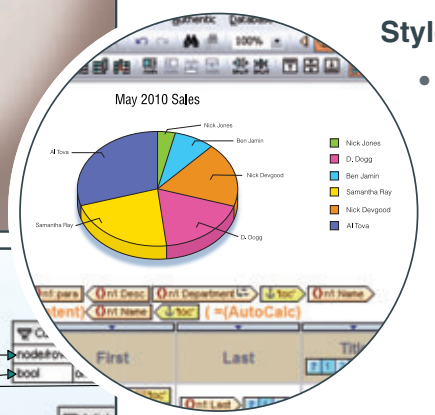
The Altova MissionKit includes multiple intelligent XML tools:

XMLSpy® – industry-leading XML editor

- Support for all XML-based technologies
- Editing of HTML4, HTML5, XHTML, CSS
- Graphical editing views, powerful debuggers, code generation, & more

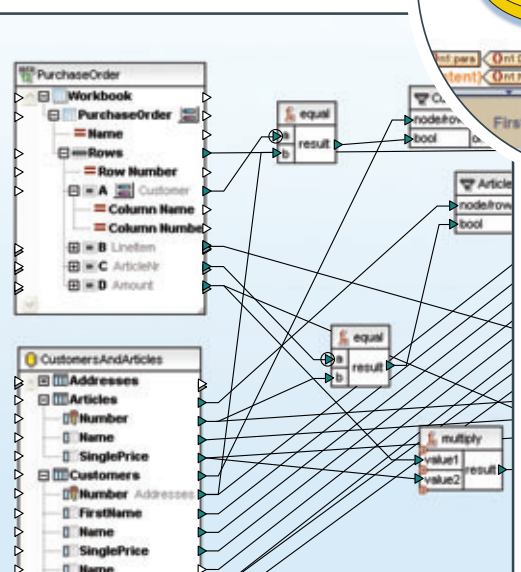
MapForce® – graphical data mapping & ETL tool

- Drag-and-drop data conversion with code generation
- Mapping of XML, DBs, EDI, Excel®, XBRL, flat files & Web services



StyleVision® – visual stylesheet & report designer

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



Download a 30 day free trial!

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

Scan to learn more about MissionKit XML tools

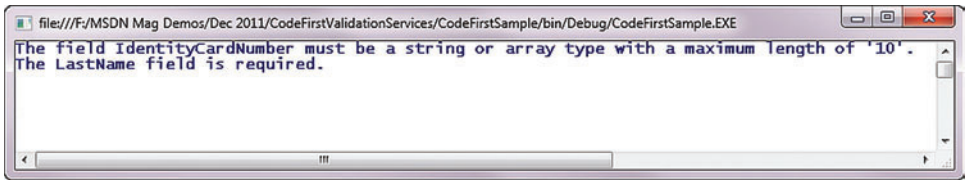


Figure 5 **Console App Displaying Multiple Errors for a Single Entity**

I'll start out with the presumption that only a single entity is being validated, which is why I'm only querying for the first EntityValidationErrors contained in the exception, as shown in **Figure 2**. If you want the service to validate multiple objects, be sure to use the SaveChangesOptions.Batch parameter when you call SaveChanges. Otherwise, only one object will be saved and validated at a time and once you hit an error, no more objects will be saved or validated.

What's happening in this method is that I first check to see if the exception is the type thrown by the Validation API. If it is, I pull the exception into the variable "ex." Next, I query for a list of all of the DbValidationErrors contained in the first set of EntityValidationErrors in the exception. Then I build up a new error string using the ErrorMessage property of each EntityValidationError and pass that string back to the calling application in a new DataServiceException. EntityValidationError has other properties, but it builds up a complete error message using the name of the property and the validation problem into the ErrorMessage. With Code First you can specify a custom error message, but I'm happy with the defaults for the purposes of this demonstration. In this example, the message is "The LastName field is required." Note that the constructor for DataServiceException has a number of overloads. I'm keeping it simple by just providing the "internal server error" 500 code and a string with the message I want to relay.

Parsing the New Exception on the Client

Now, on the client side, you'll still get an exception that says "An error occurred while processing this request," but this time the inner exception contains the message "The LastName field is required."

But it's not a simple string. The message of a DataServiceRequestException is formatted in an HTTP Response because the request is made over HTTP:

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<error xmlns=
  "http://schemas.microsoft.com/ado/2007/08/dataservices/metadata">
  <code></code>
  <message xml:lang="en-US">The LastName field is required.&#xD;
  </message>
</error>
```

One of the overloads for the DataServiceException I constructed in the service allows you to insert custom error codes. If I had used that, the custom code would show up in the <code> element of the error. If you're calling the service from a Web app, you may be able to display the HTTP response directly in your UI. Otherwise, you'll probably want to parse it so that you can handle the exception using whatever patterns you're using in your application for dealing with errors.

I'm using LINQ to XML to extract the message and then I can display it in my console application. I call SaveChanges in a try/catch block, parsing and displaying the error message (see **Figure 3**). **Figure 4** shows the results of the client-side exception.

Now I'll throw another wrench into the InsertPerson method. In addition to neglecting the LastName property, I'll put too many characters into the IdentityCard property. Remember that this property was configured to have a MaxLength of 10:

```
var person = new Person
{
  FirstName = "Julie",
  IdentityCardNumber="123456789ABCDE"
};
```

Now the HandleException method will find two DataValidationErrors for the Person instance that the service attempted to update. The StringBuilder will contain a two-line message—one describing the problem with the LastName property and another to explain the problem with the IdentityCard property.

In the console application, this will be seen as a single message in the exception:

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<error xmlns=
  "http://schemas.microsoft.com/ado/2007/08/dataservices/metadata">
  <code></code>
  <message xml:lang="en-US">The field IdentityCardNumber must be a
string or array type with a maximum length of '10'.&#xD;
    The LastName field is required.&#xD;
  </message>
</error>
```

The LINQ to XML parser will then relay the message to the console, as shown in **Figure 5**.

Benefit from Validation Even When Disconnected

You've now seen how, using a simple set of requirements applied using Code First Data Annotations, you can capture and parse EF validation exceptions, return them to a client and, on the client side, parse the exception returned through HTTP. Whether you're working with validation rules applied through property configurations or more complex rules that you can specify with IValidationObject or by overriding the ValidateEntity method, the EF will always return DbEntityValidationExceptions. You now know how to parse through those and can expand the logic to accommodate multiple objects, provide error messages containing more details, and handle them on the server or on the client as required by your application.

Because WCF Data Services returns OData, you can consume these services and leverage the validation today and practice so that you can be ready to do the same with future Metro-style technologies. ■

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

Thanks to the following technical expert for reviewing this article: *Mike Flasko*

Kendo UI

THE ART OF WEB DEVELOPMENT

Everything You Need

For JavaScript & HTML5 development

Kendo UI is a complete framework for JavaScript developers. It provides everything you need to build HTML5 and JavaScript sites and apps, including a powerful data source, crazy-fast templating, rich UI widgets, and customizable themes. Don't waste time piecing together a JavaScript framework. Download Kendo UI and start developing amazing JavaScript apps right away.

Download the future of JavaScript development
at **www.kendoui.com** or scan

Your Data on Any Device.

Develop once in .NET and XAML, deliver anywhere via HTML5 without writing a single line of Javascript code.

ComponentArt
Data Visualization

Industry-leading developer tools for building sophisticated XAML-based digital dashboards and data analysis applications. Publish your dashboards in Silverlight/WPF, or deploy to ComponentArt Dashboard Server for automatic HTML5 conversion.

ComponentArt
Dashboard Server

Enables rendering of XAML-based dashboards to any device. Supports HTML5, Silverlight, WPF, PDF and image output formats. Works with any data source. Can be deployed on dedicated physical servers or within Windows Azure cloud environment.

Supported Devices: Apple iPad & iPhone, Android tablets & phones, Windows tablets & phones, Blackberry Playbook & phones, any Windows or Mac OS computer.

Powered By:  HTML5 &  XAML

ComponentArt

Mobile Dashboards

ComponentArt's unique technology delivers rich BI dashboards to any tablet, smartphone, PC or Mac, while being perfectly positioned for Windows 8.

Try them live at ComponentArt.com

ComponentArt

Your Data in a Whole New Light

JOSEPH FULTZ

Completing the Trip with AppFabric Queues

In the October issue, I touched upon some of the new features in the Windows Azure AppFabric Service Bus (msdn.microsoft.com/magazine/hh456395). This month, I'll continue with the scenario I started by following the return trip back. So far in the scenario, a store has requested an inventory check from nearby stores by publishing inventory check requests to a Topic. The stores subscribed to the Topic received the requests based on a Filter on the subscription that limited messages to those within their region and not sent by them.

For the return trip, I'll rely on a couple of features of Windows Azure AppFabric Service Bus Queues. First, there are two important properties to which I'll assign values on the BrokeredMessage that's sent to the Topic. The first property to which I'll assign value is the ReplyTo property, to tell the recipient where to send the message. This will be a specific Queue created by the sender at the time it sends the message. I want to use a Queue instead of a Topic in this case, because the response is going to one recipient, unlike the request pattern, which equated to a broadcast for help to anyone listening.

The new part of the flow of the message is shown in **Figure 1**, as that which flows to the right of the row of stores.

Updating the Outbound Request Code

In the first pass, I got the message out to the Topic and demonstrated picking it up by the appropriate subscriptions, but two important items were left undone to support a nice, easy response. The first item is setting the CorrelationId property of the BrokeredMessage. The property is part of the API, so I don't have to include it as a part of my data class or schema. Because it's a string, it could be something meaningful, but it should be something that will uniquely identify the message such that any response that has a matching CorrelationId can't be confused to match some other request in the system. For my sample purposes, I use a GUID, but that's a pretty safe bet in practice, too. Here's the code:

```
BrokeredMessage msg = BrokeredMessage.CreateMessage(data);
// Add props to message for filtering
msg.Properties["Region"] = data.Region;
msg.Properties["RequesterID"] = data.RequesterID;

// Set properties for message
msg.TimeToLive = TimeSpan.FromSeconds(30);
msg.ReplyTo = returnQueueName;
msg.CorrelationId = Guid.NewGuid().ToString();
```

The BrokeredMessage also has a property named SessionId, which I'm not using here. SessionId is another property for logical grouping that's nice to have at the message envelope level, because it facilitates grouping of messages that are all related. The question arises of how it differs in intent to CorrelationId. There are two scenarios where the SessionId could be particularly useful. The first is in a system where multiple daemons are making various requests that are all related. The CorrelationId would be used to route the response to the requesting processor. The SessionId would be used to group all of the messages sent and received across processing nodes. Such grouping is useful in determining the state of processing in a system for analytics and debugging. For the same reasons, this is a useful construct in a system that makes many requests as part of an overall process (for example, purchase process, checking inventory, payment verification, send to fulfillment and so on), but the exact flow and timing isn't guaranteed.

Having set the CorrelationId on the outbound message, the next change I need to make is to set the ReplyTo property. I could create another Topic and have all stores monitor for response or use a single queue, but that would create unnecessary traffic and under times of load be more likely to cause a bottleneck. Thus, it makes sense to simply create a response queue at the time of request and let the recipient know where that is. Because this is a string, it could be anything, though I would suggest a fully qualified name to prevent any confusion or collision in future evolutions of the software. You could start with only the queue name, but under maintenance and in expansion, this could lead to confusion as the system starts to support multiple service bus namespaces and subqueues. Additionally, a fully qualified address will be better for recipients not using the Microsoft .NET Framework.

The last two things I do before I send the message off is to create the response queue and start a timer to check for responses. I started with the GetQueue method. At the time of this writing, the documentation (bit.ly/pnByFw) for GetQueue states for the "Return Value" that "type" equals Microsoft.ServiceBus.Messaging.Queue; a Queue handle to the queue; or null if the queue doesn't exist in the service namespace. However, this isn't the case. In fact, it will throw an exception:

```
// Check if-exists and create response queue
try
{
  returnQ = this.ServiceBusNSClient.GetQueue(returnQueueName);
}
catch (System.Exception ex)
{
  Debug.WriteLine(ex.Message);
}

if (returnQ == null)
{
  returnQ = this.ServiceBusNSClient.CreateQueue(returnQueueName);
}

checkQTimer.Enabled = true;
```

This article discusses the Windows Azure AppFabric SDK CTP – June Update. All related information is subject to change.

Code download available at code.msdn.microsoft.com/mag201112Cloudy.

Hey Developers
Are you working with FILES?

Aspose provides premium file APIs for most business-centric formats.
.NET, Java, SharePoint, SQL Reporting & JasperReports

Aspose.Words

DOC, DOCX, RTF, HTML, PDF, XPS & other document formats.

Aspose.Cells

XLS, XLSX, XLSM, XLTX, CSV SpreadsheetML & image formats.

Aspose.Slides

PPT, PDF, PPS, POTX, PPTX & other formats.

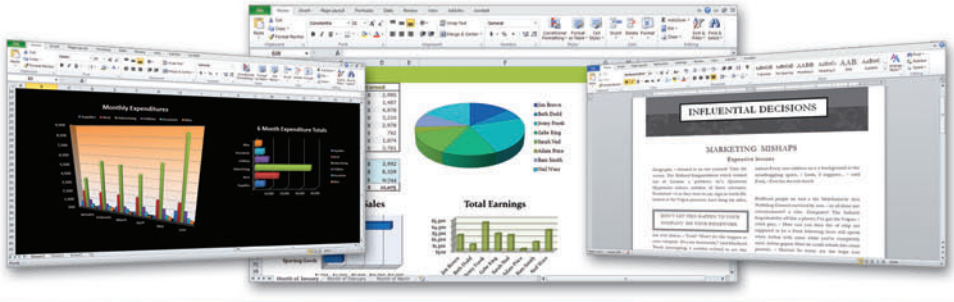
Aspose.Pdf Aspose.BarCode Aspose.Tasks
Aspose.Email Aspose.Diagram Aspose.OCR

Features:

- Create
- Edit
- Convert
- Import
- Export
- Render
- Save
- Print

When you buy Aspose.Total, you will receive our complete suite and any future products.

100% Standalone - No Office Automation



ASPOSE
Your File Format Experts

Get your FREE evaluation copy at **http://www.aspose.com**.

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

Enterprise Sales – enterprise.sales@aspose.com

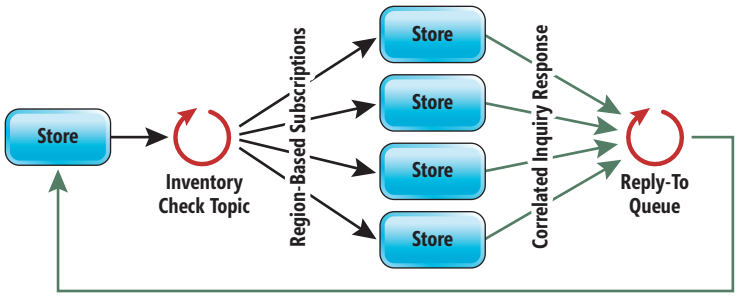


Figure 1 **Inquiry and Response Round-Trip**

Thus, I've wrapped the GetQueue method in a try-catch block and moved on. My sample code, as is typical, does nothing more than write out the error. Once the queue is created, I assign it to a variable so I can reference it for checking the queue and then enable the timer that I set up when the app started.

Updating the Recipient and Responding

With the proper modifications on the sender's side, I have to make some additions so I can respond to the requests. I'm going to use a mostly hardcoded response, but I'm going to list the messages in a grid so I can select the one to which I'll reply. I've set up an event mechanism for notifying the UI of new messages. Once a message is received, I set the current recipient's store ID as the responder and then notify the UI of the message for display:

```
recvSuccess = msgReceiver.TryReceive(TimeSpan.FromSeconds(3), out NewMsg);
if (recvSuccess)
{
  CreateInquiry.InventoryQueryData qryData =
    NewMsg.GetBody<CreateInquiry.InventoryQueryData>();

  NewMsg.Properties.Add("ResponderId", this.StoreID);
  EventContainer.RaiseMessageReceived(NewMsg);

}
```

Within the UI, I've subscribed to the event that will receive the message and I then pass it on to a method to update the UI elements that will do the necessary InvokeRequired method check:

```
void EventContainer_MessageReceivedEvent(object sender, MessagingEventArgs e)
{
  BrokeredMessage msg = (BrokeredMessage)e.MessageData;

  var RequestObject =
    msg.GetBody<CreateInquiry.InventoryQueryData>();
  RequestObject.ResponderID = msg.Properties["ResponderId"].ToString();
  this.ReceivedMessages.Add(RequestObject);
  UpdateUIElements(msg.MessageId, RequestObject);

}
```

With the code complete to fetch the messages for the Topic and to update the UI, I can now visualize the messages coming in (see **Figure 2**).

This UI (I wasn't hired for my user experience design skills, obviously) will allow me to select one of the messages and respond to it with a quantity set in the bottom text box. The response code will be a pretty simple and short task, as I only have to add a small bit of code to set the quantity on the message object and then send it to the Queue specified in the message's ReplyTo property.

For this sample, I'm simply adding received messages from the Topic to a Dictionary<string, BrokeredMessage> object where I'm using the BrokeredMessage.MessageId as the key for the dictionary. I simply use the Message Id from the grid to retrieve it from the dictionary and then create a new BrokeredMessage assigning the same CorrelationId and assigning value to the ResponderId and Quantity properties. Just as when receiving from the Topic, I'll use the MessagingFactory to create a QueueClient and from that object a MessageSender:

```
// Send message
SharedSecretCredential credential =
  TransportClientCredentialBase.CreateSharedSecretCredential(
  Constants.issuerName, Constants.issuerKey);
Uri sbUri = ServiceBusEnvironment.CreateServiceUri(
  "sb", Constants.sbNamespace, String.Empty);

MessagingFactory Factory =  MessagingFactory.Create(sbUri, credential);
QueueClient QClient = Factory.CreateQueueClient(msg.ReplyTo);
MessageSender Sender = QClient.CreateSender();

Sender.Send(ResponseMsg);
```

That sends response back, so we have to move our focus back to the originating app to process the response.

Receiving the Response

For this sample, I just want to pull the responses off of the queue. Because the code to send the request was modified to start monitoring the ReplyTo queue, I really should add the actual code in to check the queue. I start off creating a MessageReceiver and set up a simple While loop to get all of the messages available on the queue this time around:

```
void checkQTimer_Tick(object sender, EventArgs e)
{
  MessageReceiver receiver =
    QClient.CreateReceiver(ReceiveMode.ReceiveAndDelete);
  BrokeredMessage NewMessage = null;
  while (receiver.TryReceive(TimeSpan.FromSeconds(1), out NewMessage))
  {
    InquiryResponse resp = new InquiryResponse();
    resp.CorrelationID = NewMessage.CorrelationId;
    resp.Message = NewMessage;
    InquiryResponses.Add(resp);
  }
 }
```

As before, I use the TryReceive method. While it works for this sample, I'd consider doing it a little differently for a real UI, because the method blocks the thread, thus lending itself to be better

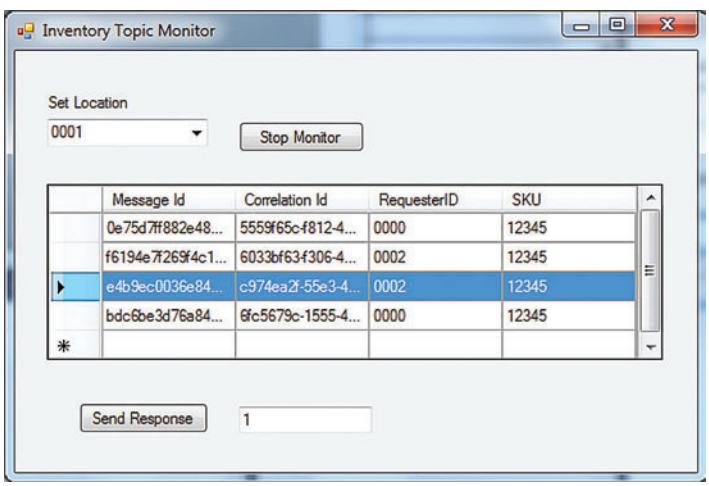


Figure 2 **The Inventory Topic Monitor**

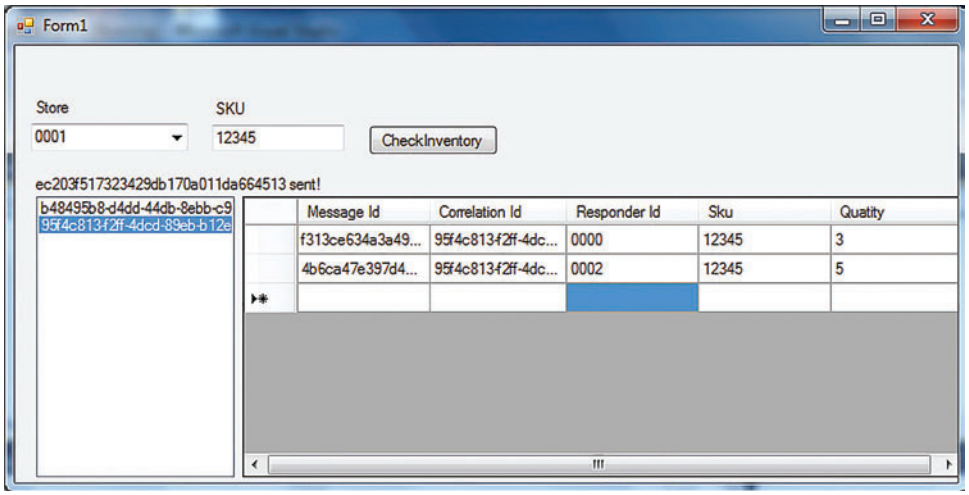


Figure 3 **Inquiry Request and Responses**

executed on a different thread. I want to fetch the entire BrokeredMessage from the list by CorrelationId, so I've created an object and use the CorrelationId to filter the object out later. I want the message as a BrokeredMessage, because I want data that is part of the BrokeredMessage envelope that now encapsulates my InquiryData object (see **Figure 3**).

Modifying the SelectedIndexChanged code of the ListBox, I simply grab the CorrelationId that I used as the item and use it to get the responses of interest out of the list I'm building as responses show up on the queue:

```
string correlationId =
  lbRequests.SelectedItem.ToString();

List<InquiryResponse> CorrelatedList =
  InquiryResponses.FindAll(
  delegate(InquiryResponse resp)
{
  return resp.CorrelationID == correlationId;
});
```

The last bit of work is to add the responses to the DataGrid to see the stores that responded to my inventory inquiry. Adding to the beauty of my barren UI, you'll notice that I'm using GUIDs, but I hope that it's obvious to the reader that this would be replaced by user-friendly descriptions, leaving the nasty GUIDs and IDs for the extended details pages.

Review

After growing up as an Army brat, I have a lot of sayings and other such things burned into my head. One example is the "Military Method" of education, which is that I'll tell you what we're going to do, we'll do it, and then I'll tell you what we did. Well, I'm at the "what we did" part. I started this writing as a single entry, but found that addressing the complete round-trip to be too much for a single column. So I split it up into two entries: one to get the request out and the second to get the response back. My goal was to present the basic features of the Windows Azure AppFabric ServiceBus and give a general feel for using it. When thinking about technology, I always like to wrap it in some type of context, and in this case I wanted the context to be an inter-store product query, because it's a feasible scenario and it makes sense to use a mix of both Topics and Queues.

As of this writing, the May CTP of the Windows Azure AppFabric ServiceBus has been released and some information on it can be found at bit.ly/it5Wo2. Additionally, you can get involved on the forums at bit.ly/oJyCYx. ■

JOSEPH FULTZ *is a software architect at Hewlett-Packard Co., working as part of the HP.com Global IT. Previously he was a software architect for Microsoft working with its top-tier enterprise and ISV customers defining architecture and designing solutions.*

THANKS to the following technical expert for reviewing this article: *Jim Keane*

The Model-View-Presenter-ViewModel Design Pattern for WPF

Bill Kratochvil

Of all the successful projects I've been a part of, the most successful ones shared a common result: As the application grew larger, the codebase became smaller. On the surface, this may seem contradictory, but coding in an Agile environment lends itself to a shrinking codebase. As requirements change, refactoring occurs, and this refactoring opportunity—coupled with hindsight—provides the ability to reuse existing components more efficiently while eliminating duplicate code in the process.

In contrast, there have been some monolithic projects, which were sometimes deemed successful, where the source code grew and grew with little opportunity for reuse. These projects became resource-intensive and a risk to future growth. What was the fundamental difference? The infrastructural design pattern used. The patterns you use will determine whether you paint yourself into a corner or keep your back to the open opportunities the future might bring.

In this article, I'll present one such pattern—overlooked by many Windows Presentation Foundation (WPF) developers because of the popularity of the Model-View-ViewModel (MVVM) pattern—called the Model-View-Presenter-ViewModel (MVPVM) pattern. This enterprise application design pattern was introduced in the Microsoft patterns & practices Prism project (Prism.CodePlex.com). (Note: This pattern was unnamed, so I refer to it as MVPVM.) Prism is best described with an excerpt from its Web site overview:

> *"Prism provides guidance designed to help you more easily design and build rich, flexible, and easy-to-maintain Windows Presentation Foundation (WPF) desktop applications, Silverlight Rich Internet Applications (RIAs), and Windows Phone 7 applications. Using design patterns that embody important architectural design principles, such as separation of concerns and loose coupling, Prism helps you to design and build applications using loosely coupled components that can evolve independently but that can be easily and seamlessly integrated into the overall application. These types of applications are known as composite applications."*

The popularity and success of MVVM overshadows MVPVM even though MVPVM is the evolution of MVVM (as this article will demonstrate). MVPVM provides all of the power and capability

This article discusses:

- The MVC design pattern
- The MVP design pattern
- The MVPVM design pattern
- The Business Logic Layer
- The Data Access Layer
- The benefit of experience and understanding history

Technologies discussed:

Windows Presentation Foundation, Model-View-Controller, Model-View-ViewModel, Model-View-Presenter, Prism

Code download available at:

amarillo.us.com/MSDN/MvpVmR2.zip

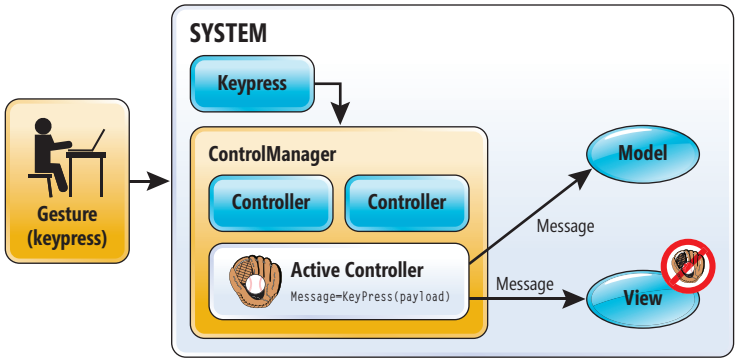


Figure 1 **Controller Receives Input and Sends Message (Method to Execute); Views Can't Catch User Input**

of MVVM while introducing the scalability and extensibility of the Model-View-Presenter (MVP) pattern. If your understanding is that MVVM has evolved *from* MVP for WPF, then you'll find this article enlightening.

Before we can fully appreciate the power and benefits of MVPVM, we have to understand how this pattern evolved—we have to understand our history. The Model-View-Controller (MVC) pattern is a crucial component to achieving this understanding, so I'll first introduce MVC, perhaps in a way you've never seen before.

The MVC Pattern

Please note that this MVC discussion is within the context of desktop applications; Web applications are another story and beyond the scope of this article.

In Martin Fowler's "GUI Architectures" document (bit.ly/110H7Y), he states the following about MVC: "Different people reading about MVC in different places take different ideas from it and describe these as 'MVC.' If this doesn't cause enough confusion, you then get the effect of misunderstandings of MVC that develop through a system of Chinese whispers." The "Whatsa Controller Anyway" document at bit.ly/7ajVeS sums up his point nicely stating: "Computer scientists in general have an annoying tendency to overload terms. That is to say, they tend to assign more than one meaning (sometimes contradictory) to the same word."

Smalltalk MVC gets further complicated by the fact that developers lack a common core of experience with their great architects in regard to the architect environment. As such, the overloading of terms and "Chinese whispers" are convoluted even more by the fact that most of us don't even know what a Controller is because we've never had to use one—the OS has always handled its functionality for us. With a few facts to provide the common core of experience we lack, you'll find MVC is actually easy to understand, and that the "C" in MVC has nothing in common with the "P" in MVP.

The Controller does have a concrete origin (which is well worth understanding but doesn't necessarily match modern approaches to MVC in the context of current technologies). The founding father of MVC, Trygve Reenskaug, wrote about that in 1979 in his document on "Models-Views-Controllers" (bit.ly/2cdqiu). This document begins to provide some insight on the purpose of the Controller. Reenskaug wrote:

"A controller is the link between a user and the system. It provides the user with input by arranging for relevant views to present themselves in appropriate places on the screen. It provides means for user output by presenting the user with menus or other means of giving commands and data. The controller receives such user output, translates it into the appropriate messages and passes these messages on to one or more of the views.

A controller should never supplement the views, it should for example never connect the views of nodes by drawing arrows between them.

Conversely, a view should never know about user input, such as mouse operations and keystrokes. It should always be possible to write a method in a controller that sends messages to views, which exactly reproduce any sequence of user commands."

This concept is illustrated in **Figure 1**.

> The popularity and success of MVVM overshadows MVPVM even though MVPVM is the evolution of MVVM.

A "message," in the context of MVC object-oriented programming (OOP), is a means of executing methods. They were described as "messages" because back then, virtual calls were a new concept and it was a way of distinguishing them from static function calls. In Chapter 1.2 of the book "Smalltalk, an Introduction to Application Development Using VisualWorks" (Prentice Hall, 1995), by Trevor Hopkins and Bernard Horan, the authors note "… if the receiving object understands the message it has been sent, then one of its internal operations (or methods) will be performed. This, in turn, may cause some computation to take place (by acting on one or more of the object's internal variables)." (Note: This "message-sending" OOP concept is available in Prism via its EventAggregator.)

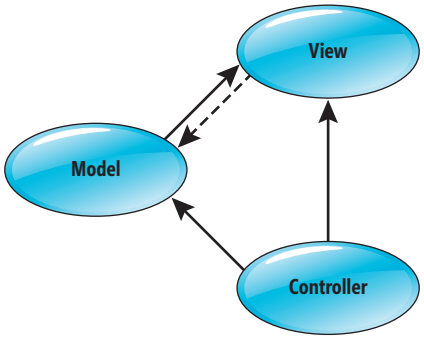


Figure 2 **Smalltalk Model-View-Controller**

The book nicely outlines the responsibilities of Smalltalk MVC in Chapter 29, "Introduction to Models, Views and Controllers," by teaching us that Controllers derive from the Controller class. It states, "Instances of this class have a reference to a sensor representing the mouse and keyboard, so that it can process input." It continues to tell us that the two distinct actions initiated from a Controller are communications with the Model (see **Figure 1**) and communications with the View *without* affecting the Model.

In Smalltalk MVC, "every" View will have a Controller and only one Controller at any given time can be active. In the original Smalltalk MVC, the UI was polled; the top-level ControlManager class asks each of the Controllers of the active View if it wants control. Only the View that contains the cursor can accept control. If a View is read-only, there's a NoController class available that's designed to refuse control. A View with subviews has the responsibility to poll the Controllers of its subviews. Once a Controller has accepted control, it will query the results of the keyboard or mouse and send messages to the View or Model as applicable, such as mouse move, button click and so on. In MVVM terms, this would be comparable to subscribing to a control's events in either the View's codebehind or via a ViewModel command. When a user interacts with the control, the applicable method will be called.

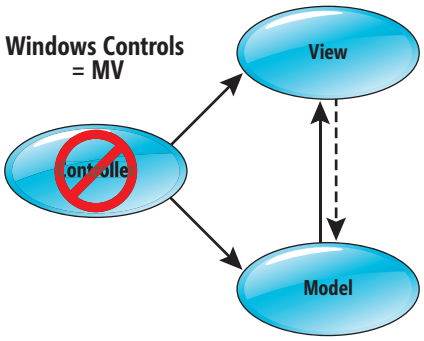


Figure 3 **Without the Controller, It Has Been Model-View (not Model-View-Controller)**

> Unlike Smalltalk MVC developers, WPF developers don't have to query keyboard buffers and package and raise events.

By this point, you can start to get a glimpse behind the purpose of the Controller within the context of MVC and an environment that doesn't automatically handle events for you. Unlike Smalltalk MVC developers, WPF developers don't have to query keyboard buffers and package and raise events. You merely subscribe to them and the applicable method "automagically" receives them via the Microsoft Event Pattern. With this knowledge, the following will perhaps have a different, and less confusing, meaning.

In Steve Burbeck's article, "Applications Programming in Smalltalk-80: How to Use Model-View-Controller (MVC)" (bit.ly/3ZfFCX), he states the following:

> *"The controller interprets the mouse and keyboard inputs from the user, commanding the model and/or the view to change as appropriate. Finally, the model manages the behavior and data of the application domain, responds to requests for information about its state (usually from the view), and responds to instructions to change state (usually from the controller)."*

See **Figure 2** for an illustration of this concept.

A final point to bring clarity to what could be a muddy topic is in the document, "Twisting the Triad, the Evolution of the Dolphin Smalltalk MVP Application Framework," by Andy Bower and Blair McGlashan at bit.ly/o8tDTQ (PDF download). This is one of the more comprehensive documents I've read on the topic. The authors note the following about the Controller: "The views, of course, are responsible for displaying the domain data while the controllers handle the raw user gestures that will eventually perform actions on this data."

I bring this final point up so that I have an opportunity to provide another definition. To fully comprehend the article (and others on MVC), you have to understand what "gestures" means (see **Figure 1**). In my research, I came across the article, "An Introduction to GUI Programming with UIL and Motif" (bit.ly/pkzmgc), which states the following:

> *"The essence of these steps is that a Motif program does nothing until the user requests some action. The actions are requested by selecting a menu item, by clicking a button or other window, or by pressing a key. These, collectively, are user gestures. Each gesture will trigger an application function to carry out some tasks."*

Andy Bower, coauthor of "Twisting the Triad," shared his thoughts on "gestures" with me:

> *"My take on 'gestures' is that they're a coalescence of one or more user events into something more meaningful.*
>
> *For example a TextController might absorb key down and key up events and convert them into individual 'keypress' gestures. Similarly, a SelectionInListController (the Controller attached to a list box) might absorb several mouse-down, mouse-tracking and mouse-up events within a list and treat them as a single list 'selection' gesture.*
>
> *Looked at like this, we see that a modern event-driven OS already does most of this gesture processing for us."*

To summarize Controller logic, you can see that the Controller function is rather consistent among the variations of MVC referenced. Because Microsoft controls (widgets) handle "the mouse and keyboard inputs from the user," you merely subscribe to the events and point to the method that needs to be executed—the "Controller" within your smart controls executes the method for you (handled at the OS level). As such, you don't have a need for a Controller, as clearly indicated by the "Twisting the Triad" article. Without the Controller, you're left with the Model-View pattern for Windows applications. You have to bear this in mind as you study MVC and its application and presentation models because without the Controller, there is no Triad (see **Figure 3**).

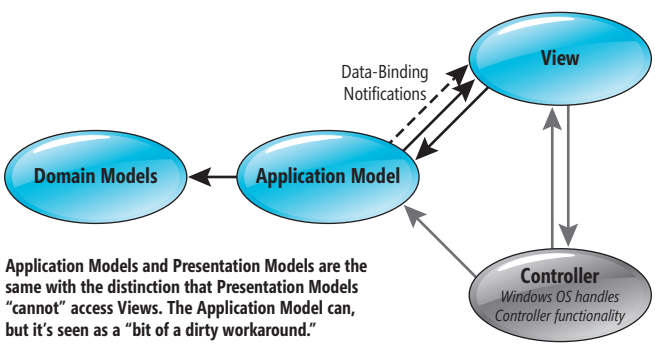


Figure 4 **VisualWorks Model-View-Controller**

Blazing-Fast **GRID CONTROLS**

Optimized for .NET

Award-Winning Presentation Controls and Reporting Libraries

Learn more and download your **FREE** evaluation copy today

Visit **DEVEXPRESS.COM/GRIDS**

***or Call Us* (818) 844-3383**

DevExpress™

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

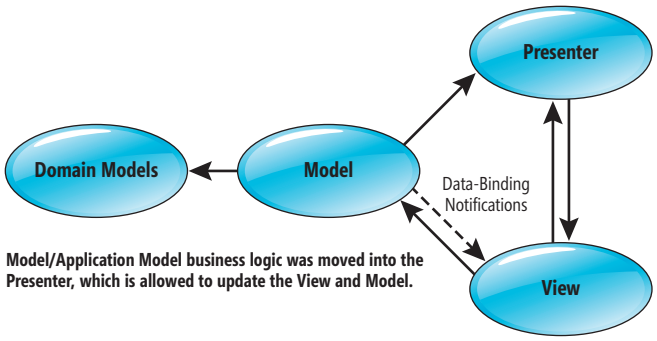


Figure 5 **Model-View-Presenter, Supervising Controller**

It's worth mentioning that it was not only the Controller that caused ambiguity with MVC. With Smalltalk MVC, the business/domain logic is in the Model, and with VisualWorks MVC the ApplicationModel contains the "application" logic required to present one or more business/domain objects to the user (see **Figure 4**). The "Twisting the Triad" article covers this in more detail. If you consider that the Application Model and Presentation Model have the same functionality, with the distinct difference being that the Presentation Model "cannot" access the View (Martin Fowler made a clear distinction so as not to overload terms), then WPF developers can quickly grasp *Presentation Model* because it's in essence MVVM. John Gossman, founding father of MVVM, explains this in his "PresentationModel and WPF" blog entry at bit.ly/pFs6cV. Like Martin Fowler, he was careful not to overload terms. This effectively gives us a clear understanding of what MVVM is; it's a "WPF-specific version of the PresentationModel pattern."

Once you can see MVC in its true light, it helps you understand the true roles of the Model and View. These are in essence the only two locations for business/domain logic. After reading "Twisting the Triad," you can see it was the responsibilities of the Application Model that were moved to the Presenter and that you can't simply replace the Controller with a Presenter—it wouldn't make sense, because they aren't synonymous.

The MVP Pattern

It's beyond the scope of this article to go into MVP with the attention it deserves; fortunately, there are numerous documents that address it, such as "Twisting the Triad" and the "Potel Paper," which you can download at bit.ly/dF4gNn (PDF).

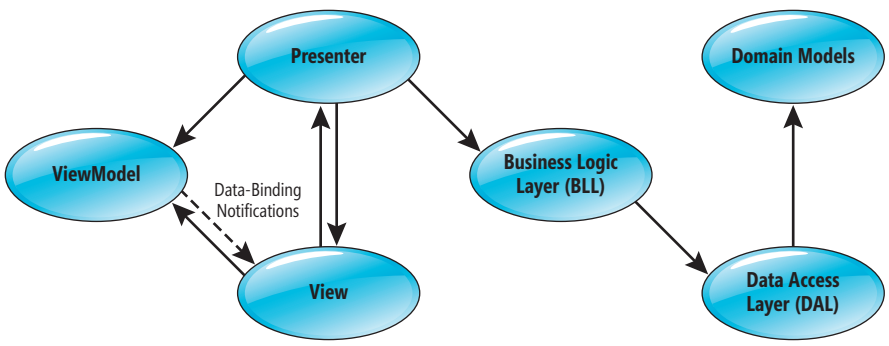


Figure 6 **Model-View-Presenter-ViewModel**

I will, however, note that the "Twisting the Triad" article states an important point that I alluded to, which led to the evolution of MVC to MVP:

> *"Another irritating feature of MVC, at least with respect to Dolphin, was that the idea of a controller did not fit neatly into the Windows environment. Microsoft Windows, like most modern graphical operating systems, provides a set of native widgets from which user interfaces can be constructed. These 'windows' already include most of the controller functionality embodied as part of the underlying operating system control."*

I'd also like to highlight from Martin Fowler's "GUI Architectures" article his statement that "This need to manipulate the widgets directly was seen by many as a bit of dirty workaround and helped develop the Model-View-Presenter approach." This is important to understand because MVVM has ingrained the Presentation Model mentality into many WPF developers; they believe it's "bad programming" to update the View directly. This is true for MVVM and the Presentation Model because once you reference the View, you can no longer reuse the ViewModel with a different View (the main reason for this rule). This limiting factor was one of the reasons Smalltalk MVC evolved to the MVP pattern. With MVPVM, developers can access the View and ViewModel directly (it's tightly coupled), which permits the View and ViewModel to remain completely decoupled (see **Figure 5**). Views can reuse different ViewModels and ViewModels can be reused by different Views (as you'll see later when I discuss the MVPVM pattern); this is one of the many great benefits of MVPVM.

Once you can see MVC in its true light, it helps you understand the true roles of the Model and View.

Andy Bower shared the following information on the topic with me to provide further clarification:

> *"It's perhaps worth pointing out that the aim of all of these patterns is reuse via plugability. Keeping the coupling loose wherever possible and the interfaces as small as possible gives maximum reuse in the way the components can be recombined.*
>
> *However, note that the Model has effectively two interfaces that must be compatible with any associated View and Presenter. The first is the standard function call interface, used for getting/setting values and performing commands. The second is the event interface, which must be the same as that expected by the View (observer pattern). This combination of interfaces can be referred to as a 'protocol.'*
>
> *For example, in Dolphin MVP, a list widget (triad) needs three components with compatible protocols. A ListModel, a ListView and a ListPresenter."*

Rock-Solid **REPORTING CONTROLS**
Optimized for.NET

Award-Winning Presentation Controls and Reporting Libraries

Learn more and download your **FREE** evaluation copy today
Visit **DEVEXPRESS.COM/REPORTING**
***or Call Us* (818) 844-3383**

DevExpress™

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

Once you understand *why* you don't want to update a View directly—and realize that this reason contributed to a new design pattern that would permit you to effectively reuse objects by eliminating the constraint of not being able to update the View—then you have the option to embrace the new opportunities that evolution offers. You live in an Agile world and—like your code—you have to change your thought patterns "lest history repeats itself."

> Lost in all of the voodoo and mystical powers of Prism, I recognized the MVP pattern, which helped me find some footing.

The MVPVM Pattern

I was taught this pattern early in Prism's lifecycle, specifically in Drop 7 of the CompositeWPF. Lost in all of the voodoo and mystical powers of Prism, I recognized the MVP pattern, which helped me find some footing. I quickly learned (as I teach new developers) to simply go to the Presenter and you'll have your starting point for figuring out the code and application. How code execution gets there can be learned later and doesn't have to interfere with critical timelines; the impact of the difficult learning curves can be managed.

The following definitions of the MVPVM components have excerpts from "Twisting the Triad" as applicable; I found this information to be informative, comprehensive and accurate, so in the interest of presenting to you the best information possible, I'll merely quote this article's definitions. These definitions hold true today for MVPVM (see **Figure 6**) as they did for MVP back in March 2000 when "Twisting the Triad" was written.

(Note: Prism is the context of this article, however, MVPVM will work without Prism; the various components will simply be tightly coupled to their implementation versus being resolved—loosely coupled by the Unity or Managed Extensibility Framework [MEF] dependency injection container.)

MVPVM: The Model

"This is the data upon which the user interface will operate. It is typically a domain object and the intention is that such objects should have no knowledge of the user interface." —"Twisting the Triad"

The separation of the Model from the ViewModel is required to address the concerns of dependency injection and its use within the Business Logic Layer (BLL) and Data Access Layer (DAL) to Create, Read, Update, Delete and List (CRUDL) persisted data. In MVPVM, only the DAL has access to the persisted Model objects.

MVPVM: The View

"The behavior of a view in MVP is much the same as in MVC. It is the view's responsibility to display the contents of a model. The model is expected to trigger appropriate change notifications whenever its data is modified and these allow the view to 'hang off' the model following the standard Observer pattern. In the same way as MVC does, this allows multiple views to be connected to a single model." —"Twisting the Triad"

(Note: I used the preceding quote to emphasize that MVP isn't a new pattern and that it's as valid today as it was when MVP evolved from MVC. However, the quote does reference the "Model," whereas MVPVM uses the "ViewModel.")

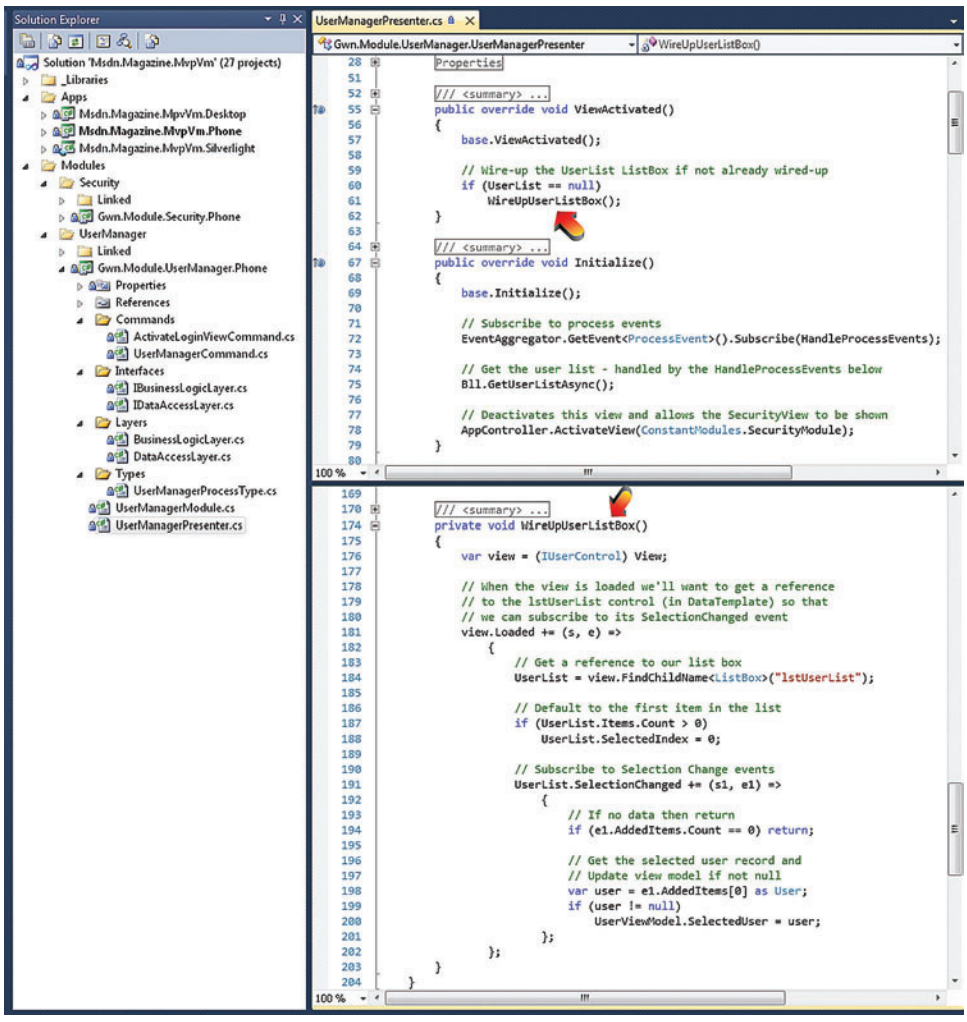


Figure 7 **Benefits of Being Able to Access the View Directly**

Powerhouse **ANALYTICS**

Optimized for.NET

Award-Winning Presentation Controls and Reporting Libraries

Learn more and download your **FREE** evaluation copy today

Visit **DEVEXPRESS.COM/ANALYTICS**

***or Call Us* (818) 844-3383**

DevExpress™

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

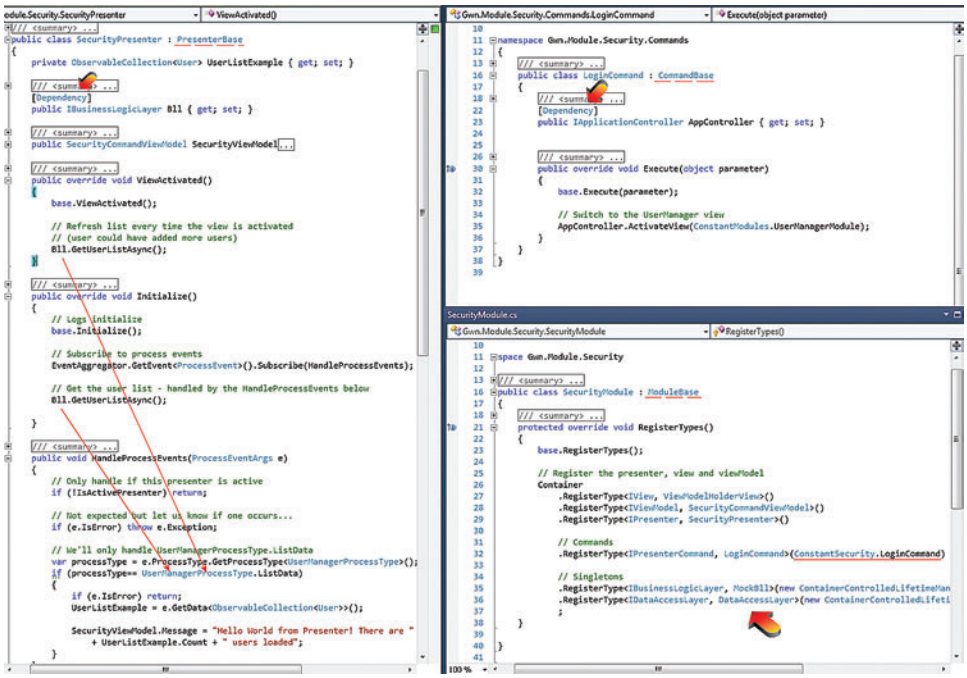


Figure 8 **Security Module Code**

With MVPVM, there's never a need to have code in codebehind. The Presenter has access to the View and can subscribe to events, manipulate controls and manipulate the UI as required. This proved beneficial while developing the multi-targeted application that accompanies this article. The User ListBox wouldn't respect the data binding for the selection changed event, so I had to develop a way that would work for all three platforms (because they share the same code). When the View is activated, I call the WireUpUserListBox (see **Figure 7**, line 174) and use the View to obtain a reference to my lstUserList control (which resides in a DataTemplate). Once it's found, I wire up to the SelectionChanged event and update the ViewModel SelectedUser property. This works on the desktop, Silverlight and Windows Phone.

A View has no knowledge of the ViewModel, so they aren't tightly coupled. As long as a ViewModel supports all of the properties to which a View binds, it can easily use the View. The Presenter is responsible for wiring up Views to their ViewModels by setting their DataContext to the applicable ViewModel.

MVPVM: The Presenter

"While it is the view's responsibility to display model data, it is the presenter that governs how the model can be manipulated and changed by the user interface. This is where the heart of an application's behavior resides. In many ways, an MVP presenter is equivalent to the application model in MVC; most of the code dealing with how a user interface works is built into a presenter class. The main difference is that a presenter is directly linked to its associated view so that the two can closely collaborate in their roles of supplying the user interface for a particular model." —"Twisting the Triad"

The Presenter will have dependencies on the interfaces for the BLLs from which it needs to retrieve domain objects (data), as shown in the left pane of **Figure 8**. It will use the resolved instances as configured in the module (see **Figure 8**, bottom right pane) or bootstrapper to access the data and populate the ViewModel.

Typically, only the Presenter will be tightly coupled in MVPVM components. It will be tightly coupled to the View, ViewModel and the BLL interfaces. Presenters aren't intended to be reused; they address specific concerns as well as the business logic/rules for those concerns. In cases where a Presenter can be reused across enterprise applications, it's likely a module would be better suited for the task—that is, you could create a login module (project) that could be reused by all of your enterprise applications. If you code against an interface, the module can be easily reused using dependency injection technologies such as the MEF or Unity.

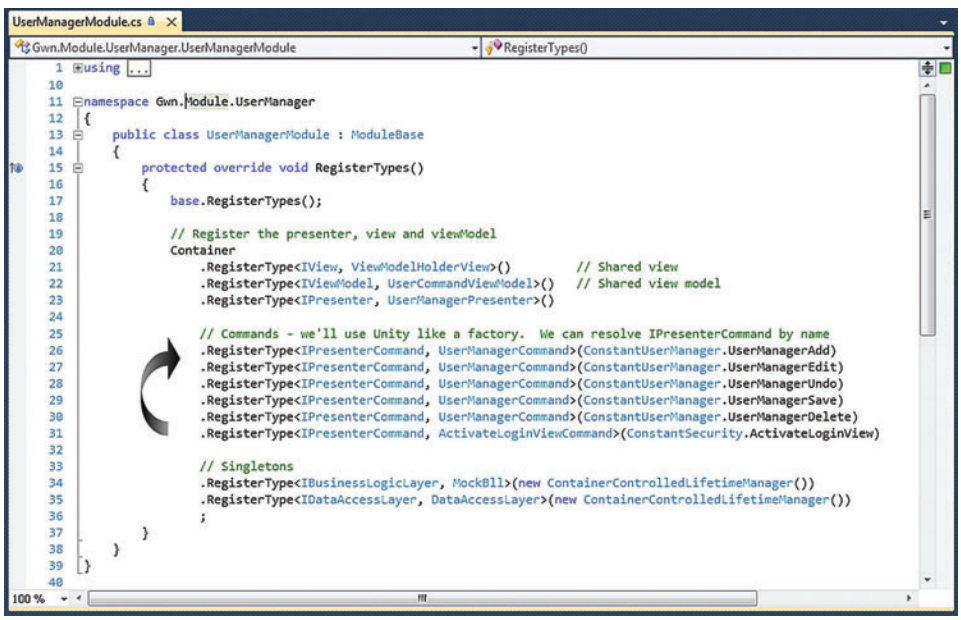


Figure 9 **Registering BLL, DAL and Commands**

MVPVM: The ViewModel

"In MVP, the model is purely a domain object and there is no expectation of (or link to) the user interface at all." —"Twisting the Triad"

Elegant **CHARTING CONTROLS**

Optimized for .NET

Award-Winning Presentation Controls and Reporting Libraries

Learn more and download your **FREE** evaluation copy today

Visit **DEVEXPRESS.COM/CHARTING**

***or Call Us* (818) 844-3383**

DevExpress™

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

This prevents the ViewModel from being tightly coupled to a single View, permitting it to be reused by numerous Views. Likewise, the ViewModel will have no application business logic, so ViewModels can be easily shared across enterprise applications. This promotes reuse and application integration. For example, look at **Figure 8**, top right pane, line 28. This SecurityCommandViewModel resides in the Gwn.Library.MvpVm.xxxx project (where xxxx = Silverlight, desktop or Windows Phone). Because a User ViewModel will be required in most applications, this is a reusable component. I have to be careful not to pollute it with business logic specific to the demo application. This isn't a problem with MVPVM, because the business logic will be handled by the Presenter, not within the ViewModel (see **Figure 6**).

> With MVPVM, only the DAL holds specific information required to retrieve data.

(Note: As the ViewModel properties are populated by the Presenter, they'll raise NotifyPropertyChanged events, which will alert the View that it has new data—it's the View's responsibility to update itself to the ViewModel data upon notification.)

The Business Logic Layer

BLLs have no knowledge of the persisted Model data. They work strictly with domain objects and interfaces. Typically, you'll use dependency injection to resolve your DAL interface within the BLL, so you can later swap out the DAL without affecting any downstream code. Notice in **Figure 9**, line 34, that I'm using a MockBll implementation for IBusinessLogicLayer for my demo application. Later I can easily replace it with a production implementation of the interface with one line of code because I've developed against an interface.

Business logic isn't limited to the BLLs. In **Figure 9**, I register named types (for IPresenterCommand) so that I can use dependency injection as a factory. When the user clicks on a button (lines 29–33 in **Figure 10**), the command parameter is resolved (instantiated) by the baseclass and the applicable command is executed. For example, the LoginCommand (**Figure 8**, top right pane) is a command that will activate the UserManagerView. All that was required to wire this up was a Button command in XAML and an entry in the SecurityModule (see **Figure 8**, bottom right pane, line 32).

The Data Access Layer

Persisted domain data could be stored in SQL databases or XML or text files, or retrieved from a REST service. With MVPVM, *only* the DAL holds specific information required to retrieve data. The DAL will only return domain objects to the BLL. This averts the need for the BLL to have knowledge of connection strings, file handles, REST services and so on. This enhances scalability and extensibility; you can easily switch your DAL from an XML file to a cloud service without affecting any existing code outside of the DAL and the application configuration file. As long as your new DAL unit tests work for the CRUDL processes, you can configure the application to use the new DAL without impacting current applications and their usage.

Understanding History, Benefitting from Experience

Thanks to the patterns & practices team, I've been able to achieve success for my clients using bleeding-edge technologies and patterns such as MVPVM, without failure. I found that by keeping up with the team's pace, technologies and patterns, that as newer technologies and patterns emerge, I can pull from past experience to help me more easily find my way in new territory.

MVP is a constant and consistent pattern that I've seen used since the earliest days of my architectural studies using patterns & practices projects. To fully appreciate MVP, you have to understand your history, particularly the components of MVC. This isn't easy, because of the numerous factors discussed.

But by understanding our past, and the reason for our evolution, we can more quickly grasp the need for and power of newer patterns. Even with Prism and all of its complexity and magic, you'll be able to bypass a difficult learning curve by simply knowing this history and appreciating the need for the newer patterns. In all of the successful projects I've been involved with since the emergence of this pattern in early drops of Prism, MVPVM has revealed no issues, speed bumps or brick walls (through evolution they appear to have been eliminated), permitting developers to increase velocity as we build applications that are scalable, extensible and stable. ■

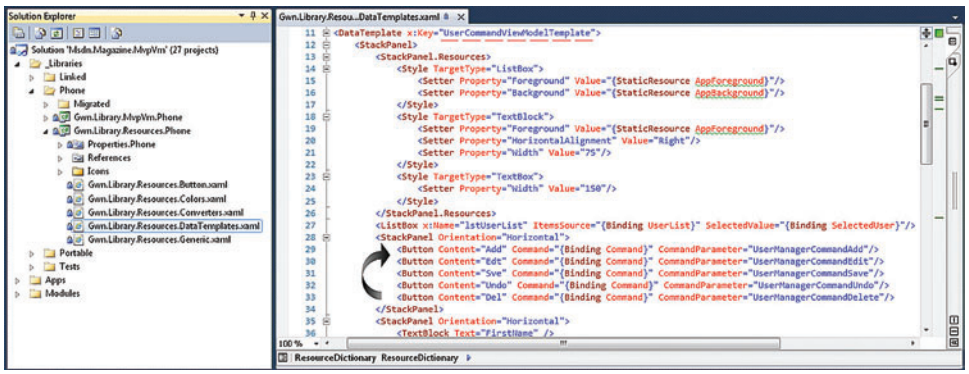


Figure 10 **DataTemplate for the UserCommandViewModel**

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

Thanks to the following technical experts for reviewing this article: *Andy Bower, Christina Helton and Steve Sanderson*

5 YEARS OF EXCELLENCE

XCEED
DataGrid
for WPF

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

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

Microsoft®
Visual Studio® Team System 2010

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

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

XCEED Professional Themes for WPF

Theme your entire app in minutes. Flawless styles for all official WPF controls.

XCEED DataGrid for Silverlight

Incredible streaming technology. Speed up your app and say goodbye to paging.

XCEED Ultimate ListBox for WPF

NEW

The world's first streaming listbox. Simple, drop-in upgrade to the WPF listbox.

XCEED Ultimate ListBox for Silverlight

NEW

Fast and fluid, with ground-breaking streaming technology.

Visual Studio PARTNER

XCEED MULTI-TALENTED COMPONENTS

Integrating Geolocation into Web Applications

Brandon Satrom

If you own a smartphone, you've probably used an application or built-in functionality to get directions from where you are to where you want to go, or to find a nearby restaurant, hotel or gas station in some unfamiliar town. The technical term for this facility is *geolocation*, and it describes a capability whereby a device can use external location data (global positioning system [GPS], cell tower triangulation and Wi-Fi data) to find out where you are and provide that information to the application or service requesting this data. Your location is typically expressed using data such as latitude and longitude, heading, speed, direction, and the like. With that information, a device can provide features such as turn-by-turn navigation or the locations of nearby taco trucks that happen to be open at 2 a.m.

This article discusses:

- The Geolocation API
- How geolocation works
- Detecting and polyfilling geolocation support

Technologies discussed:

HTML5, Geolocation API, Internet Explorer 9, WebMatrix, Bing Maps AJAX Control, Modernizr

Code download available at:

code.msdn.microsoft.com/mag201112HTML5

Geolocation is something we take for granted on our mobile devices, but its use is spreading quickly beyond native mobile applications. More and more laptops and mobile PCs are now fitted with GPS chips, and the explosion of the Web as a viable platform for mobile application development has stirred demand for geolocation services right within the browser.

Luckily for us, the W3C picked up on this trend early and created a specification that describes a standard way to use geolocation services from within Web applications. The spec, which you can find at bit.ly/3r737c, isn't "officially" part of HTML5, but because I'm using that term in this series to describe a set of open Web technologies—as the W3C does—I couldn't miss the opportunity to give you a brief tour of this exciting technology.

This month's article will give you an overview of the Geolocation API and show you how you can get started using it, as well as how to handle the most common types of errors that can occur during a geolocation request. I'll discuss how geolocation is implemented by desktop and mobile browsers, and wrap up with a brief look at how you can use a polyfilling library to provide basic geolocation support to users with older browsers.

Getting Started with Geolocation

According to the W3C specification, geolocation is an "… API that provides scripted access to geographical location information associated with the hosting device." Because geolocation is a built-in API, all you need to get started is a browser that supports

Figure 1 **Geolocation Options**

Option	Description	Values
maximumAge	Length of time (in milliseconds) to cache the reported location.	0 (default) to Infinity
Timeout	Length of time (in milliseconds) to wait for the location service to respond.	0 to Infinity (default)
enableHighAccuracy	Whether the browser should attempt to use high-accuracy location services (for example, GPS), where supported by the browser and device.	true, false (default)

it. At the time of this writing, Internet Explorer 9 (including Internet Explorer 9 on Windows Phone 7.5, code-named "Mango"),as well as the current versions of Chrome, Firefox, Safari and Opera, all support geolocation, so the specification is not only broadly supported, but probably available to a large segment of your user base.

The Geolocation API lives in the global "navigator" object in JavaScript (window.navigator.geolocation). It provides two ways to work with location services: one-shot position requests and repeated position updates. For one-shot requests, use the getCurrentPosition method, like so:

```
navigator.geolocation.getCurrentPosition(function(position) {
       var coordinates = position.coords;
       console.log("Your latitude: " + coordinates.latitude + " and longitude:
         " + coordinates.longitude + " (Accuracy of:
         " + coordinates.accuracy + " meters)");
}, errorHandler, { maximumAge: 100, timeout: 6000, enableHighAccuracy: true});
```

The method getCurrentPosition accepts three parameters. The first is a callback function that fires when the browser successfully determines the current location of a device. The second is a callback function that fires if an error occurs (more on that in a moment). The third is an object literal for specifying geolocation options. The available options and their defaults are listed in **Figure 1**.

The success handler provides a position object that holds important properties related to the user's current location. The Geolocation spec defines several properties, but only three are implemented consistently across all browsers: latitude, longitude and accuracy. Other properties such as heading, direction and altitude are available for browsers to implement, and could be quite useful for mobile browsing scenarios in the future.

For repeated position requests, the geolocation object offers two public methods, watchPosition and clearWatch; watchPosition can be used like this:

```
var watchId = navigator.geolocation.watchPosition(function(position) {
       var coordinates = position.coords;
       console.log("Your latitude: " + coordinates.latitude + " and longitude:
         " + coordinates.longitude + " (Accuracy of:
         " + coordinates.accuracy + " meters)");
}, errorHandler, {maximumAge: 100, timeout: 6000, enableHighAccuracy: true});
```

The watchPostion method takes the same parameters as getCurrentPosition, with a success handler, error handler and options. The key difference is that watchPosition immediately returns an ID that can be used to identify the function in question. Because watchPosition is a repetitive function, the browser will call it each time it detects a change in the user's location until the window is closed or the script explicitly cancels the watch. To cancel watchPosition, you simply call clearWatch with the ID returned by watchPosition, like so:

```
clearWatch(watchId);
```

Let's look at using getCurrentPosition in a sample application. For this example, I'll continue using the WebMatrix (http://aka.ms/webm) Bakery sample I used in my last article. The site is a place where users can order baked goods and have them delivered, but I want to extend this functionality to allow the user to specify a nearby brick-and-mortar location where they'll pick up the order. What Best Buy has done for in-store pickup of electronics, I want to do for the bustling baked goods industry.

I started by using the modified Bakery sample application I decked out with HTML5 Forms features in my November article (msdn.microsoft.com/magazine/hh547102), and then added a feature that allows the user to select a nearby bakery for pickup by clicking a link on the form. I'm using the Bing Maps AJAX control (which you can download at msdn.microsoft.com/library/gg427610), and after adding the proper references and creating an HTML element to contain my map, I added a few divs to the page to represent the Pickup and Shipping sections of the form. Each div contains a link that allows

Figure 2 **locateBakery.js**

```
var mapDiv = document.getElementById("map");
var _map = new Microsoft.Maps.Map(mapDiv, { credentials: myCredentials });

function hideMap() {
    $('#pickup').hide();
    $('#map').hide();
    $('#ship').show();
}

function displayError(msg) {
    $('#error').val(msg);
}

function placeLocationOnMap(latitude, longitude) {
    var location = new Microsoft.Maps.Location(latitude, longitude);
    var bakeries = lookupNearbyBakeries(latitude, longitude);
    _map.setView({ zoom: 12, center: location });

    // Add a pushpin to the map representing the current location
    var pin = new Microsoft.Maps.Pushpin(location);
    _map.entities.push(pin);

    for (var i=0;i<bakeries.length;i++) {
        _map.entities.push(bakeries[i]);
    }
}

function errorHandler(e) {
    if (e === 1) { // PERMISSION_DENIED
        hideMap();
    } else if (e === 2) { //POSITION_UNAVAILABLE
        displayError('Cannot find your location. Make sure your network
        connection is active and click the link to try again.');
        hideMap();
    } else if (e === 3) { //TIMEOUT
        displayError('Cannot find your location. Click the link to try again.');
        hideMap();
    }
}

function locate() {
    navigator.geolocation.getCurrentPosition(function (position) {
        var coordinates = position.coords;
        placeLocationOnMap(coordinates.latitude, coordinates.longitude);
    }, errorHandler);
}
```

Figure 3 **Using Geolocation with the Bakery Application**

the user to toggle between the two options. When the "Interested in picking your order up at one of our stores?" link is clicked, I execute a function in a JavaScript file called locateBakery.js. The key methods of that function are shown in **Figure 2**.

The locate function calls navigator.geolocation.getCurrentPosition and if that call succeeds, provides the coordinates to the placeLocationOnMap function, which places the user's location on a map along with a list of nearby bakeries (the bakery lookup logic was omitted for space). This allows the user to find a nearby bakery from which to pick up her order. The result is depicted in **Figure 3**.

Of course, geolocation is a network-dependent service, so what do you do when geolocation fails? Perhaps you noticed that for both the getCurrentPosition and watchPosition functions I specified a second parameter called errorHandler. According to the Geolocation specification, calls to getCurrentPosition and watchPosition should call both a success handler and an error handler. When calling the geolocation service, there are three possible errors:

- The user could reject the browser's request to track location information
- The location request could timeout
- The location request could fail before the timeout expires

When adding geolocation support to your application, you should be sure to handle all three cases. An example of this can be found in the source in **Figure 2**, where I display an error message and redisplay the shipping information section when the geolocation call fails for any reason.

How Geolocation Works

Even though the Geolocation specification is quite prescriptive in defining the public API that each browser must provide to developers, it has nothing to say regarding exactly how geolocation should be implemented in a browser. The general guidance is that each browser should attempt to balance accuracy with power-consumption considerations, with the goal of providing applications with a reasonably accurate set of coordinates.

In a desktop or laptop scenario, most browsers use the same mechanism to provide geolocation data, though the background services they depend on differ. In the case of Internet Explorer 9 and later, the browser will collect your IP address or a list of nearby Wi-Fi hotspots and then pass that information to the Microsoft Location Service (the same service built-in location facilities on Windows Phone devices use) for a set of coordinates. Google Chrome and the Mozilla Firefox browser both also use IP and Wi-Fi hotspot data, but rely on the Google Location Service for the actual request. The location service method for finding a user's location sacrifices some accuracy for speed (and low power consumption), and is quite useful as a backup, or when the device lacks a built-in GPS chip.

In a desktop or laptop scenario, most browsers use the same mechanism to provide geolocation data.

For mobile browsers, the situation is a bit more complex, and again is up to the browser in question. Because most modern smartphones have GPS chips, the browser is free to use the GPS location services provided by the mobile OS in an attempt to obtain more accurate coordinates. In the case of Internet Explorer 9 on Windows Phone 7.5 and Safari on iOS, that is exactly what those browsers do. The result is a more accurate location at the expense of additional power consumption, which is usually an acceptable trade-off in most mobile geolocation scenarios, such as turn-by-turn navigation. It's important to note, however, that a mobile device browser isn't guaranteed to use GPS-based geolocation simply because a chip is available. Because the specification leaves implementation up to the browser, the browser or underlying mobile OS service may choose to fall back to cell tower triangulation or IP lookup, or use a combination of cell triangulation and GPS. (Many mobile OSes already do this for native applications as a part of their built-in location services.)

Our Name Says It All

activePDF®

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

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

activePDF®
Leading the iPaper Revolution

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

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

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

For a Web application developer, this is great news, because it means that with geolocation, you can build cross-platform mobile applications that can rely on location facilities native to a device, just as native-application developers do.

Detecting and Polyfilling Geolocation Support

When adding geolocation support to your application, you need to consider what your approach will be for those users visiting your site with an unsupported browser. As I've discussed in the series already, the way you determine if geolocation is supported in a given browser is by performing manual feature detection ("if (!!navigator.geolocation)") or using a library like Modernizr (modernizr.com). For users without geolocation support, there are two courses of action: gracefully degrade the application or polyfill support via an external library. For the first option, you can degrade your application by showing a form that lets users give you their location, which you then can use (via a service call) to determine the latitude and longitude of that user. I covered this scenario in my September article of this series, so I won't repeat it here. You can view both that scenario and the online code sample at (msdn.microsoft.com/magazine/hh394148).

A polyfill is a library that provides a script-based implementation for an HTML5 feature.

I'll cover the second option in a bit more depth this time. As I've mentioned in previous articles, when you find yourself in the market for a cross-browser polyfill for an HTML5 technology, the first place to go is Modernizr's home on GitHub, where they've posted a pretty comprehensive list of usable polyfills (bit.ly/eBMoLW). At the time of this writing, there were three geolocation polyfills listed. I chose to use the one provided by Paul Irish (bit.ly/q20T0I) purely as an example for this article.

You'll recall from my September article that a polyfill is a library that provides a script-based implementation for an HTML5 feature, and it often adheres to the documented API for that specification in the process. The latter point is key, because it means that once I've added such a library to my application, I can continue to interact with the capabilities of that feature as if it were fully supported in the browser, and I don't have to fork the flow of my program with conditional logic to support an older browser.

Figure 4 **The getCurrentPosition Method Implemented via a Shim**

```
geolocation.getCurrentPosition = function(callback){
  if (cache) callback(cache);

  $.getScript('//www.google.com/jsapi',function(){
    cache = {
      coords : {
        "latitude": google.loader.ClientLocation.latitude,
        "longitude": google.loader.ClientLocation.longitude
      }
    };

    callback(cache);
  });
};
```

As I've done before in this series, I'm using Modernizr to help me detect support for the geolocation feature and, if no such support exists, asynchronously load the geolocationShim.js script file:

```
Modernizr.load({
  test: Modernizr.geolocation,
  nope: '../js/geolocationShim.js',
  complete: function() {
  locate();
  }
});
```

If geolocation is supported, execution will continue to the locate function. If not, the Geolocation polyfill will be loaded and, when complete, execution will then continue to the locate function. When I run the sample with this logic in Internet Explorer 9, everything works as it should and the native geolocation functionality is called.

To test what this looks like in an older browser, I can open up the F12 Developer Tools for Internet Explorer 9 and switch the Browser Mode to Internet Explorer 8. When I do this and execute the page again, Modernizr will detect that geolocation is not supported and load my shim. Then, when the getCurrentPosition method in my shim is called, the code in **Figure 4** will be executed.

Because this polyfilling library adheres to the Geolocation API specification and provides the getCurrentPosition and watchPosition methods, I'm not required to implement any additional checks or conditional logic in my main getLocation function.

Whether you choose to gracefully degrade your application and allow users to manually provide their location or try out implementing a shim, you have options when it comes to adopting geolocation in your applications, while making sure that your site continues to provide a great experience to those who don't yet use a modern browser.

Much of what the world is calling HTML5, or the Open Web Technologies, is a set of technologies geared toward making real application development possible on the Web. The gap between the Web and the desktop is narrowing, and geolocation is a great example of what's possible with Web applications today. In this article, I covered the basics of the Geolocation spec. I showed how you can get started, how geolocation is implemented across desktop and mobile browsers, and how to polyfill geolocation support in older browsers. Now it's time for you to add this exciting feature to your Web applications, so go make some awesome apps today!

If you're looking for more information on geolocation support in Internet Explorer 9, check out the Internet Explorer 9 Guide for Developers at msdn.microsoft.com/ie/ff468705. For other cross-browser polyfills for geolocation, check out the complete list at bit.ly/qNE92g.

Finally, all of the demos for this article—which are available online—were built using WebMatrix, a free, lightweight Web development tool from Microsoft. You can try WebMatrix out for yourself at aka.ms/webm. ■

Brandon Satrom *works as a developer evangelist for Microsoft outside of Austin. He blogs at userinexperience.com and can be found on Twitter at twitter.com/BrandonSatrom.*

Thanks to the following technical experts for reviewing this article: *Jon Box, John Hrvatin, Clark Sell and Andy Zeigler*

Janus WinForms Controls Suite V4.0 | from $853.44

Janus systems

Add powerful Outlook style interfaces to your .NET applications.

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

TX Text Control .NET for Windows Forms/WPF | from $499.59

TX TEXT CONTROL word processing components

Word processing components for Visual Studio .NET.

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

FusionCharts | from $195.02

InfoSoft Global
empowering human thoughts

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

- Liven up your web applications using animated & data-driven charts
- Create AJAX-enabled charts with drill-down capabilities in minutes
- Export charts as images/PDF and data as CSV from charts itself
- Create gauges, dashboards, financial charts and over 550 maps
- Trusted by over 19,000 customers and 400,000 users in 110 countries

Spread for .NET Professional | from $1,469.02

GrapeCity PowerTools

Includes MultiRow, Stand-Alone Chart, Formula Provider and Runtime Spread Designer.

- Programmable embedded spreadsheet platform with royalty-free licensing
- Full import/export support for Excel documents
- Built-in support for Excel functions
- Flexible printing and PDF export options
- Built-in chart with hundreds of chart styles

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

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

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

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

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

GSA Schedule Contract GS-35F-0188R

Writing a Plug-in for Sysinternals ProcDump v4.0

Andrew Richards

You've been up all night installing the latest upgrade of your mission-critical application and everything has gone perfectly. And then it happens—the application hangs, just as everyone starts to arrive at work. At times like this, you need to cut your losses, accept that the release is a failure, gather relevant evidence as fast as possible, and then start that ever-important rollback plan.

Capturing a memory dump of an application at times like this is a common troubleshooting tactic, whether for hang, crash or performance reasons. Most dump capture tools take an all-or-nothing approach: they give you everything (full dumps) or very little (mini dumps). The mini dumps are usually so small that fruitful debug analysis isn't possible because the heaps are missing. Full dumps have always been preferred, but they're rarely an option anymore. Ever-increasing memory means that a full dump can take 15, 30 or even 60 minutes. Moreover, the dump files are becoming so big that they can't easily be moved around for analysis, even when compressed.

Last year, Sysinternals ProcDump v3.0 introduced the MiniPlus switch (-mp) to address the size issue for native applications. This creates a dump that's somewhere between a mini dump and a full dump in size. The MiniPlus switch's memory inclusion decisions are based on a multitude of heuristic algorithms that consider memory type, memory protection, allocation size, region size and stack contents. Depending on the layout of the target application, the dump file can be 50 percent to 95 percent smaller than a full dump. More important, the dump is as functional as a full dump for most analysis tasks. When the MiniPlus switch is applied to the Microsoft Exchange 2010 Information Store running with 48GB of memory, the result is a 1GB–2GB dump file (a 95 percent reduction).

Mark Russinovich and I have been working on a new release of ProcDump that now lets you make the memory inclusion decisions. Sysinternals ProcDump v4.0 exposes the same API that MiniPlus uses internally as an external DLL-based plug-in via the -d switch.

In this article, I'm going to dissect how Sysinternals ProcDump v4.0 works by building a series of sample applications that expand

This article discusses:

- Terms associated with dump collection
- The MiniDumpWriteDump function
- Exception context records
- Runtime and post mortem exceptions
- The MiniDumpCallback function
- Using Sysinternals VMMap to look at an application's memory

Technologies discussed:

Sysinternals ProcDump v4.0, Sysinternals VMMap

Code download available at:

code.msdn.microsoft.com/mag201112ProcDump

Figure 1 **MiniDump01.cpp**

```
// MiniDump01.cpp : Capture a hang dump.
//
#include "stdafx.h"
#include <windows.h>
#include <dbghelp.h>

int WriteDump(HANDLE hProcess, DWORD dwProcessId, HANDLE hFile, MINIDUMP_
TYPE miniDumpType);

int _tmain(int argc, TCHAR* argv[])
{
  int nResult = -1;
  HANDLE hProcess = INVALID_HANDLE_VALUE;
  DWORD dwProcessId = 0;
  HANDLE hFile = INVALID_HANDLE_VALUE;
  MINIDUMP_TYPE miniDumpType;

  // DbgHelp v5.2
  miniDumpType = (MINIDUMP_TYPE) (MiniDumpNormal | MiniDumpWithProcessThreadData |
    MiniDumpWithDataSegs | MiniDumpWithHandleData);
  // DbgHelp v6.3 - Passing unsupported flags to a lower version of DbgHelp
     does not cause any issues
  miniDumpType = (MINIDUMP_TYPE) (miniDumpType | MiniDumpWithFullMemoryInfo |
    MiniDumpWithThreadInfo);

  if ((argc == 2) && (_stscanf_s(argv[1], _T("%ld"), &dwProcessId) == 1))
  {
    // Generate the filename (ISO format)
    SYSTEMTIME systemTime;
    GetLocalTime(&systemTime);
    TCHAR szFilename[64];
    _stprintf_s(szFilename, 64, _T("c:\\dumps\\minidump_%04d-%02d-
      %02d_%02d-%02d-%02d-%03d.dmp"),
        systemTime.wYear, systemTime.wMonth, systemTime.wDay,
        systemTime.wHour, systemTime.wMinute, systemTime.wSecond,
        systemTime.wMilliseconds);

    // Create the folder and file
    CreateDirectory(_T("c:\\dumps"), NULL);
    if ((hFile = CreateFile(szFilename, GENERIC_WRITE, 0, NULL, CREATE_ALWAYS,
      FILE_ATTRIBUTE_NORMAL, NULL)) == INVALID_HANDLE_VALUE)
    {
      _tprintf(_T("Unable to open '%s' for write (Error: %08x)\n"), szFilename,
        GetLastError());
      nResult = 2;
    }
    // Open the process
    else if ((hProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE,
dwProcessId)) == NULL)
    {
      _tprintf(_T("Unable to open process %ld (Error: %08x)\n"), dwProcessId,
        GetLastError());
      nResult = 3;
    }
    // Take a hang dump
    else
    {
      nResult = WriteDump(hProcess, dwProcessId, hFile, miniDumpType);
    }
    if (hFile) CloseHandle(hFile);
    if (hProcess) CloseHandle(hProcess);
    if (nResult == 0)
    {
      _tprintf(_T("Dump Created - '%s'\n"), szFilename);
    }
    else
    {
      DeleteFile(szFilename);
    }
  }
  else
  {
    _tprintf(_T("Usage: %s <pid>\n"), argv[0]);
    nResult = 1;
  }

  return 0;
}

int WriteDump(HANDLE hProcess, DWORD dwProcessId, HANDLE hFile, MINIDUMP_
TYPE miniDumpType)
{
  if (!MiniDumpWriteDump(hProcess, dwProcessId, hFile, miniDumpType,
NULL, NULL, NULL))
  {
    _tprintf(_T("Failed to create hang dump (Error: %08x)\n"), GetLastError());
    return 11;
  }
  return 0;
}
```

on each other, implementing more and more of the ProcDump functionality. By delving into how ProcDump works under the covers, I'll show you how you can write a plug-in that interacts with ProcDump and the underlying DbgHelp API.

The code download contains the sample applications and also a collection of applications that crash in various ways (so you can test your code). The MiniDump05 sample has all of the APIs implemented as a standalone application. The MiniDump06 sample implements the MiniDump05 sample as a plug-in for Sysinternals ProcDump v4.0.

Terminology

It's easy to get all the terms associated with dump collection confused—the term "Mini" is used a lot. There is the MiniDump *file format*, Mini and MiniPlus *dump contents*, and the MiniDumpWriteDump and MiniDumpCallback *functions*.

Windows supports the MiniDump file format via DbgHelp.dll. A MiniDump dump file (*.dmp) is a container that supports partial or complete capture of the memory in a user mode or kernel mode target. The file format supports the use of "streams" to store additional metadata (comments, process statistics and so forth). The file format's name is derived from the requirement to support the capture of a minimal amount of data. The DbgHelp API MiniDumpWriteDump and MiniDumpCallback functions are prefixed with MiniDump to match the file format they produce.

Mini, MiniPlus and Full are used to describe the different amounts of content in the dump files. Mini is the smallest (*mini*mal) and includes the process environment block (PEB), thread environment blocks (TEBs), partial stacks, loaded modules and data segments. Mark and I coined MiniPlus to describe the contents of a Sysinternals ProcDump -mp capture; it includes the contents of a Mini dump, *plus* memory heuristically deemed important. And a Full dump (procdump.exe -ma) includes the entire virtual address space of the process, regardless of whether the memory is paged in to RAM.

MiniDumpWriteDump Function

To capture a process in MiniDump file format to a file, you call the DbgHelp MiniDumpWriteDump function. The function requires a handle to the target process (with PROCESS_QUERY_INFORMATION and PROCESS_VM_READ access), the PID of the target process, a handle to a file (with FILE_GENERIC_WRITE access), a bitmask of MINIDUMP_TYPE flags, and three optional parameters: an Exception Information structure (used to include an exception context record); a User Stream Information structure (commonly used to include a comment in the dump via the

Figure 2 **Debugger Commands**

MINIDUMP_TYPE	Debugger Commands
MiniDumpNormal	knL99
MiniDumpWithProcessThreadData	!peb, !teb
MiniDumpWithDataSegs	lm, dt <global>
MiniDumpWithHandleData	!handle, !cs
MiniDumpWithFullMemoryInfo	!address
MiniDumpWithThreadInfo	!runaway

CommentStreamA/W MINIDUMP_STREAM_TYPE types); and a Callback Information structure (used to modify what's captured during the call):

```
BOOL WINAPI MiniDumpWriteDump(
  __in  HANDLE hProcess,
  __in  DWORD ProcessId,
  __in  HANDLE hFile,
  __in  MINIDUMP_TYPE DumpType,
  __in  PMINIDUMP_EXCEPTION_INFORMATION ExceptionParam,
  __in  PMINIDUMP_USER_STREAM_INFORMATION UserStreamParam,
  __in  PMINIDUMP_CALLBACK_INFORMATION CallbackParam
);
```

The MiniDump01 sample application (see **Figure 1**) shows how you call MiniDumpWriteDump to take a Mini dump without any of the optional parameters. It starts by checking the command-line arguments for a PID and then calls OpenProcess to get a process handle of the target. It then calls CreateFile to get a file handle. (Note that MiniDumpWriteDump supports any I/O target.) The file has an ISO date/time-based filename for uniqueness and chronological sorting: C:\dumps\minidump_YYYY-MM-DD_HH-MM-SS-MS.dmp. The directory is hardcoded to C:\dumps to ensure write access. This is required when doing post mortem debugging because the current folder (for example, System32) might not be writable.

The DumpType parameter is a MINIDUMP_TYPE-based bitmask that causes the inclusion (or exclusion) of particular types of memory. The MINIDUMP_TYPE flags are quite powerful and allow you to direct the capture of lots of memory regions without the need for additional coding via a callback. The options used by the MiniDump01 sample are the same as used by ProcDump. They create a (Mini) dump that can be used to summarize a process.

The DumpType always has the MiniDumpNormal flag present because it has a value of 0x00000000. The DumpType used includes every stack (MiniDumpNormal), all PEB and TEB information (MiniDumpWithProcessThreadData), the loaded module information plus any globals (MiniDumpWithDataSegs), all handle information (MiniDumpWithHandleData), all memory region information (MiniDumpWithFullMemoryInfo) and all thread time and affinity informaton (MiniDumpWithThreadInfo). With these flags, the dump created is a rich version of a Mini dump but it's still quite small (less than 30MB even for the biggest of applications). Example debugger commands supported by these MINIDUMP_TYPE flags are listed in **Figure 2**.

When using MiniDumpWriteDump, the dump taken will match the architecture of the capture program, not the target, so use a 32-bit version of your capture program when capturing a 32-bit process, and a 64-bit version of your capture program when capturing a 64-bit process. If you need to debug "Windows 32-bit on Windows 64-bit" (WOW64), you should take a 64-bit dump of the 32-bit process.

If you don't match the architecture (by accident or on purpose), you'll have to change the effective machine (.effmach x86) in the debugger to gain access to the 32-bit stacks in a 64-bit dump. Note that lots of debugger extensions fail in this scenario.

Exception Context Record

Microsoft Support engineers use the terms "hang dump" and "crash dump." When they ask for a crash dump, they want a dump with an exception context record. When they ask for a hang dump, they (usually) mean a series of dumps without one. A dump with exception information isn't always from the time of a crash, though; it can be from any time. The exception information is just a means to provide additional data in a dump. The user stream information is similar to the exception information in this respect.

An exception context record is the combination of a CONTEXT structure (the CPU registers) and an EXCEPTION_RECORD structure (the exception code, the instruction's address and so on). If you include an exception context record in the dump and run .ecxr, the current debugger context (thread and register state) is set to the instruction that raised the exception (see **Figure 3**).

To support .ecxr, the optional MINIDUMP_EXCEPTION_INFORMATION structure needs to be passed to MiniDumpWriteDump. You can get exception information at run time or at post mortem.

Runtime Exceptions

If you implement a debugger event loop, exception information is passed to you when the exception occurs. The debugger event loop will receive an EXCEPTION_DEBUG_EVENT structure for breakpoints, first chance exceptions and second chance exceptions.

The MiniDump02 sample application shows how to call MiniDumpWriteDump from within a debugger event loop so that the second chance exception's context record is included in the dump (equivalent to "procdump.exe -e"). This functionality runs when you use the -e switch. Because the code is quite long, the pseudo code for the application is shown in **Figure 4**. Refer to this article's code download for the full source code.

The application starts by checking the command-line arguments for a PID. Next it calls OpenProcess to get a process handle of the target, and then it calls CreateFile to get a file handle. If the -e switch is missing it takes a hang dump as before. If the -e switch is present, the application attaches to the target (as a debugger) using

Figure 3 **Changing Context to the Exception Context Record**

```
This dump file has an exception of interest stored in it.
The stored exception information can be accessed via .ecxr.
(17cc.1968174c.6f8): Access violation - code c0000005 (first/second
chance not available)
eax=00000000 ebx=001df788 ecx=75ba31e7 edx=00000000 esi=00000002 edi=00000000
eip=77c7014d esp=001df738 ebp=001df7d4 iopl=0         nv up ei pl zr na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000246
ntdll!NtWaitForMultipleObjects+0x15:
77c7014d 83c404          add     esp,4

0:000> .ecxr
eax=00000000 ebx=00000000 ecx=75ba31e7 edx=00000000 esi=00000001 edi=003b3374
eip=003b100d esp=001dfdbc ebp=001dfdfc iopl=0         nv up ei pl zr na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010246
CrashAV_x86!wmain+0x140xd:
00000001`3f251014 45891b003b100d 8900            mov     dword ptr
[r11],r11deax],eax  ds:002b:00000000`00000000=????????=????????
```

Your Online Source for Developer News!

Welcome to GrapeVine TV!

GvTv.GCPowerTools.com

GrapeCity PowerTools

GCPowerTools.com

Figure 4 **MiniDump02 Pseudo Code**

```
Function Main
Begin
  Check Command Line Arguments
  CreateFile(c:\dumps\minidump_YYYY-MM-DD_HH-MM-SS-MS.dmp)
  OpenProcess(PID)
  If "-e" Then
    DebugEventDump
    TerminateProcess(Process)
  Else
    WriteDump(NULL)

  CloseHandle(Process)
  CloseHandle(File)
End

Function WriteDump(Optional Exception Context Record)
Begin
  MiniDumpWriteDump(Optional Exception Context Record)
End

Function DebugEventDump
Begin
  DebugActiveProcess(PID)
  While (Not Done)
  Begin
    WaitForDebugEvent
    Switch (Debug Event Code)
    Begin
    Case EXCEPTION_DEBUG_EVENT
      If EXCEPTION_BREAKPOINT
        ContinueDebugEvent(DBG_CONTINUE)
      Else If "First Chance Exception"
        ContinueDebugEvent(DBG_EXCEPTION_NOT_HANDLED)
      Else "Second Chance Exception"
        OpenThread(Debug Event Thread ID)
        GetThreadContext
        WriteDump(Exception Context Record)
        CloseHandle(Thread)
        Done = True
    Case EXIT_PROCESS_DEBUG_EVENT
      ContinueDebugEvent(DBG_CONTINUE)
      Done = True
    Case CREATE_PROCESS_DEBUG_EVENT
      CloseHandle(CreateProcessInfo.hFile)
      ContinueDebugEvent(DBG_CONTINUE)
    Case LOAD_DLL_DEBUG_EVENT
      CloseHandle(LoadDll.hFile)
      ContinueDebugEvent(DBG_CONTINUE)
    Default
      ContinueDebugEvent(DBG_CONTINUE)
    End Switch
  End While
  DebugActiveProcessStop(PID)
End
```

DebugActiveProcess. In a while loop, it waits for a DEBUG_EVENT structure to be returned by WaitForDebugEvent. The switch statement uses the dwDebugEventCode member of the DEBUG_EVENT structure. After the dump has been taken, or the process has ended, DebugActiveProcessStop is called to detach from the target.

The EXCEPTION_DEBUG_EVENT within the DEBUG_EVENT structure contains an exception record within an exception. If the exception record is a breakpoint, it's handled locally by calling ContinueDebugEvent with DBG_CONTINUE. If the exception is a first chance, it isn't handled so that it can turn in to a second-chance exception (if the target doesn't have a handler). To do this, ContinueDebugEvent is called with DBG_EXCEPTION_NOT_HANDLED. The remaining scenario is a second-chance exception. Using the DEBUG_EVENT structure's dwThreadId, OpenThread is called to get a handle to the thread with the exception. The thread handle is used with GetThreadContext to populate

Figure 5 **MiniDump03 – JIT_DEBUG_INFO Handler**

```
int JitInfoDump(HANDLE hProcess, DWORD dwProcessId, HANDLE hFile,
MINIDUMP_TYPE miniDumpType, ULONG64 ulJitInfoAddr)
{
  int nResult = -1;

  JIT_DEBUG_INFO jitInfoTarget;
  SIZE_T numberOfBytesRead;

  if (ReadProcessMemory(hProcess, (void*)ulJitInfoAddr, &jitInfoTarget,
sizeof(jitInfoTarget), &numberOfBytesRead) &&
    (numberOfBytesRead == sizeof(jitInfoTarget)))
  {
    EXCEPTION_POINTERS exceptionPointers = {0};
    exceptionPointers.ContextRecord = (PCONTEXT)jitInfoTarget.lpContextRecord;
    exceptionPointers.ExceptionRecord = (PEXCEPTION_RECORD)jitInfoTarget.
lpExceptionRecord;

    MINIDUMP_EXCEPTION_INFORMATION    exceptionInfo = {0};
    exceptionInfo.ThreadId = jitInfoTarget.dwThreadID;
    exceptionInfo.ExceptionPointers = &exceptionPointers;
    exceptionInfo.ClientPointers = TRUE;

    nResult = WriteDump(hProcess, dwProcessId, hFile, miniDumpType, &exceptionInfo);
  }
  else
  {
    nResult = WriteDump(hProcess, dwProcessId, hFile, miniDumpType, NULL);
  }

  return nResult;
}
```

the CONTEXT structure required. (A word of caution here: the CONTEXT structure has grown in size over the years as additional registers have been added to processors. If a later OS increases the size of the CONTEXT structure, you'll need to recompile this code.) The obtained CONTEXT structure and the EXCEPTION_RECORD from the DEBUG_EVENT are used to populate an EXCEPTION_POINTERS structure, and this is used to populate a MINIDUMP_EXCEPTION_INFORMATION structure. This structure is passed to the application's WriteDump function for use with MiniDumpWriteDump.

The EXIT_PROCESS_DEBUG_EVENT is handled specifically for the scenario where the target ends before an exception occurs. ContinueDebugEvent is called with DBG_CONTINUE to acknowledge this event and the while loop is exited.

The CREATE_PROCESS_DEBUG_EVENT and LOAD_DLL_DEBUG_EVENT events are specifically handled as a HANDLE needs to be closed. These areas call ContinueDebugEvent with DBG_CONTINUE.

The Default case handles all other events by calling ContinueDebugEvent with DBG_CONTINUE to continue execution and close the handle passed.

Post Mortem Exceptions

Windows Vista introduced a third parameter to the post mortem debugger command line to support the passing of exception information. To receive the third parameter, you need to have a Debugger value (in the AeDebug key) that includes three %ld substitutions. The three values are: Process ID, Event ID and JIT Address. The JIT Address is the address of a JIT_DEBUG_INFO structure in the target's address space. Windows Error Reporting (WER) allocates this memory in the target's address space when

Figure 6 **Template Implementation of the MiniDumpCallback Function Prototype**

```
BOOL CALLBACK MiniDumpCallbackRoutine(
  __in     PVOID CallbackParam,
  __in     const PMINIDUMP_CALLBACK_INPUT CallbackInput,
  __inout  PMINIDUMP_CALLBACK_OUTPUT CallbackOutput
)
{    // Callback supported in Windows 2003 SP1 unless indicated
  // Switch statement is in call order
  switch (CallbackInput->CallbackType)
  {
  case IoStartCallback:  //  Available in Vista/Win2008
    break;
  case SecondaryFlagsCallback:  //  Available in Vista/Win2008
    break;
  case CancelCallback:
    break;
  case IncludeThreadCallback:
    break;
  case IncludeModuleCallback:
    break;
  case ModuleCallback:
    break;
  case ThreadCallback:
    break;
  case ThreadExCallback:
    break;
  case MemoryCallback:
    break;
  case RemoveMemoryCallback:
    break;
  case WriteKernelMinidumpCallback:
    break;
  case KernelMinidumpStatusCallback:
    break;
  case IncludeVmRegionCallback:  //  Available in Vista/Win2008
    break;
  case IoWriteAllCallback:  //  Available in Vista/Win2008
    break;
  case IoFinishCallback:  //  Available in Vista/Win2008
    break;
  case ReadMemoryFailureCallback:  // Available in Vista/Win2008
    break;
  }
  return TRUE;
}
```

WER is invoked as the result of an unhandled exception. It fills in the JIT_DEBUG_INFO structure, invokes the post mortem debugger (passing the address of the allocation), and then frees the memory after the post mortem debugger ends.

To determine the exception context record, the post mortem application reads the JIT_DEBUG_INFO structure from the target's address space. The structure has the address of a CONTEXT structure and EXCEPTION_RECORD structure in the target's address space. Instead of reading the CONTEXT and EXCEPTION_RECORD structures from the target's address space, I just filled in the EXCEPTION_POINTERS structure with these addresses and then set the ClientPointers member to TRUE in the MINIDUMP_EXCEPTION_INFORMATION structure. This makes the debugger do all of the heavy lifting. It will read the data from the target's address space (making allowance for architectural differences so that it's possible to take a 64-bit dump of a 32-bit process).

The MiniDump03 sample application shows how to implement the JIT_DEBUG_INFO support (see **Figure 5**).

When an application is invoked as the post mortem debugger, it is the application's responsibility to forcibly terminate the process via a TerminateProcess call. In the MiniDump03 example, TerminateProcess is called after the dump has been taken:

```
// Post Mortem (AeDebug) dump - JIT_DEBUG_INFO - Vista+
else if ((argc == 4) && (_stscanf_s(argv[3], _T("%ld"), &ulJitInfoAddr) == 1))
{
  nResult = JitInfoDump(hProcess, dwProcessId, hFile, miniDumpType, ulJitInfoAddr);
  // Terminate the process
  TerminateProcess(hProcess, -1);
}
```

To replace the post mortem debugger with your own application, you point the Debugger value of the AeDebug keys to the post mortem application with the appropriate architecture. By using the matching application, you remove the need to adjust the effective machine (.effmach) in the debugger.

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AeDebug
Debugger (REG_SZ) = "C:\dumps\minidump03_x64.exe %ld %ld %ld"

HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\Windows NT\
CurrentVersion\AeDebug
Debugger (REG_SZ) = "C:\dumps\minidump03_x86.exe %ld %ld %ld"
```

MiniDumpCallback Function

So far, the dump taken contains memory that the DumpType parameter indicated to include (and optionally an exception context record). By implementing a MiniDumpCallback function prototype, we can add not only additional memory regions but also some error handling. I'll describe later how you can implement *solely* the MiniDumpCallback function prototype for use with Sysinternals ProcDump v4.0.

There are currently 16 callback types that allow you to control multiple aspects of the dumping, such as the memory included for modules and threads, memory itself, the ability to cancel a dump that's in progress, the control of the dump file I/O and the handling of errors.

The switch statement in my template code (see **Figure 6**) includes all the callback types roughly in the call order of their first invocation. Some callbacks can be called more than once and can subsequently occur out of sequence. Equally, there's no contract for the order and thus it could change in future releases.

The MiniDump04 sample contains a callback that does two things; it includes the entire stack contents, and it ignores read failures. This example uses ThreadCallback and MemoryCallback

Figure 7 **WriteDump from the MiniDump04 Example**

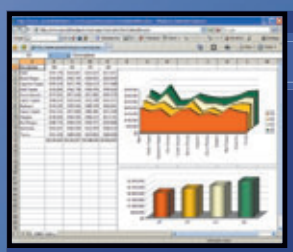
```
int WriteDump(HANDLE hProcess, DWORD dwProcessId, HANDLE hFile, MINIDUMP_
TYPE miniDumpType, PMINIDUMP_EXCEPTION_INFORMATION pExceptionParam)
{
  MemoryInfoNode* pRootMemoryInfoNode = NULL;

  MINIDUMP_CALLBACK_INFORMATION callbackInfo;
  callbackInfo.CallbackParam = &pRootMemoryInfoNode;
  callbackInfo.CallbackRoutine = MiniDumpCallbackRoutine;

  if (!MiniDumpWriteDump(hProcess, dwProcessId, hFile, miniDumpType,
pExceptionParam, NULL, &callbackInfo))
  {
    _tprintf(_T("Failed to create hang dump (Error: %08x)\n"), GetLastError());
    while (pRootMemoryInfoNode)
    {    // If there was an error, we'll need to cleanup here
      MemoryInfoNode* pNode = pRootMemoryInfoNode;
      pRootMemoryInfoNode = pNode->pNext;
      delete pNode;
    }
    return 11;
  }
  return 0;
}
```

20 Minutes to 4 Seconds...

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

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

SpreadsheetGear 2010



ASP.NET Excel Reporting

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

Excel Compatible Windows Forms Control

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

Create Dashboards from Excel Charts and Ranges

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

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

www.SpreadsheetGear.com.

SpreadsheetGear

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

Figure 8 **Stack Contents Comparison**

```
0:003> !teb
TEB at 000007fffffd8000
  ExceptionList:        0000000000000000
  StackBase:            000000001b4b0000
  StackLimit:           000000001b4af000
  SubSystemTib:         0000000000000000
...

// No Callback
0:003> dp poi($teb+10) L6
00000000`1b4af000  ????????`???????? ????????`????????
00000000`1b4af010  ????????`???????? ????????`????????
00000000`1b4af020  ????????`???????? ????????`????????

// Callback
0:003> dp poi($teb+10) L6
00000000`1b4af000  00000000`00000000 00000000`00000000
00000000`1b4af010  00000000`00000000 00000000`00000000
00000000`1b4af020  00000000`00000000 00000000`00000000
```

to include the entire stack, and the ReadMemoryFailureCallback to ignore the read failures.

To invoke the callback, the optional MINIDUMP_CALLBACK_INFORMATION structure is passed to the MiniDumpWriteDump function. The structure's CallbackRoutine member is used to point to the MiniDumpCallback function implemented (MiniDumpCallbackRoutine in my template and sample). The CallbackParam member is a VOID* pointer that allows you to retain context between callback calls. My WriteDump function from the MiniDump04 sample is in **Figure 7**. The structure I have defined (MemoryInfoNode) to retain context between calls is a link list node that contains an address and a size, like so:

```
struct MemoryInfoNode
{
  MemoryInfoNode* pNext;
  ULONG64 ulBase;
  ULONG ulSize;
};
```

Entire Stack

When you use the MiniDumpWithProcessThreadData flag in the DumpType parameter, the contents of each stack are included from the stack base to the current stack pointer. My MiniDumpCallbackRoutine function implemented in the MiniDump04 sample augments this by including the remainder of the stack. By including the entire stack, you might be able to determine the origin of a stack trash.

A stack trash is when a buffer overflow occurs to a stack-based buffer. The buffer overflow writes over the stack return address and causes the execution of the "ret" opcode to POP the buffer contents as an instruction pointer, instead of the instruction pointer PUSHed by the "call" opcode. This results in execution of an invalid memory address, or even worse, the execution of a random piece of code.

When a stack trash occurs, the memory below (remember, stacks grow downward) the current stack pointer will still contain the unmodified stack data. With this data, you can determine the contents of the buffer and—most of the time—the functions that were called to generate the contents of the buffer.

If you compare the memory above the stack limit in a dump taken with and without the additional stack memory, you will see that the memory is displayed as missing (the ? symbol) in the normal dump, but is included in the dump made with the callback (see **Figure 8**).

The first part of the "entire stack" code is the handling of the ThreadCallback callback type (see **Figure 9**). This callback type is called once per thread in the process. The callback is passed a MINIDUMP_THREAD_CALLBACK structure via the CallbackInput parameter. The structure includes a StackBase member that's the stack base of the thread. The StackEnd member is the current stack pointer (esp/rsp for x86/x64 respectively). The structure doesn't include the stack limit (part of the Thread Environment Block).

The example takes a simplistic approach and assumes that the stack is 1MB in size—this is the default for most applications. In the case where this is below the stack pointer, the DumpType parameter will cause the memory to be included. In the case where the stack is larger than 1MB, a partial stack will be included. And in the case where the stack is smaller than 1MB, additional data will just be included. Note that if the memory range requested by the callback spans a free region or overlaps another inclusion, no error occurs.

The StackBase and the 1MB offset are recorded in a new instantiation of the MemoryInfoNode structure I've defined. The new instantiation is added to the front of the link list that's passed to the callback via the CallbackParam argument. After the multiple invocations of ThreadCallback, the linked list contains multiple nodes of additional memory to include.

The last part of the "entire stack" code is the handling of the MemoryCallback callback type (see **Figure 10**). MemoryCallback is called continuously while you return TRUE from the callback, and provides a nonzero value for the MemoryBase and MemorySize members of the MINIDUMP_CALLBACK_OUTPUT structure.

The code sets the values on the CallbackOutput parameter and then deletes the node from the linked list. After multiple invocations of MemoryCallback, the linked list will contain no more nodes, and zero values are returned to end the MemoryCallback invocation.

Figure 9 **ThreadCallback Is Used to Collect the Stack Region of Each Thread**

```
case ThreadCallback:
{    // We aren't passed the StackLimit so we use a 1MB offset from StackBase
  MemoryInfoNode** ppRoot = (MemoryInfoNode**)CallbackParam;
  if (ppRoot)
  {
    MemoryInfoNode* pNode = new MemoryInfoNode;
    pNode->ulBase = CallbackInput->Thread.StackBase - 0x100000;
    pNode->ulSize = 0x100000; // 1MB
    pNode->pNext = *ppRoot;
    *ppRoot = pNode;
  }
}
break;
```

Figure 10 **MemoryCallback Is Continually Called While Returning a Stack Region**

```
case MemoryCallback:
{    // Remove the root node and return its members in the callback
  MemoryInfoNode** ppRoot = (MemoryInfoNode**)CallbackParam;
  if (ppRoot && *ppRoot)
  {
    MemoryInfoNode* pNode = *ppRoot;
    *ppRoot = pNode->pNext;
    CallbackOutput->MemoryBase = pNode->ulBase;
    CallbackOutput->MemorySize = pNode->ulSize;
    delete pNode;
  }
}
break;
```

Figure 11 **The .dumpdebug Command Output**

```
0:000> .dumpdebug
----- User Mini Dump Analysis
...
Stream 3: type MemoryListStream (5), size 00000194, RVA 00000E86
  25 memory ranges
  range#    RVA      Address      Size
       0 0000101A    7efdb000   00005000
       1 0000601A    001d6000   00009734
       2 0000F74E    00010000   00021000
       3 0003074E    003b0f8d   00000100
       4 0003084E    003b3000   00001000
...
```

Figure 12 **ReadMemoryFailureCallback Is Called on Read Failures**

```
case ReadMemoryFailureCallback:  // DbgHelp.dll v6.5; Available in Vista/Win2008
  {    //  There has been a failure to read memory. Set Status to S_OK to ignore it.
    CallbackOutput->Status = S_OK;
  }
  break;
```

Note that the MemoryBase and MemorySize members are set to zero when passed; you just need to return TRUE.

You can use the MemoryListStream area of the .dumpdebug command output to see all of the memory regions in the dump (note that adjacent blocks might be merged). See **Figure 11**.

Read Memory Failure

The last piece of code is quite simple (see **Figure 12**). It sets the Status member of the MINIDUMP_CALLBACK_OUTPUT structure to S_OK to signal that it's OK to omit a region of memory that could not be read during the capture.

In this simple implementation, the callback implements the same functionality as the MiniDumpIgnoreInaccessibleMemory flag. The ReadMemoryFailureCallback callback is passed the offset, number of bytes and the failure code via the MINIDUMP_READ_MEMORY_FAILURE_CALLBACK structure in the CallbackInput parameter. In a more complex callback, you could use this information to determine if the memory is critical to dump analysis, and whether the dump should be aborted.

Dissecting Memory

So how do you know what you can and can't afford to remove from a dump? Sysinternals VMMap is a great way to see what an application's memory looks like. If you use Sysinternals VMMap on a managed process, you'll notice that there are allocations associated with the garbage collected (GC) heap, and allocations associated with the application's images and mapped files. It is the GC heap that you need in a dump of a managed process because the Son of Strike (SOS) debugger extension requires intact data structures from within the GC heap to interpret the dump.

To determine the location of the GC heap, you could take an exacting approach by starting a Debugging Engine (DbgEng) session against the target using DebugCreate and IDebugClient::AttachProcess. With this debugging session, you could load the SOS debugger extension and run commands to return the heap information (this is an example of using domain knowledge).

Alternatively, you could use heuristics. You include all regions that have a memory type of Private (MEMORY_PRIVATE) or a Protection of Read/Write (PAGE_READWRITE or PAGE_EXECUTE_READWRITE). This collects more memory than absolutely necessary but still makes a significant saving by excluding the application itself. The MiniDump05 sample takes this approach (see **Figure 13**) by replacing the MiniDump04 sample's thread stack code with a once-only VirtualQueryEx loop in the ThreadCallback callback (the new logic still causes the entire stack to be included as before). It then uses the same MemoryCallback code used in the MiniDump04 sample to include the memory in the dump.

Mapped Memory Image Files

You may be wondering how you can debug a dump with image regions (MEM_IMAGE) missing. For example: How would you see the code that's executing? The answer is a bit out-of-the-box. When the debugger needs to access a missing image region in a non-Full dump, it obtains the data from the image file instead. It finds the image file by using the module load path, the PDB's original image file location, or uses the .sympath/.exepath search paths. If you run lmvm <module>, you'll see a "Mapped memory image file" line indicating that the file has been mapped in to the dump, like so:

```
0:000> lmvm CrashAV_x86
start    end        module name
003b0000 003b6000   CrashAV_x86   (deferred)
  Mapped memory image file: C:\dumps\CrashAV_x86.exe
  Image path: C:\dumps\CrashAV_x86.exe
  Image name: CrashAV_x86.exe
...
```

Relying on the "Mapped memory image file" ability of the debugger is a great approach to keeping the size of dumps small. It works particularly well with native applications, because the binaries compiled are the ones used and are therefore available on your internal build server (and pointed to by the PDBs). With managed applications, the JIT compilation on the remote customer's computer complicates this. If you want to debug a managed application dump from another

Figure 13 **MiniDump05—ThreadCallback Is Used Once to Collect the Memory Regions**

```
case ThreadCallback:
{    // Collect all of the committed MEM_PRIVATE and R/W memory
  MemoryInfoNode** ppRoot = (MemoryInfoNode**)CallbackParam;
  if (ppRoot && !*ppRoot)    // Only do this once
  {
    MEMORY_BASIC_INFORMATION mbi;
    ULONG64 ulAddress = 0;
    SIZE_T dwSize = VirtualQueryEx(CallbackInput->ProcessHandle, (void*)
ulAddress, &mbi, sizeof(MEMORY_BASIC_INFORMATION));
    while (dwSize == sizeof(MEMORY_BASIC_INFORMATION))
    {
      if ((mbi.State == MEM_COMMIT) &&
        ((mbi.Type == MEM_PRIVATE) || (mbi.Protect == PAGE_READWRITE) ||
(mbi.Protect == PAGE_EXECUTE_READWRITE)))
      {
        MemoryInfoNode* pNode = new MemoryInfoNode;
        pNode->ulBase = (ULONG64)mbi.BaseAddress;
        pNode->ulSize = (ULONG)mbi.RegionSize;
        pNode->pNext = *ppRoot;
        *ppRoot = pNode;
      }
      // Loop
      ulAddress = (ULONG64)mbi.BaseAddress + mbi.RegionSize;
      dwSize = VirtualQueryEx(CallbackInput->ProcessHandle, (void*)
ulAddress, &mbi, sizeof(MEMORY_BASIC_INFORMATION));
    }
  }
}
break;
```

Hosted Team Foundation Server 2010

No Risk Trial: First 30 days FREE & No Setup Fees

discountASP.net
Team Foundation Server Hosting

Microsoft Partner
Gold Hosting

TFS Hosting Features:

Source Code Version Control
Bug/Issue Tracking
Unlimited Projects
5gb of Disk Space
Monthly Billing
Visual Studio 2010/2008 Integration

TFS Build Server Available

Continuous Integration
Rolling Builds
Gated Check-in
Private Builds
Schedule Builds

For First 30 Days Free & No Setup Fees
www.DiscountASP.NET/tfs/msdn

Watch Getting Started with Hosted TFS Now
Scan this code to watch how to get started with Hosted Team Foundation Server from your mobile phone.

Figure 14 **MiniDumpCallbackRoutine Changed to Use a Global Instead of CallbackParam**

```
MemoryInfoNode* g_pRootMemoryInfoNode = NULL;
...
case IncludeThreadCallback:
{
  while (g_pRootMemoryInfoNode)
  {    // Unexpected cleanup required
    MemoryInfoNode* pNode = g_pRootMemoryInfoNode;
    g_pRootMemoryInfoNode = pNode->pNext;
    delete pNode;
  }
}
break;
...
case ThreadCallback:
{    // Collect all of committed MEM_PRIVATE and R/W memory
  if (!g_pRootMemoryInfoNode)    // Only do this once
  {
...
    pNode->pNext = g_pRootMemoryInfoNode;
    g_pRootMemoryInfoNode = pNode;
...
  }
}
break;
...
case MemoryCallback:
{    // Remove the root node and return its members in the callback
  if (g_pRootMemoryInfoNode)
  {
    MemoryInfoNode* pNode = g_pRootMemoryInfoNode;
    g_pRootMemoryInfoNode = pNode->pNext;
    CallbackOutput->MemoryBase = pNode->ulBase;
    CallbackOutput->MemorySize = pNode->ulSize;
    delete pNode;
  }
}
break;
```

computer, you'll need to copy the binaries (as well as the dumps) locally. This is still a savings because multiple dumps can be taken quickly, and then a single (large) application image file collection can be done without outage. To simplify the file collection, you could use the ModuleCallback to write out a script that collects the modules (files) referenced in the dump.

Plug Me In!

Changing the standalone application to use Sysinternals ProcDump v4.0 makes your life much easier. You no longer have to implement all of the code associated with calling MiniDumpWriteDump, and, more important, all of the code to trigger the dump at the right time. You just need to implement a MiniDumpCallback function and export it as MiniDumpCallbackRoutine in a DLL.

The MiniDump06 sample (see **Figure 14**) includes the callback code from MiniDump05 with just a few modifications. The new MiniDump06 project compiles the callback code as a DLL. The project exports the MiniDumpCallbackRoutine (case sensitive) using a DEF file:

```
LIBRARY    "MiniDump06"
EXPORTS
  MiniDumpCallbackRoutine   @1
```

Since ProcDump passes a CallbackParam value of NULL, the function needs to use a global variable instead to track its progress via my MemoryInfoNode structure. On the first IncludeThreadCallback, there's new code to reset (delete) the global variable if set from a prior capture. This replaces the code that was implemented after a failed MiniDumpWriteDump call in my WriteDump function.

To use your DLL with ProcDump, you specify the -d switch followed by the name of your DLL that matches the architecture of the capture. The -d switch is available when taking Mini dumps (no switch) and Full (-ma) dumps; it isn't available when taking MiniPlus (-mp) dumps:

```
procdump.exe -d MiniDump06_x64.dll notepad.exe
procdump.exe -ma -d MiniDump06_x64.dll notepad.exe
```

Note that the callback is invoked with different callback types than the ones described in my samples when a Full dump (-ma) is being taken (refer to the MSDN Library for the documentation). The MiniDumpWriteDump function treats the dump as a Full dump when the DumpType parameter contains MiniDumpWithFullMemory.

Sysinternals ProcDump (procdump.exe) is a 32-bit application that extracts from itself the 64-bit version (procdump64.exe) when required. After procdump64.exe has been extracted and launched by procdump.exe, procdump64.exe will load your (64-bit) DLL. As such, debugging your 64-bit DLL is quite tricky because the launched application is not the desired target. The easiest thing to do to support debugging your 64-bit DLL is to copy the temporary procdump64.exe to another folder and then debug using the copy. In this way, no extraction will occur and your DLL will be loaded in the application you launch from within the debugger (Visual Studio, for example).

Break!

Determining the origin of crashes and hangs is not easy when you can afford only a Mini dump. Making a dump file with additional key information solves this without incurring the expense of a Full dump.

If you're interested in implementing your own dump application or DLL, I suggest you investigate the effectiveness of the Sysinternals ProcDump, WER and AdPlus utilities first—don't reinvent the wheel.

When writing a callback, make sure you spend the time to understand the exact layout of memory within your application. Take Sysinternals VMMap snapshots and use dumps of the application to delve into the details. Making smaller and smaller dumps is an iterative approach. Start including/excluding obvious areas, and then refine your algorithm. You may need to use both heuristic and domain knowledge approaches to get you to your goal. You can help your decision making by modifying the target application. For example, you could use well-known (and unique) allocation sizes for each type of memory use. The important thing is to think creatively when deciding how to determine what memory is required, in both the target application and the dumping application.

If you're a kernel developer and are interested in callbacks, there's a similar kernel-based mechanism; refer to the BugCheckCallback Routine documentation on msdn.com. ■

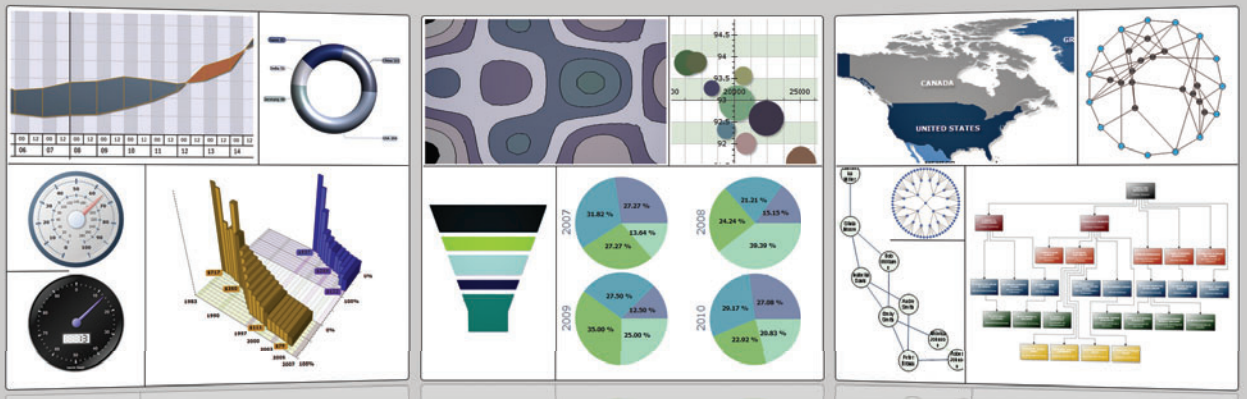
ANDREW RICHARDS *is a Microsoft senior escalation engineer for Windows OEM. He has a passion for support tools, and is continually creating debugger extensions and callbacks, and applications that simplify the job of support engineers.He can be contacted at andrew.richards@microsoft.com.*

THANKS to the following technical experts for reviewing this article: *Drew Bliss and Mark Russinovich*

NEVRON
visualize your success

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 PROFFESIONALS

- Nevron Reporting Services Vision instantly enhances your sql server reporting services 2005/2008 reports with the industry leading data visualization technology.
- Chart for SSRS
- Gauge for SSRS

- Nevron Sharepoint Vision instantly converts your Sharepoint pages into advanced dashboards and reports, that unite powerful data analysis with industry leading data visualization.
- Chart for Sharepoint
- Gauge for Sharepoint

Make sure that your data is making the visual statement it deserves by downloading your free evaluation copy from www.nevron.com today.

Microsoft .NET | ASP.net AJAX | Microsoft SharePoint 2007/2010 | Microsoft SQL Server 2005/2008 | Microsoft Visual Studio PARTNER

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

Experience the Devexpress Difference

"As a training, mentoring and consulting company, we are often put on the spot as to which vendors we like for .Net tools. There is only one answer from my company and that is Developer Express. We have used Developer Express tools in our projects for the past five years and the company continues to impress me with the quality of their tools.

They simply work. You get what you pay for. Mileage will vary with other vendors but I can assure you Developer Express is a sure bet. "

–Mark Dunn, MCT, MCAD, MCDBA, MCSD .Net

"Our flagship product needed extensive visualizations including charts and graphs. We were looking for a single charting system that could address all of these needs, and handle large volumes of data. It needed to look attractive yet blend into our application.

With *XtraCharts* we were able to create high performance, real-time graphs of performance data through to detailed analytical bar charts. *XtraCharts* was the only option that was fast enough to handle the tens of thousands of data points our customers routinely throw at it. "

–Kendall Miller

Read More User Comments at:
DevExpress.com/Comments

Hack-Proofing Your ASP.NET Applications

Adam Tuliper

Almost every day, the mainstream media reports that another site has been hacked. These constant intrusions by prominent hacker groups leave developers wondering if those groups are using advanced techniques for their nefarious work. Although some modern attacks can be quite complex, often, the most effective ones are remarkably simple—and have been in use for years. Luckily, these attacks are usually surprisingly easy to protect against.

I'm going to take a look at some of the most common types of hacking attacks over the course of two articles. This first article will cover SQL injection and parameter tampering, while the second part, which will appear in the January issue, will focus on cross-site scripting and cross-site request forgery.

In case you're wondering whether only large sites have to be concerned about hacking, the answer is simple: All developers must be concerned about hack attempts in their applications. It's your job to protect your applications; your users expect it. Even small applications on the Internet are open to probing because of the vast number of automatic hacking programs available. Suppose your Users or Customers table is stolen and the passwords for those tables are also used for other applications. Of course, it's recommended that users always use different passwords, but the reality is, they don't. You don't want to have to notify your users that your application was the gateway for information stealing. The very small blog of a friend of mine, covering his trip to Mt. Everest was hacked and everything deleted for no apparent reason. Without protection, virtually no application is safe.

Unless your network is physically unplugged from any outside communications device, the potential exists for someone to hop on to your network via proxy configuration issues; Remote Desktop Protocol (RDP) or Virtual Private Network (VPN) attacks; remote code execution vulnerabilities executed by an internal user simply visiting a Web page; guessed passwords; inadequate firewall rules; Wi-Fi (most Wi-Fi security can be cracked by an attacker sitting in your parking lot); social engineering tricks that get people to voluntarily give out sensitive information, and other entry points. Unless completely detached from the outside world, no environment can be assumed to be entirely safe.

Now that I've scared you (I hope) into believing that the threat of hacking is very real and all of your applications can be probed, let's get started understanding these hacks and how you can prevent them!

This article discusses:

- Web Forms and MVC security
- SQL injection
- Parameter tampering

Technologies discussed:

ASP.NET, SQL Server

SQL Injection

What Is It? SQL injection is an attack in which one or more commands are inserted into a query to form a new query that was never intended by the developer. This almost always occurs when dynamic SQL is being used; that is, when you're concatenating strings in your code to form SQL statements. SQL injection can occur in

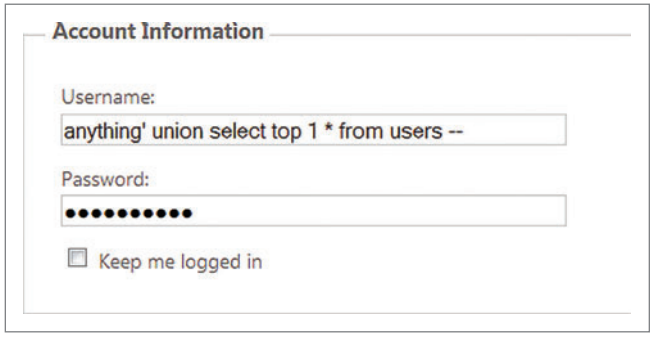


Figure 1 **Malicious Input Instead of a Valid Username**

your Microsoft .NET Framework code if you're forming a query or procedure call, and it can occur in your server-side T-SQL code as well, such as in the case of dynamic SQL in stored procedures.

SQL injection is particularly dangerous because it can be used not only to query and edit data, but also to run database commands that are limited only by database user or database service account permissions. If your SQL Server is configured to run as an administrator account and your application user belongs to the sysadmin role, be especially concerned. Attacks using SQL injection can run system commands to perform the following:

- Install backdoors
- Transfer an entire database over port 80
- Install network sniffers to steal passwords and other sensitive data
- Crack passwords
- Enumerate your internal network, including scanning the ports of other machines
- Download files
- Run programs
- Delete files
- Become part of a botnet
- Query autocomplete passwords stored on the system
- Create new users
- Create, delete and edit data; create and drop tables

Figure 2 **Parameterized Query**

```
using (SqlConnection connection = new SqlConnection(
  ConfigurationManager.ConnectionStrings[1].ConnectionString))
{
  using (SqlDataAdapter adapter = new SqlDataAdapter())
  {
    // Note we use a dynamic 'like' clause
    string query = @"Select Name, Description, Keywords From Product
                  Where Name Like '%' + @ProductName + '%'
                  Order By Name Asc";
    using (SqlCommand command = new SqlCommand(query, connection))
    {
      command.Parameters.Add(new SqlParameter("@ProductName", searchText));

      // Get data
      DataSet dataSet = new DataSet();
      adapter.SelectCommand = command;
      adapter.Fill(dataSet, "ProductResults");

      // Populate the datagrid
      productResults.DataSource = dataSet.Tables[0];
      productResults.DataBind();
    }
  }
}
```

This is not a comprehensive list and the danger is constrained only by the permissions in place—and the creativity of the attacker.

SQL injection has been around for so long that I'm often asked if it's still a concern. The answer is *absolutely* and attackers use it very often. In fact, outside of Denial of Service (DoS) attacks, SQL injection is the most commonly used attack.

How Is It Exploited? SQL injection is generally exploited through direct entry on a Web page or through parameter tampering, which usually involves altering not just forms or URIs—cookies, headers and so forth are also subject to attack if the application uses these values in an insecure SQL statement (I'll discuss this later in the article).

Let's look at an example of SQL injection via form tampering. This is a scenario I've seen many times in production code. Your code may not be exactly the same, but this is a common way developers check login credentials.

Here's the dynamically formed SQL statement for retrieving the user's login:

```
string loginSql = string.Format("select * from users where loginid= '{0}
  ' and password= '{1} '"", txtLoginId.Text, txtPassword.Text);
```

This forms the SQL statement:

```
select * from dbo.users where loginid='Administrator' and
  password='12345'
```

By itself, this isn't a problem. However, suppose the form field input looks like what's shown in **Figure 1**.

This input forms the SQL statement:

```
select * from dbo.users where loginid='anything' union select top 1 *
  from users --' and password='12345'
```

This example injects an unknown login ID of "anything," which by itself would return no records. However, it then *unions* those results with the first record in the database, while the subsequent "--" comments out the entire rest of the query so it's ignored. The attacker is not only able to log in, he can also return a valid user's record to the calling code without actually having any idea of a valid username.

Your code may not replicate this exact scenario, obviously, but the important point when analyzing your applications is to think about where the values that are included in queries commonly come from, including:

- Form fields
- URL parameters
- Stored values in the database
- Cookies
- Headers
- Files
- Isolated Storage

Not all of these may be immediately obvious. For example, why are headers a potential issue? If your application stores user profile information in headers and those values are used in dynamic queries, you could be vulnerable. These can all be sources for attack if you use dynamic SQL.

Web pages that include search functionality can be exceptionally easy prey for attackers because they provide a direct method to attempt injections.

This can give an attacker almost query editor functionality in a vulnerable application.

People have become far more aware of security issues in recent years, so systems are generally more hardened by default. For

instance, the system procedure xp_cmdshell is disabled in SQL Server 2005 and later (including SQL Express) instances. Don't let this give you the idea that an attacker can't run commands on your server, though. If the account your application uses for the database has a high enough permissions level, an attacker can simply inject the following command to re-enable the option:

```
EXECUTE SP_CONFIGURE 'xp_cmdshell', '1'
```

How Do You Prevent SQL Injection? Let's first discuss how not to fix this problem. A very common approach to fixing classic ASP applications was to simply replace dashes and quotes. Unfortunately, this is still widely used in .NET applications, often as the only means of protection:

```
string safeSql = "select * from users where loginId = " + userInput.
Replace("--", "");
safeSql = safeSql.Replace("'","''");
safeSql = safeSql.Replace("%","");
```

This approach assumes you have:

1. Protected every single query properly with these types of calls. It relies on the developer remembering to include these inline checks everywhere, instead of using a pattern that protects by default, even after coding all weekend and having run out of caffeine.
2. Typed-checked every single parameter. It's usually only a matter of time before a developer forgets to check that a parameter from a Web page is, for example, really a number, and then uses this number in a query for something like a ProductId without doing any string checks on it because, after all, it's a number. What if an attacker changes the ProductId and then it's simply read from the query string as shown here:

   URI: http://yoursite/product.aspx?productId=10

   Yields

   ```
   select * from products where productid=10
   ```

   And then the attacker injects commands such as:

   URI: http://yoursite/product.aspx?productId=10;select 1 col1 into #temp; drop table #temp;

   Which yields

   ```
   select * from products where productid=10;select 1 col1 into
   #temp; drop table #temp;
   ```

Oh no! You've just injected into an integer field that wasn't filtered through a string function or type checked. This is called a direct injection attack because no quotes are required and the injected portion is directly used in the query without quoting. You may say "I always make sure all of my data is checked," but that puts the responsibility of checking every single parameter manually on the developer and is very open to mistakes. Why not fix it the correct way by using a better pattern throughout your application?

So what's the correct way to prevent SQL injection? It's really fairly simple for most data access scenarios. The key is to use parameterized calls. You can actually have dynamic SQL that's safe as long as you make those calls parameterized. Here are the basic rules:

1. Ensure you're using only:
   - Stored procedures (without dynamic SQL)
   - Parameterized queries (see **Figure 2**)
   - Parameterized stored procedure calls (see **Figure 3**)
2. Dynamic SQL in stored procedures should be *parameterized* calls to sp_executesql. Avoid using exec—it doesn't support parameterized calls. Avoid concatenating strings with user input. See **Figure 4**.
3. Don't just replace dashes and quotes and think you're safe. Choose *consistent* data access methods as already described that prevent SQL injection without manual developer intervention and stick to them. If you rely on an escaping routine and happen to forget to call it in one location, you could be hacked. Moreover, there could be a vulnerability in the way you implement the escaping routine, as can be the case with a SQL truncation attack.
4. Validate input (see the following Parameter Tampering section) via type checks and casting; use regular expressions to limit to, for example, alpha-numeric data only

Figure 3 **Parameterized Stored Procedure Call**

```
//Example Parameterized Stored Procedure Call
string searchText = txtSearch.Text.Trim();
using (SqlConnection connection = new SqlConnection(ConfigurationManager.
ConnectionStrings[0].ConnectionString))
{
  using (SqlDataAdapter adapter = new SqlDataAdapter())
  {
    // Note: you do NOT use a query like:
    // string query = "dbo.Proc_SearchProduct" + productName + ")";

    // Keep this parameterized and use CommandType.StoredProcedure!!
    string query = "dbo.Proc_SearchProduct";
    Trace.Write(string.Format("Query is: {0}", query));

    using (SqlCommand command = new SqlCommand(query, connection))
    {
      command.Parameters.Add(new SqlParameter("@ProductName", searchText));
      command.CommandType = CommandType.StoredProcedure;

      // Get the data.
      DataSet products = new DataSet();
      adapter.SelectCommand = command;
      adapter.Fill(products, "ProductResults");

      // Populate the datagrid.
      productResults.DataSource = products.Tables[0];
      productResults.DataBind();
    }
  }
}
```

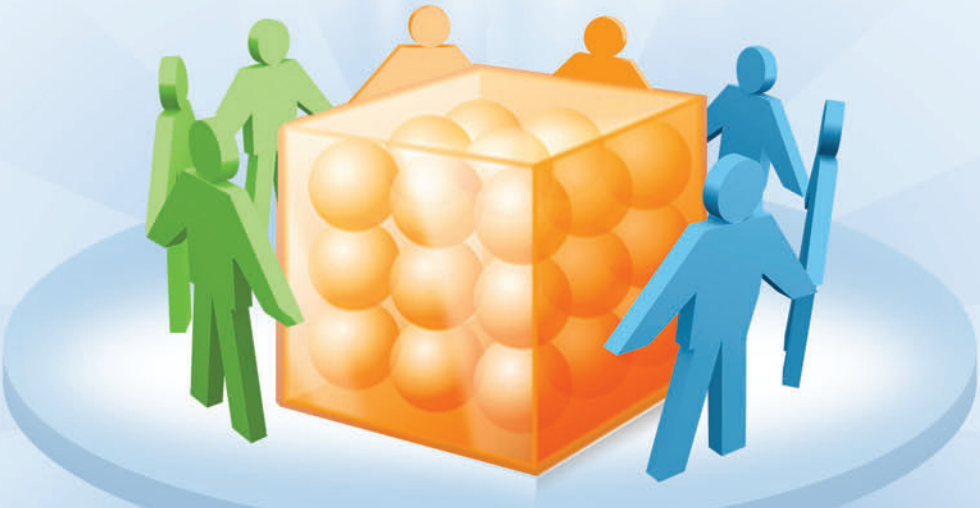
Figure 4 **Parameterized Call to sp_executesql**

```
/*
This is a demo of using dynamic sql, but using a safe parameterized query
*/

DECLARE @name varchar(20)
DECLARE @sql nvarchar(500)
DECLARE @parameter nvarchar(500)

/* Build the SQL string one time.*/
SET @sql= N'SELECT * FROM Customer  WHERE FirstName Like @Name Or
LastName Like @Name +''%''';
SET @parameter= N'@Name varchar(20)';
/* Execute the string with the first parameter value. */
SET @name = 'm%'; --ex. mary, m%, etc. note: -- does nothing as we would hope!
EXECUTE sp_executesql @sql, @parameter,
                      @Name = @name;

Note because of the lack of parameter support you DO NOT use:
exec 'select .. ' + @sql
```

Figure 5 **An Altered URL**

InstallAware

NEW INSTALLAWARE 2012



App-V for the Masses

Visit www.installaware.com/landing/msdn.asp for this special pricing

SPECIAL

Instantly Build
App-V Packages

for Only
$890

Windows 8 Ultimate

NOW WITH SUPPORT FOR WINDOWS 8

App-V Builder:
Compiles any existing InstallAware project as an App-V Package

App-V Viewer:
Opens and inspects the contents of any pre-existing App-V Package

Hybrid 32 bit & 64 bit:
Combines both 32 bit and 64 bit applications inside a single App-V

MSI Deployment:
Creates an MSI file to silently push your App-V Packages

Command Line Support:
Automates your App-V Build process through the command line

or pull important data from known sources; don't trust data coming from the Web page.

5. Audit your database object permissions to limit application user scope, thus limiting the surface area for attack. Grant permissions such as update, delete and insert only if that user must be able to perform those operations. Each separate application should have its own login to the database with restricted permissions. My open source SQL Server Permissions Auditor can help with this; please check it out at sqlpermissionsaudit.codeplex.com.

It's very important to audit your table permissions if you use parameterized queries. Parameterized queries require a user or a role to have permissions to access a table. Your application may be protected against SQL injection but what if another application that isn't protected touches your database? An attacker could start querying away in your database, so you'll want to make sure that each application has its own unique, restricted login. You should also audit permissions on database objects like views, procedures and tables. Stored procedures only require permissions on the procedure itself, not on the table, generally as long as there's no dynamic SQL inside the stored procedure, so security is a bit easier to manage on them. Again, my SQL Server Permissions Auditor can help.

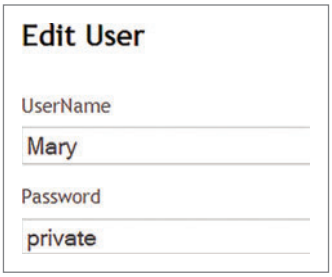


Figure 6 **An Edit User Form**

Figure 7 **Revealing a Hidden Field on the Form**

> It is very important to audit your table permissions if you use parameterized queries.

Note that the Entity Framework uses parameterized queries behind the scenes and thus is protected against SQL injection for normal usage scenarios. Some prefer to map their entities to stored procedures rather than opening up table permissions for the dynamic parameterized queries but there are valid arguments on both sides, and that's a decision for you to make. Note that if you're explicitly using Entity SQL, you'll want to be aware of some additional security considerations for your queries. Please see the MSDN Library page, "Security Considerations (Entity Framework)," at msdn.microsoft.com/library/cc716760.

Parameter Tampering

What Is It? Parameter tampering is an attack in which parameters are altered in order to change the application's expected functionality. The parameters may be on a form, query string, cookies, database and so on. I'll discuss attacks involving Web-based parameters.

How Is It Exploited? An attacker alters parameters to trick the application into performing an action it wasn't intending. Suppose you save a user's record by reading her user ID from the query string. Is this safe? No. An attacker can tamper with a URL in your application in a way similar to what's shown in **Figure 5**.

By doing so, the attacker could load a user account that you didn't intend. And all too often, application code like the following blindly trusts this userId:

```
// Bad practice!
string userId = Request.QueryString["userId"];
// Load user based on this ID
var user = LoadUser(userId);
```

Is there a better way? Yes! You can read values from a more trusted source like a user's session or from a membership or profile provider rather than trusting the form.

Various tools make it quite easy to tamper with more than just the query string. I recommend you check out some of the Web browser developer toolbars available to see hidden elements on your pages. I think you'll be surprised at what's uncovered and how easy your data is to tamper with. Consider the "Edit User" page shown in **Figure 6**. If you reveal the hidden fields on the page, you can see a user ID embedded right in the form, ready for tampering (see **Figure 7**). This field is used as the primary key for this user's record, and tampering with it alters which record is saved back to the database.

How Do You Prevent Parameter Tampering? Don't trust user-supplied data, and validate the data you receive that decisions are based on. You generally don't care if a user alters the middle name stored in his profile. However, you'd surely care if he tampered with the hidden form ID that represents his user record key. In cases like this, you can pull trusted data from a known source on the server rather than from the Web page. This information could be stored in the user's session upon login or in the membership provider.

For example, a far better approach uses information from the membership provider rather than using the form data:

Figure 8 **Using MVC Model Binding**

```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Edit([Bind(Exclude="UserId")] Order order)
{
   ...
   // All properties on the order object have been automatically populated and
   // typecast by the MVC model binder from the form to the model.
   Trace.Write(order.AddressId);
   Trace.Write(order.TotalAmount);
   // We don't want to trust the customer ID from a page
   // in case it's tampered with.
   // Get it from the profile provider or membership object
   order.UserId = Profile.UserId;
   // Or grab it from this location
   order.UserId = Membership.GetUser().ProviderUserKey.ToString();
   ...
   order.Save();}
   ...
   // etc.

}
```

```
// Better practice
int userId = Membership.GetUser().ProviderUserKey.ToString();
// Load user based on this ID
var user = LoadUser(userId);
```

Now that you've seen how untrustworthy data from the browser can be, let's take a look at some examples of validating this data to help clean things up a bit. Here are some typical Web Forms scenarios:

```
// 1. No check! Especially a problem because this productId is really numeric.
string productId = Request.QueryString["ProductId"];

// 2. Better check
int productId = int.Parse(Request.QueryString["ProductId"]);

// 3.Even better check
int productId = int.Parse(Request.
QueryString["ProductId"]);
if (!IsValidProductId(productId))
{
    throw new InvalidProductIdException(productId);
}
```

Figure 8 shows a typical MVC scenario with model binding that does basic automatic type conversions without having to explicitly cast parameters.

Model binding is a great feature of Model-View-Controller (MVC) that helps with parameter checks as the properties on the Order object will automatically be populated and converted to their defined types based on the form information. You can define Data Annotations on your model as well to include many different validations. Just be careful to limit what properties you allow to be populated and, again, *don't trust the page data for important items*. A good rule of thumb is to have one ViewModel for each View, so you'd completely exclude a UserId from the model in this Edit example.

Note that I use the [Bind(Exclude)] attribute here to limit what MVC is binding into my Model in order to control what I do or don't trust. This ensures the UserId won't come from the form data and thus can't be tampered with. Model binding and Data Annotations are beyond the scope of this article. I took just a brief look here to show how parameter typing can work in both Web Forms and MVC.

If you must include an ID field on the Web page that you "trust," please visit the MVC Security Extensions link (mvcsecurity.codeplex.com) for an attribute to assist with this.

Wrapping Up

In this article, I've presented two of the most common ways applications are hacked. As you can see, however, the attacks can be prevented or at least limited by making just a few changes in your applications. Of course, there are variations on these attacks and other ways your applications can be exploited. I'll cover two more types of attacks, cross-site scripting and cross-site request forgery, in the next issue. ■

ADAM TULIPER *is a software architect with Cegedim and has been developing software for more than 20 years. He's a national INETA Community Speaker, and regularly speaks at conferences and .NET user groups. Check him out on Twitter at twitter.com/AdamTuliper, his blog at completedevelopment.blogspot.com or secure-coding.com.*

THANKS to the following technical expert for reviewing this article: *Barry Dorrans*

Saving and Reusing Video Encoding Settings

Adi Shavit

From remote surveillance systems to embedded robotic platforms, applications that automatically or autonomously record video are getting ever more ubiquitous as the price of equipment such as cameras and storage space drops and computational performance increases. However, configuring such systems remotely often isn't an option, while deployment and installation must be simple and automatic.

I recently needed a way to preconfigure video recording settings on a remote system so it would automatically load and use the same settings at startup time. A Web search brought up many similar questions in multiple forums going back more than a decade, but with only partial and unsatisfying solutions.

In this article, I'll present a simple yet general way to allow video-processing applications to save video with consistent compression settings, thus avoiding having to manually specify the codec settings each time the application or the machine is started. I'll show how to access the internal setting buffers of a codec so that they can be easily saved, reloaded and reused. This will work for any video codec installed on the machine, without requiring the use of any codec-specific APIs.

This article discusses:

- Saving video with VfW
- Common ways to save settings
- A better hybrid approach
- The advantages of this approach

Technologies discussed:

Video for Windows

Code download available at:

code.msdn.microsoft.com/mag201112Video

Video 101

Raw video data is huge. Consider a modest VGA camera ambling along at a humble rate of 15 frames per second. Each three-channel 640×480 pixel frame is 900KB and generates a data rate of over 13MBps (that's about 105Mbps)! Slightly less modest video equipment, of the type you might find on a mobile phone, might have high definition (HD) resolutions, higher frame rates or more than 8 bits of color depth. A 90-minute HD movie will consume approximately 1TB in raw form. Fortunately, video data contains a lot of redundancy, and video compression algorithms can store video files quite efficiently. The type of compression algorithm chosen depends on the particular application, its requirements and its constraints. Several factors vary among compression methods, including real-time rates, performance considerations and the trade-off between visual quality and the resulting file size.

On Windows, video compression services are provided by codecs (COmpressor-DECompressor). I'll focus on the Video for Windows (VfW) API for saving compressed .avi video files. Audio Video Interleave (.avi) is a multimedia container file format that can contain multiple streams of both audio and video and allow synchronous audio-with-video playback of selected streams. I'll show how compression codec settings can be manually selected and stored in a persistent way so they can be loaded back later and automatically reused by the application on any computer on which the same codec is installed.

Saving Video with VfW

Although it was introduced back in 1992, VfW is an enduring API, still widely used today. New codecs and codec implementations, such as the ffmpeg suite (ffmpeg.org), continue to provide a VfW interface on Microsoft Windows.

The typical flow for a video recording program usually takes the form of the following steps:

1. Create a new AVI file, create a new uncompressed video stream within it, choose the required compression settings and create a new compressed stream from the uncompressed one.
2. Add new frames as desired.
3. When done, release resources in reverse order.

Using VfW it looks as follows:

1. Open and prepare the AVI video file using the following:
    1. AVIFileInit: Initializes the AVIFile library. Must be called before using any other AVIFile functions.
    2. AVIFileOpen: Opens an AVI file.
    3. AVIFileCreateStream: Creates a new video stream within the AVI file. An AVI file can include multiple streams (of various types).
    4. AVISaveOptions: Presents a standard Compression Options dialog (see Figure 1). When the user is finished selecting the compression options, the options are returned in an AVICOMPRESSOPTIONS structure.
    5. AVIMakeCompressedStream: Creates a compressed stream from the uncompressed stream returned from AVIFileCreateStream and the compression filter returned from AVISaveOptions.
    6. AVIStreamSetFormat: Sets the image format for the stream.
2. Add frames to the video stream:
    1. AVIStreamWrite: Add a single frame to the video stream. Call repeatedly with additional frames as desired.
3. Clean up and close:
    1. AVIStreamRelease: Closes the compressed stream (from AVIMakeCompressedStream).
    2. AVIStreamRelease: Closes the *un*compressed stream (from AVIFileCreateStream).
    3. AVISaveOptionsFree: Frees the resources allocated by the AVISaveOptions function (this is often forgotten, leading to memory leaks).
    4. AVIFileRelease: Closes the AVI file (from AVIFileOpen).
    5. AVIFileExit: Exit the AVIFile library.

Explaining in detail how to use VfW for recording video is beyond the scope of this article. You can find many examples and tutorials online. I'll focus here only on Step 1.4: the compressor selection and codec settings part. The source code accompanying this article is based on the C++ implementation from the OpenCV (opencv.willowgarage.com) open source project and can be downloaded from code.msdn.microsoft.com/mag201112Video. However, the techniques I show here are applicable to—and easily integrated into—any video recording application that uses the VfW API and isn't specifically targeted at a particular implementation or programming language.

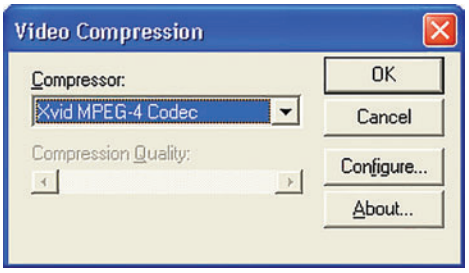


Figure 1 **The Video Compression Dialog Box Opened by the AVISaveOptions Function**

Common Solutions

As shown earlier, the common way to specify the compression settings is by calling the AVISaveOptions function to display the Compression Options dialog box. The user can then manually select the desired codec. The chosen codec may or may not have a codec "Configure…" option, which allows making codec-specific settings customizations. Naturally, on systems without an active user or even a GUI, this dialog box isn't an option.

The common alternative to manual selection is to programmatically select a particular codec using the codec's *fourcc* (four character code; see fourcc.org) identifier, and to set some additional, general settings common to all codecs, such as data rate, quality and key frame rate. The major drawback of this method is that it doesn't allow setting any *codec-specific* settings. Further, it makes the codec use whatever its current internal settings are. These may be defaults or, even worse, settings arbitrarily set by other programs. Additionally, not all fourcc codecs available on a system support the VfW API, which may cause the compressed stream creation to fail.

Some codecs provide special configuration tools that allow changing their internal settings externally. These tools may take the form of a GUI or a command-line tool. However, most codecs don't provide such tools, and because each tool is customized for a particular codec, you'd have to learn its particular syntax and usage to take advantage of it.

What I wanted was the flexibility of the manual selection mode with automated selection of the codec and all its internal settings.

The Best of Both Worlds

In the first approach, the AVISaveOptions call returns a fully filled-in AVICOMPRESSOPTIONS object. The second approach is, essentially, filling up some of the values in the AVICOMPRESSOPTIONS object, from within the code, without using AVISaveOptions.

In fact, it's possible to save the *full, internal* codec state after a call to AVISaveOptions so this state can later be restored without having to call AVISaveOptions again. The secret of *internal* video-compressor-specific data and format is in the opaque pointers lpParms and lpFormat of AVICOMPRESSOPTIONS. When returned from AVISaveOptions, these two pointers contain everything the AVIMakeCompressedStream function needs to make a compressed stream (these are presumably the resources released by AVISaveOptionsFree).

The essence of my technique, then, is saving the opaque data of lpParms and lpFormat to allow it to be reused. Fortunately, this is simple to do. The AVICOMPRESSOPTIONS struct contains two helpful integer members, cbParms and cbFormat, which state the size of the opaque binary buffers pointed to by lpParms and lpFormat, respectively. These sizes change according to the particular codec selected.

To summarize, there are three binary "blobs" that need to be saved after a call to

AVISaveOptions. These are the AVICOMPRESSOPTIONS object itself and the two binary buffers pointed to by its lpParms and lpFormat members. The length in bytes of these buffers is given, respectively, by cbParms and cbFormat. Restoring the codec settings is just reversing the process: Read the AVICOMPRESSOPTIONS object from the file and set its lpParms and lpFormat members to point at the respective binary buffers read from the file. The restored binary buffers are allocated and managed by the application itself.

There are many ways to store binary data. As shown in **Figure 2**, my sample code saves the data in binary form. Alternatively, in cases where nonprintable characters should be avoided (such as .txt or .xml files), I've successfully used Base-64 encoding to store the buffers.

Figure 2 **Example of Storing and Restoring Codec Settings**

```
bool CvVideoWriter_VFW::writeCodecParams( char const* configFileName ) const
{
  using namespace std;
  try
  { // Open config file
    ofstream cfgFile(configFileName, ios::out | ios::binary);
    cfgFile.exceptions ( fstream::failbit | fstream::badbit );
    // Write AVICOMPRESSOPTIONS struct data
    cfgFile.write(reinterpret_cast<char const*>(&copts_), sizeof(copts_));

    if (copts_.cbParms != 0)// Write codec param buffer
      cfgFile.write(reinterpret_cast<char const*>(copts_.lpParms), copts_.cbParms);

    if (copts_.cbFormat != 0)// Write codec format buffer
      cfgFile.write(reinterpret_cast<char const*>(copts_.lpFormat),
        copts_.cbFormat);
  }
  catch (fstream::failure const&)
  { return false; } // Write failed
  return true;
}

bool CvVideoWriter_VFW::readCodecParams( char const* configFileName )
{
  using namespace std;
  try
  { // Open config file
    ifstream cfgFile(configFileName, ios::in | ios::binary);
    cfgFile.exceptions (
      fstream::failbit | fstream::badbit | fstream::eofbit );
    // Read AVICOMPRESSOPTIONS struct data
    cfgFile.read(reinterpret_cast<char*>(&copts_), sizeof(copts_));

    if (copts_.cbParms != 0)
    { copts_Parms_.resize(copts_.cbParms,0);               // Allocate buffer
      cfgFile.read(&copts_Parms_[0], copts_Parms_.size()); // Read param buffer
      copts_.lpParms = &copts_Parms_[0];                   // Set lpParms to buffer
    }
    else
    { copts_Parms_.clear();
      copts_.lpParms = NULL;
    }

    if (copts_.cbFormat != 0)
    { copts_Format_.resize(copts_.cbFormat,0);             // Allocate buffer
      cfgFile.read(&copts_Format_[0], copts_Format_.size());// Read format buffer
      copts_.lpFormat = &copts_Format_[0];                 // Set lpFormat to buffer
    }
    else
    { copts_Format_.clear();
      copts_.lpFormat = NULL;
    }
  }
  catch (fstream::failure const&)
  { // A read failed, clean up
    ZeroMemory(&copts_,sizeof(AVICOMPRESSOPTIONS));
    copts_Parms_.clear();
    copts_Format_.clear();
    return false;
  }
  return true;
}
```

Sample Code Notes

The sample code captures video from a video camera or webcam and saves it to an AVI file. It uses the OpenCV VideoCapture class to access the camera and capture the video. Saving the video is done using a version of the OpenCV CvVideoWriter_VFW class that I modified to support saving the codec settings. I tried to keep the changes to the original code as small as possible. In the code, the second argument of CvVideoWriter_VFW::open is an std::string. If this string is four characters long (as in a *fourcc*), it's assumed to be a fourcc and the codec corresponding to this fourcc is chosen—converting from the four 1-byte characters to the fourcc integer code is commonly done with the mmioFOURCC macro, as in mmioFOURCC('D','I','V','X'); see tinyurl.com/mmioFOURCC for details. Otherwise, the string is taken as the name of a codec settings file. If the file exists, the settings are read from the file and used for the video compression. If it doesn't exist, then the Compression Options dialog box opens and a codec and its settings can be manually selected; the chosen settings are then saved to the file of the selected name to be reused the next time the application is run. Note: You'll need an OpenCV installation to compile and run the sample code.

Several Advantages

The solution I presented has several advantages. Changing the compression settings for an application is just a matter of changing one of its arguments. There's no need to recompile it or even stop the running app. The method is codec-independent and works for all VfW codecs. Some codecs even allow for additional customized image processing such as frame resizing and deinterlacing. All these options are automatically captured within the codec settings file.

The method works just as well on legacy Microsoft Windows systems (from Windows 2000 onward) and also with older compilers such as Visual Studio 6.0. Also, it can be used with a variety of languages that expose the VfW interface, such as Visual Basic and C#. It can be deployed to multiple machines regardless of the current internal codec state of each machine.

The main caveat is that this solution is guaranteed to work only when the exact same version of the codec is used. If a codec is missing on a particular machine, or is a different version of the one used to make the settings file, the results might be undetermined. Obviously, non VfW codecs can't be used, because this solution is based on the VfW API.

Possible extensions to this work include similarly saving audio compressor settings, which are handled identically to video compressor settings. Also, because AVISaveOptions natively supports AVIs with multiple streams, the same solution will also work in these cases, and the codec settings for each of the streams can be written to a single file.

The other common and more modern video and audio processing API on Windows is DirectShow. It should be possible to implement a similar solution for DirectShow as well, but that's a subject for a future article. ■

ADI SHAVIT *is a computer vision consultant. He has been working on image and video processing systems for more than 15 years, specializing in real-time video analysis. He can be reached at adishavit@gmail.com.*

THANKS to the following technical experts for reviewing this article: *Dana Shavit and Matthew Wilson*

Visual Studio ALM Rangers – Rise of the VM Factory

Brian Blackman, Paul Meyer and Willy-Peter Schaub

In this article, we delve into the Virtual Machine Factory (VM Factory), exploring its concepts and advantages.

This article is part of a series in which the Visual Studio ALM Rangers present guidance to assist you in solving problems in complex, real-world environments. Our goal is to help you improve the consistency and quality of your solutions and your overall Application Lifecycle Management (ALM) process.

To recap, the Rangers are a group of experts who promote collaboration among the Visual Studio product group, Microsoft Services and the Microsoft Most Valuable Professional (MVP) community by addressing missing functionality, removing adoption blockers and publishing best practices and guidance based on real-world experience.

The VM Factory (rangersvsvmfactory.codeplex.com) is the reference implementation of a software solution and associated guidance that automates the creation of virtual environments using a nearly fully automated and consistent factory strategy. The purpose of the VM Factory is to develop prescriptive guidance around virtualization, particularly for Visual Studio, Team Foundation Server (TFS), Lab Management and associated prerequisites. The main objectives, as shown (with corresponding numerals) in **Figure 1**, are: (1) flexible schema-driven automation; (2) a simple one-click virtual machine (VM) creation scenario; (3) a reusable factory that automates the installation and configuration of physical and virtual environments; (4) consistent and repeatable environments; (5) the ability to reuse preconfigured environments.

What the VM Factory objectives do not pursue is a reduction in build and configuration time. This is called an *anti-pattern*.

Although the VM Factory is far more efficient and disciplined than a human counterpart, the bulk of the environment generation time is made up of actual installation time, which can't be noticeably reduced by automation. The VM Factory initiative emerged from the Rangers ecosystem, which is renowned for the distributed, disconnected and often poor bandwidth environment constraints that it operates within (see the September 2011 article, "Visual Studio ALM Rangers—Reflections on Virtual Teams," at msdn.microsoft.com/magazine/hh394152). Even though sharing and downloading VMs over the Internet is feasible and is an accepted practice, it can be a challenge in locations where bandwidth is an expensive and scarce resource. Instead of investing days, or possibly weeks, to download a VM, the VM Factory allows identical images to be recreated remotely using local resources. Despite the limitations that local resources might have, these recreated images are delivered with minimal or no user intervention

This article discusses:

- VM Factory architecture
- Using Windows PowerShell
- Administration of the VM Factory
- Extending the VM Factory
- More resources

Technologies discussed:

VM Factory, Windows PowerShell, Lab Management, Team Foundation Server, Visual Studio 2010

Code download available at:

rangersvsvmfactory.codeplex.com

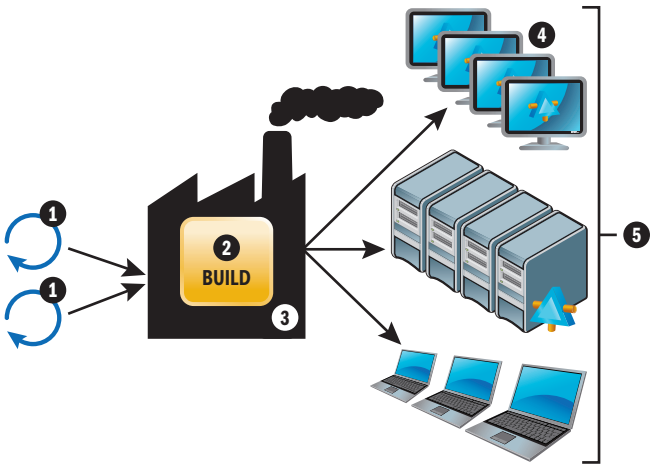


Figure 1 **VM Factory Objectives**

The delivery of consistent images is another core advantage we often forget to mention, but it's one of the main reasons the Rangers use the VM Factory. The image schemas flow from version to version, which means that when we create a new version (v1.1) to fix an issue, then another build to include a new software version (v2), we know the fix and the new software will flow to subsequent versions. This is especially helpful when the next version (v3) might not be built for months or even years. When that much time elapses between versions, a different team might be assigned to the project. If consistent images are in place, the VM Factory Task Sequence configuration and structure setup acts as a living document with which the new team can work.

Last but not least, the VM Factory allows on-demand image creation. This reduces the disk storage requirements on both desktop and portable computers, as well as the "angst" about whether an image pulled from storage is the most up-to-date image.

Virtualization within Lab Management provides an automated workflow for build, deployment and testing of software. Creating these virtual environments requires a lot of manual effort. You currently have to set up a library of VMs and templates for Microsoft System Center Virtual Machine Manager (VMM). Additionally, you have to work within a Lab Management topology.

Using the VM Factory significantly reduces the effort required to create VMs and VM templates. More importantly, the VM Factory makes it easier to change an OS or an application that's used for a VM or VM template. The VM Factory is easy because you only have to change a task sequence and then execute the automated recreation of the image. In the VM Factory, these are known as golden images and function as source VMs for Lab Management.

When you create the VM, you'll be prompted to select the task sequence to execute. We provide two task sequences for Visual Studio ALM Rangers Lab Management, both of which we'll cover in more detail later.

Factory Architecture at a Glance

As shown (with corresponding numerals) in **Figure 2**, the VM Factory architecture can be hosted on (1) any Windows platform with minimal prerequisites, such as a minimum 1GB of RAM and 100GB of free disk space dedicated to the VM Factory. The architecture is based on the (2) Microsoft Deployment Toolkit (MDT), one of the Microsoft solution accelerators (bit.ly/tp5QX4). The toolkit provides a (3) common administration console and comprehensive tools and guidance to define and manage deployment of OSes and applications. Using (4) task sequences, we define the sequence of installation and configuration steps that are required to deploy an OS and applications, such as Microsoft Office and Visual Studio, (5) use of custom scripts such as the Windows PowerShell scripts that are covered later in this article, and (6) referencing the media for the OS and applications to be used. Currently, the VM Factory assumes that all the media is physically present in the VM Factory. However, we're researching options that will allow us to reference other media sources, such as MSDN downloads, to deliver greater flexibility and reuse of existing resources.

Each VM Factory comes with (7) Litetouch CD Boot ISO images. These allow you to initiate a new installation and connect to the deployment share hosted on the VM Factory. During the boot and installation process, the (8) appropriate task sequence is selected. The task sequence defines the software, configuration and the scripts that are applied to the (9) target machine during the Litetouch guided installation.

The Visual Studio ALM Rangers VM Factory is a prepopulated MDT instance with predefined Factory Task Sequences. These task sequences build on top of the MDT task sequence templates for new machine deployment. They define all steps necessary to create a new machine from bare metal (or bare virtual metal in case of the VM Factory). Starting with formatting the drive and installing the OS, the steps continue up to (and including) the installation of applications.

The task sequences included in the VM Factory are used by the Rangers to build images for demonstrations, proof of concepts, hands-on labs and testing purposes. Following are descriptions of these task sequences:

- *Team Foundation Server base images* task sequences define *all-inclusive* Visual Studio images containing both TFS client and server features. Also included are Word 2007, Excel 2007, PowerPoint 2007 and, optionally, Microsoft Office SharePoint Server (MOSS) 2007. Task sequences for both x86 (based on Windows Server 2008) and x64 (Windows Server 2008 R2) are included in the VM Factory.

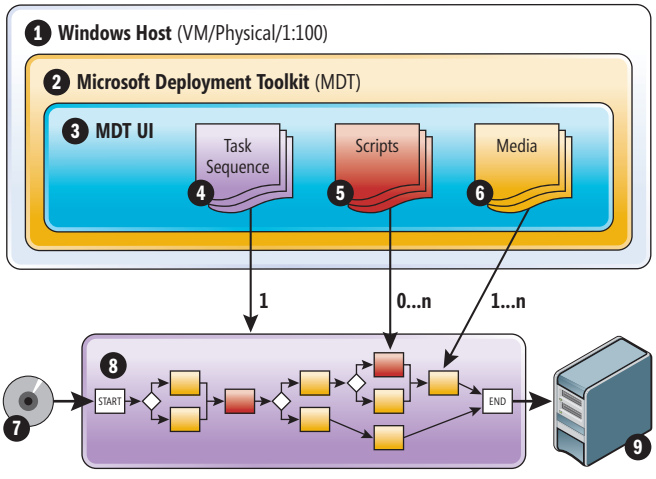


Figure 2 **VM Factory Architecture**

Figure 3 **Use of NTRights.exe in a Windows PowerShell Script**

```
function createTFSAccounts
{
  blankline
  "Creating the 5 TFS accounts. Namely TFSREPORTS"
  ", TFSSERVICE TFSBUILD,WSSSERVICE & SQLSERVICE with"
  " all their passwords set to P@ssw0rd"
  createUser "TFSREPORTS" "P@ssw0rd"
  createUser "TFSSERVICE" "P@ssw0rd"
  createUser "TFSBUILD" "P@ssw0rd" $true
  createUser "WSSSERVICE" "P@ssw0rd"
  createUser "SQLSERVICE" "P@ssw0rd"

  $currentLocal = Get-Location
# Set-Location 'C:\Program Files (x86)\Windows Resource Kits\Tools\'
  ./ntrights.exe +r SeInteractiveLogonRight -u TFSSERVICE
  ./ntrights.exe +r SeInteractiveLogonRight -u TFSBUILD
  Set-Location $currentLocal
}
```

Figure 4 **Use of the Windows PowerShell ADSI Type Adapter**

```
#Creates a user and sets the password to never expire
function createUser([String]$accountName, [string]$password,
[bool]$delegated = $false)
{
  $computer = [adsi] "WinNT://$hostname"
  $user = $computer.Create("User", $accountName)
  $user.SetPassword($password)
  $flags = 65536 -bor 64
  if ($delegated)
  {
    $flags = $flags -bor 1048576
  }
  $user.put("UserFlags",$flags)
  $user.SetInfo()
}
```

Figure 5 **Manipulating Shutdown Event Tracker**

```
function ShutdownEventTracker
{
  new-item 'HKLM:\SOFTWARE\Policies\Microsoft\Windows NT\Reliability'
    -erroraction silentlycontinue
  Set-ItemProperty -Path
    'HKLM:\SOFTWARE\Policies\Microsoft\Windows NT\Reliability'
    -name "ShutdownReasonOn" -Value 0
  Set-ItemProperty -Path
    'HKLM:\SOFTWARE\Policies\Microsoft\Windows NT\Reliability'
    -name "ShutdownReasonUI" -Value 0
}
```

- *Lab Management Hands-on Lab image* task sequence defines an all-up demonstration environment for Lab Management. It's based on the x64 TFS base image, but also contains Active Directory, DHCP and VMM. DHCP is fully configured but disabled for security reasons. VMM and Visual Studio Lab Management are both completely configured and will only require that you add a domain-joined Hyper-V host to the default host group in VMM.
- *Lab Management base image* task sequence is useful to create images to test your software using Visual Studio Lab Management. It defines an image based on Windows Server 2008 R2 and installs Visual Studio Test Agent, Visual Studio Test Controller, Visual Studio Lab Management Agent and Visual Studio Team Foundation Build. Most test configurations will require a combination of these agents.

When you set up your own factory, you can modify these sequences to your needs or use them as a base for your own custom task sequence. For more information about how to do this, see the "Administration" section later in this article.

Automation with Windows PowerShell

The VM Factory contains Windows PowerShell scripts developed as part of the Ranger factory initiative to automate the setup, configuration and tuning of the Visual Studio ALM Rangers base images. In addition to the Windows PowerShell scripts and CMD files, we make use of other Microsoft services or product applications such as ntrights.exe, stsadm.exe, psconfig.exe and iisreset.exe.

Initial factory plans called for only Windows PowerShell scripts. However, we adopted MDT as the Visual Studio ALM Rangers base image factory and call Windows PowerShell scripts to perform special configuration, tuning and tweaking of the MDT-based setup. We'll cover some important notes and a few snapshots of the scripts, directing you to the VM Factory for the full source.

To run the Windows PowerShell scripts successfully, you must have Windows PowerShell installed. The scripts are designed to run on Windows Server 2008 or later versions. Because the scripts aren't signed, and elevated privileges are required to make system

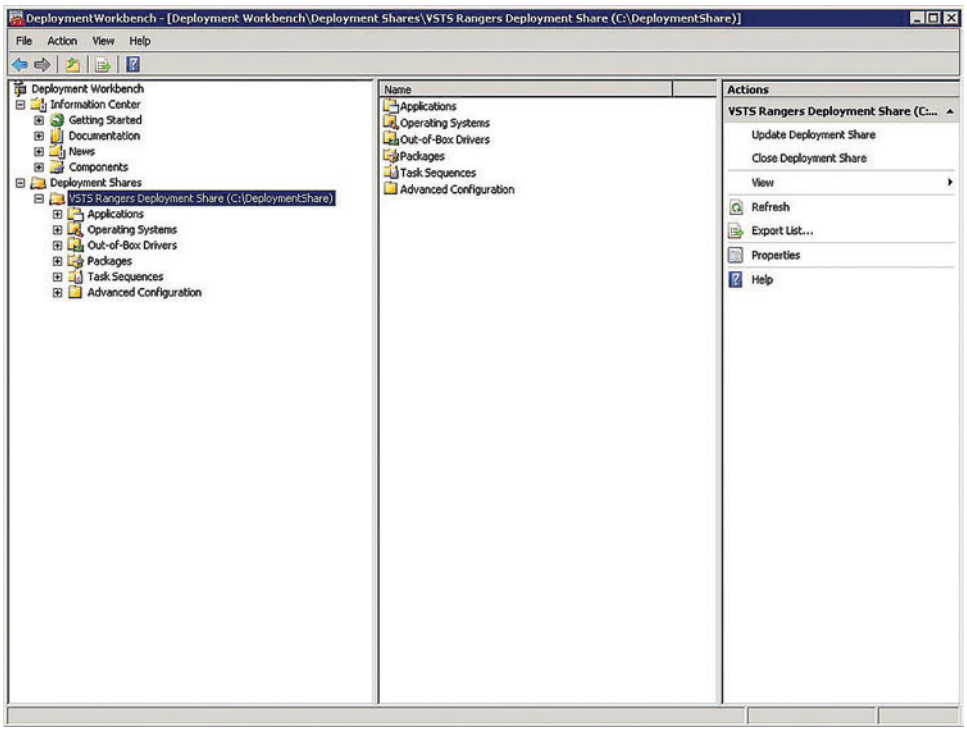


Figure 6 **Deployment Workbench Administration Interface**

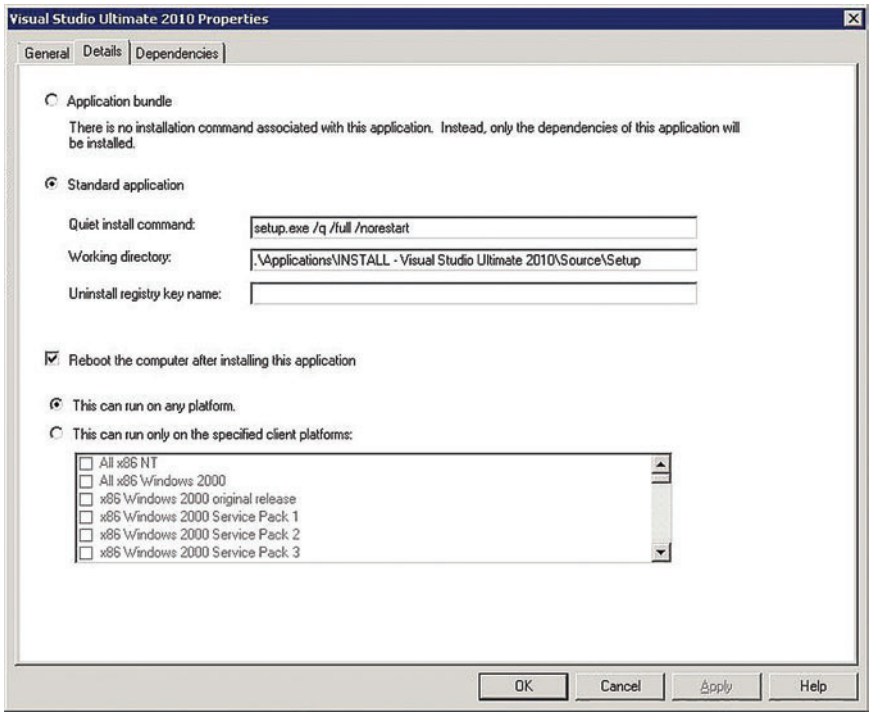


Figure 7 **Task Sequence Task Details**

changes, you must run the Windows PowerShell command "Set-ExecutionPolicy Unrestricted."

If you use PowerGUI from powergui.org, which is a GUI and script editor for Windows PowerShell, the scripts won't run in debug mode. You must run them in Windows PowerShell proper or by using Alt+F5 while using PowerGUI. And finally, the 1-pre.ps1 script requires NTRights.exe, which is installed from the Windows Server 2003 Resource Kit. Without NTRights.exe, you'll have to disable the calls at the last few lines of the script in the function createTFSAccounts and manually set Interactive Logon rights for the TFSSERVICE and TFSBUILD services accounts (see **Figure 3**). You can download the Windows Server 2003 Resource Kit Tools from: bit.ly/fbafh.

The pre- and post-factory installed Windows PowerShell scripts contain several functions that use various objects and type adapters, such as the Active Directory Services Interface (ADSI) type adapter shown in **Figure 4**.

The Windows PowerShell ADSI type adapter is used to create user accounts, set passwords and set the admin password to never expire. There's liberal use of the Set-ItemProperty and New-ItemProperty Windows PowerShell cmdlets to manipulate the registry for the VM Factory, such as enabling Prefetch, setting program priority and disabling some of the Windows Server 2003 reliability features. Making these changes to the registry minimizes manual configuration tasks that are required to achieve the benefits of automation, as shown in **Figure 5**.

As we mentioned earlier, we recommend that you explore the full source code so that you can understand how we use Windows PowerShell to automate many other tasks such as disabling the screen saver, disabling the inbound firewall, enabling remote desktop, disabling IE Enhanced Security, enabling super fetch and changing the server name.

Administration

Administration of the VM Factory is straightforward. Because the VM Factory is basically a prepopulated MDT deployment share, all possibilities of the MDT are at your disposal for customization or extension. The main tool for administering MDT deployment shares is the Deployment Workbench. The Help topics document the Deployment Workbench very well, but print-ready documentation can be downloaded from go.microsoft.com/?linkid=9707467. We'll provide a short description to get you started.

When you start the Deployment Workbench, you're presented with the familiar Management Console UI as shown in **Figure 6**. In the left pane, expand the Deployment Shares node. If you followed the instructions from the guidance on CodePlex correctly, the "VSTS Rangers Deployment Share" should be available. If not,

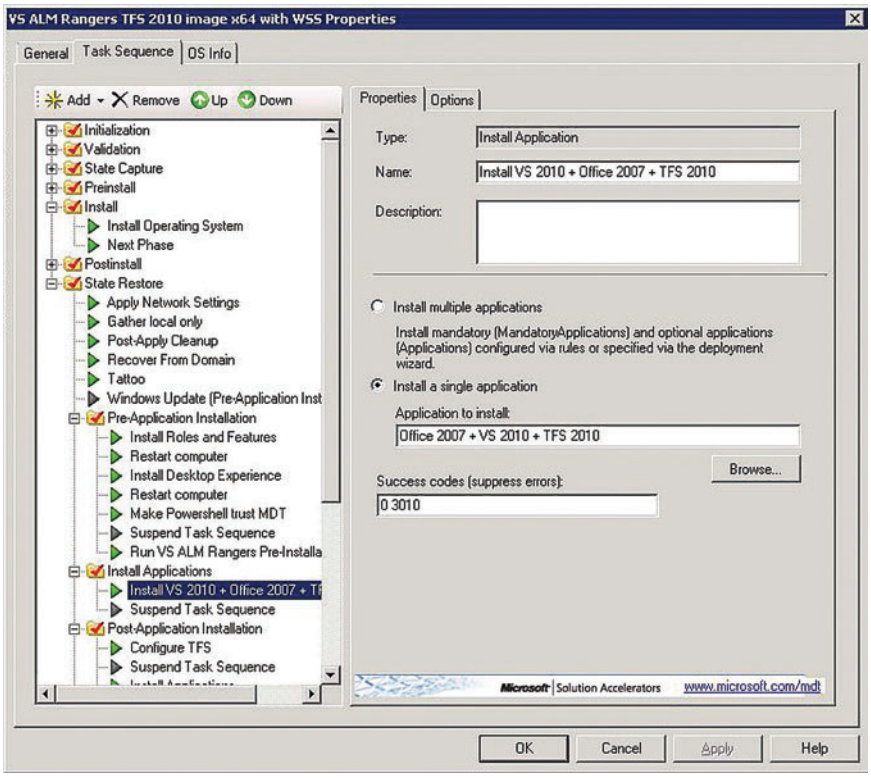


Figure 8 **Task Sequence Properties**

you can open it now by using the appropriate action on the rightmost pane.

Let's look at **Figure 6**. By expanding the VSTS Rangers Deployment Share, you can browse through the contents of the deployment share. This is a user-friendly view of the XML files in the Control directory of the deployment share directory on disk. Applications, Operating Systems and Task Sequences are our main points of interest. Whenever you make changes to the Deployment Share settings, you should select Update Deployment Share. This will regenerate the boot images that you use to start any of the task sequences.

Figure 9 **Result of Running MetricsSetup.exe /?**

Application definitions describe where the source files for an application are located in the Deployment Share and how to install the application using the Quiet install command, as shown in **Figure 7**. The command shouldn't require any user interaction to complete the installation. Sometimes this requires that you prepare the setup by creating an unattended installation script. Where necessary, the Rangers have already done that for the applications in the VM Factory.

For OS definitions, the MDT is able to extract all necessary information from the OS installation source. Usually, one set of source files will contain files for multiple versions of an OS. All the different versions will be listed under Operating Systems and can be selected for installation in Task Sequences.

The task sequences included in the VM Factory, described earlier, are located under the Task Sequences node. Viewing the properties of a task sequence lets you modify the task sequence itself. **Figure 8** shows that task sequences are divided into several phases. The important phases are the Install phase, which installs the OS, and the State Restore phase, which handles all post-OS installations. The toolbar provides options to add or remove steps from the task sequence. MDT provides several different steps. **Figure 8** also shows the properties of an Install Applications task.

Extending the VM Factory

Extending the VM Factory is also straightforward. Let's look at the basic steps to add another tool to the image. As an example, we'll add the Code Metrics Power Tool (microsoft.com/download/en/details.aspx?id=9422) to the task sequence. The tool is installed by an executable called MetricsSetup.exe. The process consists of the following steps:

1. Determine quiet installation command line for the application
2. Copy application to Deployment Share
3. Start MDT Deployment Workbench
4. Create new Application Definition
5. Add application to Task Sequence
6. Test updated Task Sequence

Step 1 is to find out how to install this tool without user interaction—an *unattended installation*. We could look at the product documentation for unattended installation information. Another method that usually gives us a hint is executing the file with /? or -? as an argument. When we run MetricsSetup.exe /? from the command line, the Usage dialog box in **Figure 9** is shown. We verify that the quiet install command line is MetricsSetup.exe /q by running this on a test computer.

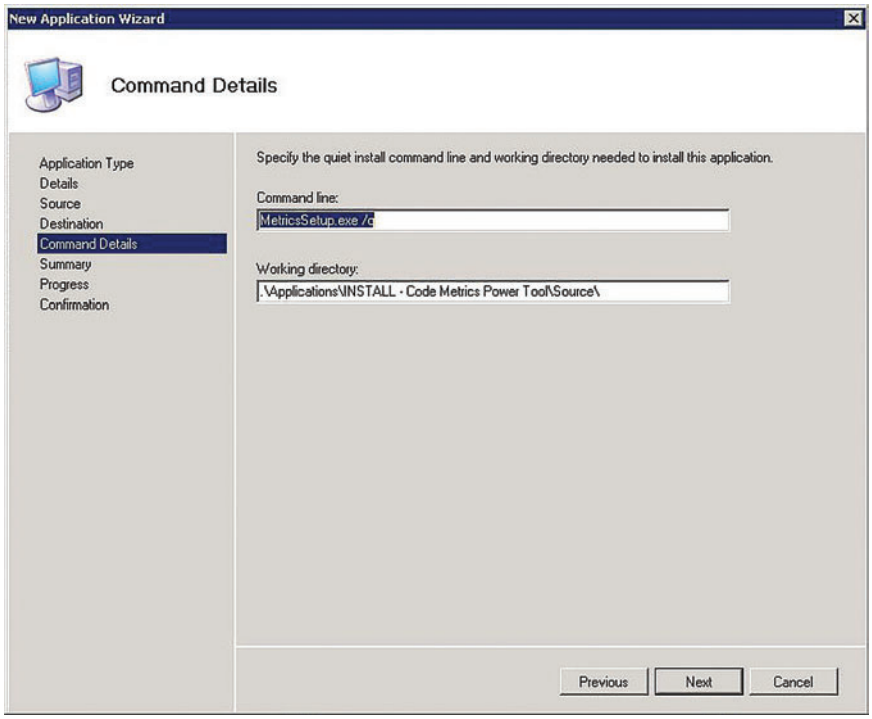


Figure 10 **New Application Wizard: Defining Command Details**

Step 2 is to move or copy the software to the correct location in the deployment share. To be consistent with the rest of the software in the deployment share, we create a directory called "C:\Deploymentshare\INSTALL – Code Metrics Power Tool\Source\" and copy the executable to the new directory.

Now that the files are in the correct location, we start the MDT Deployment Workbench (step 3) to add the application definition (step 4). From the MDT's initial view (see **Figure 6**), we find and select the Applications node in the left pane and choose New Application. A wizard helps us create the new application definition. On the first page of the wizard, we choose "Application without source files or elsewhere on the network," because we already moved the executable to the correct location.

On the next wizard page, we need to enter the Application Name that will be shown in all the interfaces and some additional, optional information that's not used in the process. The last wizard page in the

application definition process is to enter the quiet install command and the source directory (see **Figure 10**). After finishing the wizard, you'll find the new application in the application list.

To actually install the application, we need to add a step to a task sequence (step 5). In the task sequence editor, find the location where you want to add the install action. On the Add menu on the toolbar, select General | Install Application to add a new action to the task sequence. In the properties pane on the right, select "Install a single application" and click Browse to select the Code Metrics Power Tool application. Click OK to save the task sequence. We're ready for testing (step 6).

The Latest Guidance

Now you know everything you need to know to start tweaking your deployment share to your specific needs.

You can find the latest guidance and deployment share templates on CodePlex (bit.ly/aLOmxZ) and a table of contents of associated blogs, videos and other information on the Rangers blog (bit.ly/qEIRPI).

> Now you know everything you need to know to start tweaking your deployment share.

Having proven the concepts and the technical feasibility of the automation and consistency offered by the VM Factory, we're planning the following improvements:

- An easier implementation and administration of new factories
- Friendlier and controlled maintenance of existing factories, following the great principles of the Deployment Hydration initiative
- Ability to schedule factory automation
- Ability to use existing media libraries, such as MSDN downloads, instead of duplicating the media in each factory

Stay tuned as we continue to enhance this vital tool. ■

Brian Blackman *is a principal consultant with the Microsoft Services Partner ISV team, focusing on affecting ISV partners' success in engineering and in the marketplace. He is a CSM, MCSD (C++), MCTS and Visual Studio ALM Core Ranger. He spends his time writing code, creating and delivering workshops and consulting in various concentrations and all things ALM.*

Paul Meyer *is a senior application platform consultant with Microsoft Services in the Netherlands, with a strong focus on development processes and tooling. He has been involved in Visual Studio ALM Rangers projects for several years.*

Willy-Peter Schaub *is a senior program manager with the Visual Studio ALM Rangers at the Microsoft Canada Development Center. Since the mid-'80s, he's been striving for simplicity and maintainability in software engineering. His blog is at blogs.msdn.com/b/willy-peter_schaub and he's on Twitter at twitter.com\wpschaub.*

Thanks to the following technical experts for reviewing this article: *Vladimir Gusarov, Bill Heys, Bijan Javidi, Zayd Kara, Vijay Machiraju, Robert MacLean, Rui Melo and Patricia Wagner*

JAMES MCCAFFREY

Tabu Algorithms and Maximum Clique

In this month's column, I'll present an advanced solution to the graph maximum clique problem. The solution uses what's called a tabu algorithm, and I'll discuss how to design and test these algorithms. The idea of the maximum clique problem is to find the largest group of nodes in a graph that are all connected to one another. Take a look at the simple graph in **Figure 1**. The graph has nine nodes and 13 edges. The graph is unweighted (there are no priorities associated with the edges) and undirected (all edges are bidirectional). Nodes 2, 4 and 5 form a clique of size three. The maximum clique is the node set 0, 1, 3 and 4, which forms a clique of size four.

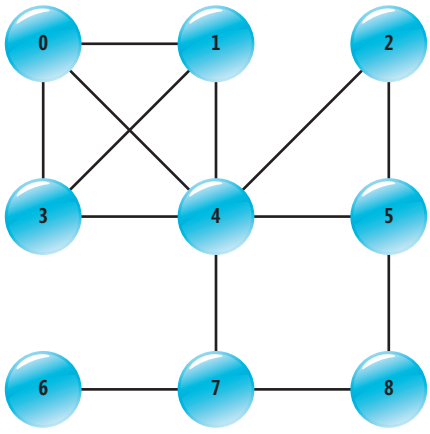


Figure 1 **Graph for a Tabu Maximum Clique Algorithm**

The maximum clique problem is interesting for several reasons. Although it's not apparent from the simple graph in **Figure 1**, the maximum clique problem is one of the most challenging in computer science. It arises in many important practical problems, such as social network communication analysis, where nodes represent people and edges represent messages or relationships. And the problem uses techniques such as greedy algorithms and tabu algorithms, which are important advanced programming techniques. Having a solution to the maximum clique problem in your personal library can be a useful practical tool, and understanding the algorithm used can add new techniques to your skill set.

This is the third column in a series that uses the maximum clique problem to illustrate advanced coding and testing techniques, but this column can be read without direct reference to the previous two. In my October column, "Graph Structures and Maximum Clique" (msdn.microsoft.com/magazine/hh456397), I described how to code an efficient data structure to hold a graph data structure in memory. In my November column, "Greedy Algorithms and Maximum Clique" (msdn.microsoft.com/magazine/hh547104), I described how a relatively simple algorithm can be used to find a solution to the difficult maximum clique problem.

In informal terms, a greedy algorithm is an algorithm that starts with a simple, incomplete solution to a difficult problem and then iteratively looks for the best way to improve the solution. The process is repeated until some stopping condition is reached. However, greedy algorithms typically have a weakness: They'll repeatedly generate the same solution over and over. Tabu algorithms are designed to deal with this weakness. The word *tabu*, sometimes spelled *taboo*, means forbidden. In simple terms, tabu algorithms maintain a list of forbidden data. The processing part of the algorithm isn't permitted to use tabu data until some prohibit time has passed. Simple forms of tabu algorithms use a fixed prohibit time. Advanced tabu algorithms adapt the prohibit time dynamically while the algorithm executes.

Figure 2 illustrates the important ideas of a tabu algorithm applied to the maximum clique problem and shows you where I'm headed in this column.

I have a console application that begins by loading into memory the graph corresponding to the one shown in **Figure 1**. The file uses a standard graph file format called DIMACS. Designing and coding an efficient graph data structure is a significant problem in itself. I explained the graph data structure code in my October column.

The algorithm begins by selecting a single node at random and initializing a clique with that node. The algorithm then iteratively tries to find and add the best node that will increase the size of the clique. I'll explain what best node means later. If the algorithm can't find a node to add, it attempts to find the best node to drop from the clique. Behind the scenes, the algorithm remembers the last time each node was moved into the current solution clique or moved out from the clique. The algorithm uses this information to prohibit adding or dropping recently used nodes for a certain amount of time. This is the tabu part of the algorithm.

The algorithm restarts itself every so often when no progress has been made in finding a better (larger) clique. The algorithm silently stores the solutions (cliques) it has previously seen. The algorithm uses that solution history information to dynamically adapt the prohibit time. If the algorithm keeps encountering the same solutions, it increases the prohibit time to discourage recently used nodes from being used. If the algorithm doesn't see the same solutions, it decreases the prohibit time so there's a larger pool of nodes from which to choose. This is the adaptive part of the tabu algorithm.

Code download available at code.msdn.microsoft.com/mag201112TestRun.

In most situations where a greedy algorithm is used, the optimal solution is not known, so one or more stopping conditions must be specified. When the algorithm hits a stopping condition, the best solution found is displayed.

In the sections that follow, I'll walk you through the code that produced the screenshot in **Figure 2**. The complete source code is available at code.msdn.microsoft.com/mag201112TestRun. This column assumes you have intermediate-level programming skills with a C-family language or the Visual Basic .NET language. I use C#, but I've written the code so that you'll be able to refactor to another language such as F# or Python without too much difficulty if you want.

Overall Program Structure

The overall structure of the program shown in **Figure 2** is presented in **Figure 3**. I decided to place all code in a single console application using static methods for simplicity, but you might want to modularize parts of the code into class libraries and use an object-oriented approach. The program is less complicated than you might suspect in looking at **Figure 3** because most of the methods are short helper methods.

The program has two high-level classes, and each of those classes has a helper subclass. Class TabuMaxCliqueProgram contains the Main method and all algorithm logic, along with subclass CliqueInfo, which is used to maintain a history of previously seen clique solutions. Class MyGraph encapsulates an efficient graph representation and contains subclass BitMatrix, which is used to store node adjacency information in a condensed form. A class scope Random object named random is used to initialize the clique to a random node and to break ties when there are multiple best nodes to add or drop.

> The idea of the maximum clique problem is to find the largest group of nodes in a graph that are all connected to one another.

With some WriteLine statements and try-catch code removed, the Main method is:

```
Console.WriteLine("\nBegin tabu algorithm maximum clique demo\n");
string graphFile = "..\\..\\DimacsGraph.clq";
MyGraph graph = new MyGraph(graphFile, "DIMACS");
int maxTime = 50;
int targetCliqueSize = graph.NumberNodes;
List<int> maxClique = FindMaxClique(graph, maxTime, targetCliqueSize);
Console.WriteLine("\nSize of best clique found = " + maxClique.Count);
Console.WriteLine(ListAsString(maxClique));
```

The graph is represented as a program-defined MyGraph object. The graph is loaded from an external text file that uses the DIMACS file format. The key method of MyGraph is AreAdjacent, which returns true if two nodes are connected. I set maxTime to 50 iterations to establish an absolute stopping condition for the greedy algorithm. This is artificially small. In a real maximum clique problem, the max time is typically in the range of 1,000 to 100,000. I set targetClique size to establish a second stopping condition; if a clique is found that has the same number of nodes as there are in the graph, the maximum clique must have been found. Most of the work is done by the FindMaxClique method, which uses a greedy, adaptive tabu algorithm to search for the largest possible clique. The helper method ListAsString simply creates a string representation of a List<int> object.

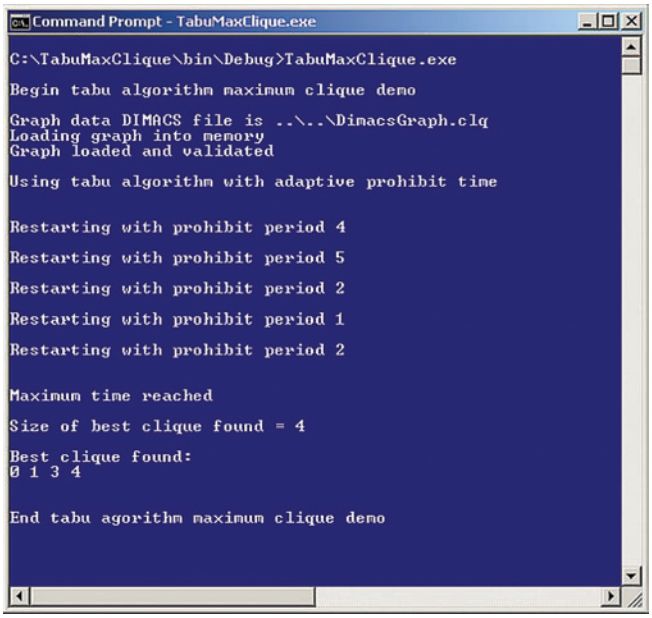


Figure 2 **Tabu Maximum Clique Demo Run**

The FindMaxClique Method

The FindMaxClique method calls several helper methods, which I'll describe shortly. In high-level pseudocode, the FindMaxClique algorithm is given in **Figure 4**. The pseudocode leaves out several important details for clarity but presents the main points of the algorithm.

The FindMaxClique method definition begins:

```
static List<int> FindMaxClique(MyGraph graph, int maxTime,
  int targetCliqueSize)
{
  List<int> clique = new List<int>();
  random = new Random(1);
  int time = 0;
  int timeBestClique = 0;
  int timeRestart = 0;
  int nodeToAdd = -1;
  int nodeToDrop = -1;
...
```

The local clique object is the current solution clique. The random object is instantiated with an arbitrary seed value of 1. The time variable is the counter for the algorithm's main processing loop. The timeBestClique and timeRestart variables are used to determine if there has been a lack of progress so the algorithm can restart itself. Next:

```
int prohibitPeriod = 1;
int timeProhibitChanged = 0;
int[] lastMoved = new int[graph.NumberNodes];
for (int i = 0; i < lastMoved.Length; ++i) {
  lastMoved[i] = int.MinValue;
}
Hashtable history = new Hashtable();
...
```

The prohibit period is initialized to 1. This means that—initially, at least—after a node has been used by the algorithm, it can't be used for one time iteration. If I had used a value of 0, the effect would have been to have no prohibit time at all. Most tabu algorithms use a fixed value for the prohibit time, but the problem with that approach is that research has shown the best value for a fixed prohibit time varies from problem to problem. The adaptive approach I present updates the prohibit time while the algorithm runs, based on the history of previous solutions. I call this technique adaptive tabu, but some research papers call the technique reactive or dynamic.

The lastMoved array is used to determine if a node is allowed or prohibited at any given time. The index of the array represents a node, and the corresponding array value is the time the node was last moved (added or dropped). By using the lastMoved array, I eliminate the need to explicitly create a list of allowed nodes (or, equivalently, prohibited nodes). I initialize all cells of lastMoved to int.MinValue so I can later determine if a node has never been used.

The history Hashtable object holds a collection of previously seen solution cliques. Each item in the hash table is a CliqueInfo object, which I'll describe in detail later. I could've used a generic Dictionary object instead of the non-generic Hashtable object. FindMaxClique continues:

```
int randomNode = random.Next(0, graph.NumberNodes);
clique.Add(randomNode);
List<int> bestClique = new List<int>();
bestClique.AddRange(clique);
int bestSize = bestClique.Count;
timeBestClique = time;
...
```

I initialize the current solution clique to a random node from the graph. I instantiate a bestClique list to keep track of the best (largest) clique solution found by the algorithm. Next comes:

```
List<int> possibleAdd = MakePossibleAdd(graph, clique);
List<int> oneMissing = MakeOneMissing(graph, clique);
...
```

The MakePossibleAdd method constructs a list of all nodes that can be added to the current clique to increase the size of the clique by 1. In other words, the method creates a list of nodes that aren't in the clique but are connected to every node in the clique. This possibleAdd list could have a count of 0 or could contain prohibited nodes. MakePossibleAdd calls helper method FormsALargerClique.

The MakeOneMissing method constructs a list of all nodes that are connected to all but exactly one of the nodes in the current clique. It's not obvious, but it turns out that maintaining such a list creates a clever way to determine which node in the current clique is the best node to drop, as I'll explain later. The main processing loop begins:

```
while (time < maxTime && bestSize < targetCliqueSize)
{
  ++time;
  bool cliqueChanged = false;
...
```

The main processing loop will terminate when the maximum number of iterations is reached or if a best solution with the target clique size is found. I use the cliqueChanged variable to control the branching logic between adding and dropping nodes, as you'll see. The logic to add an allowed node is shown in **Figure 5**.

The algorithm checks to see if there are any nodes that will increase the size of the current clique. If so, method SelectAllowedNodes creates a list of nodes that are allowed—that is, not tabu—from the list of nodes that can be added.

The key line in SelectAllowedNodes is:

```
if (time > lastMoved[currNode] + prohibitPeriod)
  result.Add(currNode); // Allowed
```

The currNode is one of the nodes in the possibleAdd list. The logic checks to see if enough time has elaspsed since the node was last used so that the node is past the prohibit period. If so, the node is added to the list of allowedAdd nodes.

Now, if there are any nodes that are allowed and will increase the size of the clique, method GetNodeToAdd determines the best node to add to the clique. The best node is the node from the allowedAdd list that's most connected to nodes in the possibleAdd list. The idea is that you want to add a node that will give you the most chances of finding a node to add in the next iteration of the algorithm. There could be

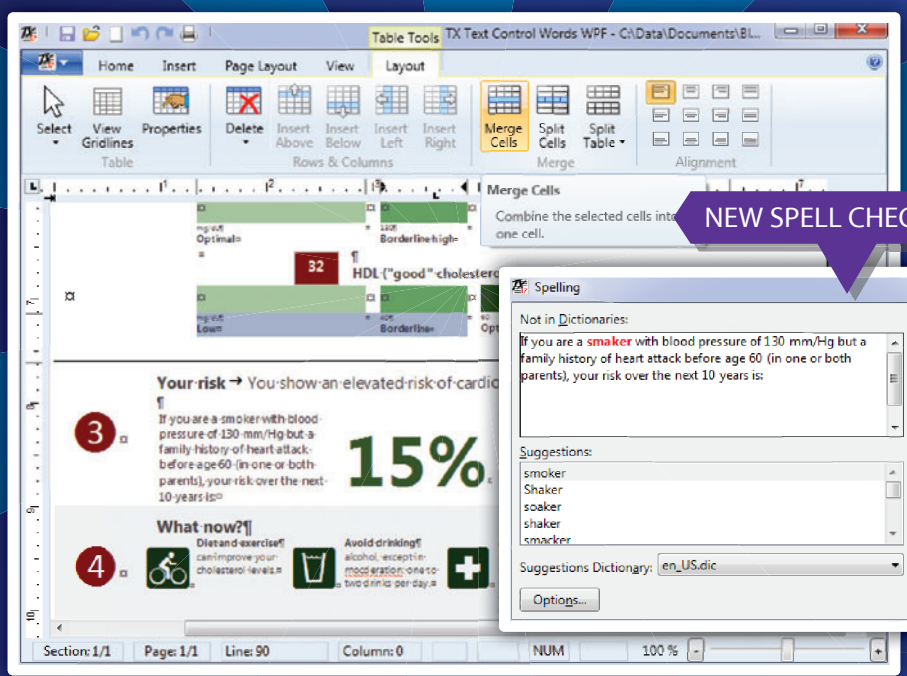
Figure 3 **Overall Program Structure**

```
using System;
using System.Collections.Generic; // List<int>
using System.Text; // StringBuilder
using System.IO; // FileStream
using System.Collections; // Hashtable

namespace TabuMaxClique
{
  class TabuMaxCliqueProgram
  {
    static Random random = null;

    static void Main(string[] args) { ... }

    static List<int> FindMaxClique(MyGraph graph, int maxTime,
      int targetCliqueSize) { ...  }

    static List<int> MakePossibleAdd(MyGraph graph,
      List<int> clique) { ... }

    static bool FormsALargerClique(MyGraph graph,
      List<int> clique, int node) { ... }

    static int GetNodeToAdd(MyGraph graph,
      List<int> allowedAdd, List<int> possibleAdd) { ... }

    static int GetNodeToDrop(MyGraph graph, List<int> clique,
      List<int> oneMissing) { ... }

    static List<int> MakeOneMissing(MyGraph graph,
      List<int> clique) { ... }

    static List<int> SelectAllowedNodes(List<int> listOfNodes,
      int time, int prohibitPeriod, int[] lastMoved) { ... }

    static int UpdateProhibitPeriod(MyGraph graph, List<int> clique,
      int bestSize, Hashtable history, int time, int prohibitPeriod,
      ref int timeProhibitChanged) { ... }

    static string ListAsString(List<int> list) { ... }

    private class CliqueInfo
    {
      // ...
    }
  }

  public class MyGraph
  {
    // ...
    private class BitMatrix
    {
      // ...
    }
  }
} // ns
```

WINDOWS FORMS | WPF | ASP.NET | ACTIVEX

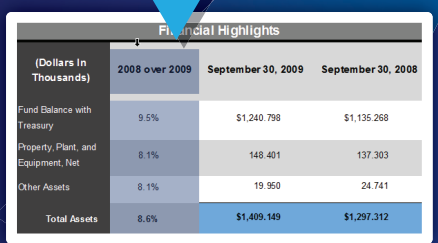
WORD PROCESSING COMPONENTS

VERSION 17.0 RELEASED

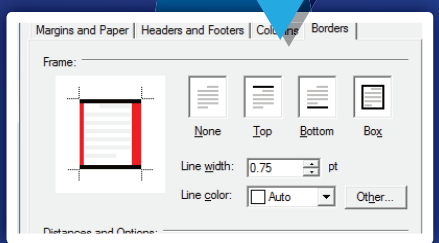


NEW SPELL CHECK COMPONENT

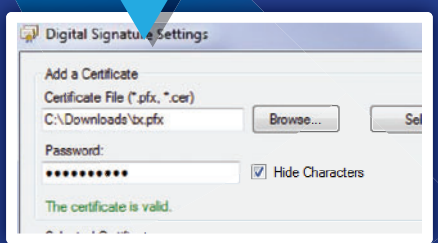
CELL MERGING, COL SELECTION



PAGE BORDERS



DIGITAL SIGNATURES IN PDF



TX TEXT CONTROL® word processing components

Microsoft® Visual Studio® PARTNER

Word Processing Components for Windows Forms, WPF & ASP.NET

WWW.TEXTCONTROL.COM

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

Figure 4 **Adaptive Tabu Maximum Clique Algorithm**

```
initialize clique with a single node
get list of candidate nodes to add
loop until stop condition met
  if there are nodes which can be added
    get list of allowed nodes
    if there are allowed nodes to add
      find best node to add and add to clique
      mark added node as recently used
      record current clique
 else
   get list of allowed nodes to drop
   if there are allowed nodes which can be dropped
     find best node to drop and drop from clique
     mark dropped node as recently used
     record current clique
 else
   select a random node to drop and drop from clique
   mark dropped node as recently used
   record current clique

 if lack of progress
   restart algorithm

  update list of candidate nodes
  update prohibit time
end loop
return largest clique found
```

multiple nodes in the allowedAdd list that are tied as most connected to the nodes in possibleAdd. If so, one of the tied nodes is selected at random. After adding the best node, the current clique solution is sorted for reasons that I'll explain later. The code to drop a node is:

```
if (cliqueChanged == false) {
  if (clique.Count > 0) {
    List<int> allowedInClique = SelectAllowedNodes(clique, time,
      prohibitPeriod, lastMoved);
    if (allowedInClique.Count > 0) {
      nodeToDrop = GetNodeToDrop(graph, clique, oneMissing);
      clique.Remove(nodeToDrop);
      lastMoved[nodeToDrop] = time;
      clique.Sort();  cliqueChanged = true;
    }
  }
}
...
```

If the algorithm can't add a node, it attempts to drop an allowed node in the hope that backtracking will yield a solution that will allow a new node to be added in the next iteration. The list of nodes in the current clique is filtered by the SelectAllowedNodes method to get nodes that are not tabu. The GetNodeToDrop picks the best of these nodes to drop using the list of oneMissing nodes.

The idea is to drop the allowed node in the current clique, which will result in the largest increase in the list of possibleAdd nodes. One way to do this is to test each node in the allowed list by actually removing it from the current clique and then computing the size of the resulting possibleAdd list. But there's a much more efficient approach that uses a list of nodes that are connected to all but exactly one of the nodes in the current clique. Using the oneMissing list of nodes, the list can be used as follows. Scan through each node in the allowed clique nodes and count how many nodes in the oneMissing list are *not* connected to the clique node. The node in the allowed clique list that's *least* connected to the nodes in the oneMissing list is the best node to drop. After dropping this least-connected node, all the nodes in the oneMissing list that weren't connected to the dropped node will now be fully connected to the remaining nodes in the clique and therefore become new possibleAdd nodes.

At this point in the logic, it might not have been possible to add an allowed node or drop an allowed node. To unstick itself, the algorithm drops a random node from the current clique:

```
if (cliqueChanged == false) {
  if (clique.Count > 0) {
    nodeToDrop = clique[random.Next(0, clique.Count)];
    clique.Remove(nodeToDrop);
    lastMoved[nodeToDrop] = time;
    clique.Sort();
    cliqueChanged = true;
  }
}
...
```

Next, the algorithm checks to see if there has been a lack of progress and, if so, restarts itself:

```
int restart = 100 * bestSize;
if (time - timeBestClique > restart && time - timeRestart > restart) {
  timeRestart = time;
  prohibitPeriod = 1;
  timeProhibitChanged = time;
  history.Clear();
...
```

The restart variable is the number of iterations to allow where there's no improvement or since the last restart. Here I use a value of 100 times the size of the current best solution. The value to use for the restart interval isn't well understood and you might want to try alternatives. (I used a dummy value of 2 to produce more restarts for the screenshot in **Figure 2**). The value I use has worked well for me in practice, however. If there is a restart, the algorithm resets the prohibit time and clears the history hash table holding solution cliques that have been seen. Note that the algorithm hasn't yet updated the history hash table. The restart code doesn't, however, reset the lastVisited array, which stores information about when nodes were last added to or dropped from the solution clique. Next comes:

```
int seedNode = -1;
List<int> temp = new List<int>();
for (int i = 0; i < lastMoved.Length; ++i) {
  if (lastMoved[i] == int.MinValue) temp.Add(i);
}
if (temp.Count > 0)
  seedNode = temp[random.Next(0, temp.Count)];
else
  seedNode = random.Next(0, graph.NumberNodes);
...
```

The algorithm attempts to reseed the solution clique with a node that has never been used before. If there are several unused nodes, one is selected at random. If there are no unused nodes, a random node is selected. Instead of using a random node, an unexplored

Figure 5 **The Logic to Add an Allowed Node**

```
if (possibleAdd.Count > 0) {
  List<int> allowedAdd = SelectAllowedNodes(possibleAdd, time,
    prohibitPeriod, lastMoved);
  if (allowedAdd.Count > 0) {
    nodeToAdd = GetNodeToAdd(graph, allowedAdd, possibleAdd);
    clique.Add(nodeToAdd);
    lastMoved[nodeToAdd] = time;
    clique.Sort();  cliqueChanged = true;
    if (clique.Count > bestSize) {
      bestSize = clique.Count;
      bestClique.Clear();
      bestClique.AddRange(clique);
      timeBestClique = time;
    }
  }
}
...
```

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

YOUR MAP TO THE .NET DEVELOPMENT PLATFORM

WHAT YOU LEARN IN VEGAS WON'T STAY IN VEGAS

Intense Take-Home Training for Developers, Software Architects and Designers

Las Vegas | **March 26-30** | Mirage Resort and Casino

SUPPORTED BY

Microsoft | Microsoft Visual Studio | msdn | Visual Studio MAGAZINE

PRODUCED BY

1105 MEDIA

Coding, Casinos and More!

If you're looking for unbiased, hard-hitting and practical .NET Developer training, look no further than Visual Studio Live! Las Vegas.

Code with the pros and learn how to maximize the development capabilities of Visual Studio and .NET during five action-packed days of 60+ workshops and sessions led by expert instructors.

Topics will include:

- Windows 8, WinRT and Metro
- Silverlight / WPF
- Web
- Visual Studio 2010+ / .NET 4.0+
- SharePoint
- Cloud Computing
- Data Management
- HTML5
- Windows Phone 7
- Cross Platform Mobile with MonoTouch and MonoDroid

Be a more valuable part of your company's development team.

Register Today and Save $400!

Use Promo Code DECAD

vslive.com/lasvegas

Use Promo Code DECAD and Save $400!

Figure 6 **The CliqueInfo Class**

```
private class CliqueInfo
{
  private List<int> clique;
  private int lastSeen;

  public CliqueInfo(List<int> clique, int lastSeen)
  {
    this.clique = new List<int>();
    this.clique.AddRange(clique);
    this.lastSeen = lastSeen;
  }

  public int LastSeen
  {
    get { return this.lastSeen; }
    set { this.lastSeen = value; }
  }

  public override int GetHashCode()
  {
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < clique.Count; ++i) {
      sb.Append(clique[i]);
      sb.Append(" ");
    }
    string s = sb.ToString();
    return s.GetHashCode();
  }

  public override string ToString()
  {
    string s = "";
    for (int i = 0; i < clique.Count; ++i)
      s += clique[i] + " ";
      return s;
  }
}
```

alternative is to select the node that was least recently moved. The restart code finishes with the following:

```
...
  clique.Clear();
  clique.Add(seedNode);
}
```

The main processing loop and FindMaxClique wrap up like so:

```
...
    possibleAdd = MakePossibleAdd(graph, clique);
    oneMissing = MakeOneMissing(graph, clique);
    prohibitPeriod = UpdateProhibitPeriod(graph, clique, bestSize,
      history, time, prohibitPeriod, ref timeProhibitChanged);
  } // Main processing loop
  return bestClique;
}
```

The possibleAdd and oneMissing lists are regenerated. An alternative is to maintain auxiliary data structures and update these two lists instead of recreating them from scratch. The UpdateProhibitPeriod method is the key component of the adaptive part of the tabu algorithm and I'll describe it shortly.

Remembering Previous Solutions

Method UpdateProhibitPeriod uses a hash table of previously seen solutions to dynamically increase or decrease the prohibit time. Recall that a solution is a clique that's a List<int> of nodes that are all connected to one another. But I also need to store the time when a solution was last seen. Therefore, I encapsulate a clique solution and the last time it was seen in a CliqueInfo class as listed in **Figure 6**.

Because a CliqueInfo item will be added to the solution history Hashtable object, I need a custom GetHashCode hash function. Writing custom hash functions is surprisingly tricky, and a thorough discussion of all the issues involved would require an entire column. In this situation, I use the simplest—but not the most efficient—approach. I create a string representation of the nodes in the clique field, separated by spaces, and then use the built-in String.GetHashCode. For example, if a clique contained nodes {3 5 8}, I generate "3 5 8 " (with a trailing space) and then generate a hash code from that string. Recall that cliques are always maintained in a sorted order, so it wouldn't be possible to have one clique {3 5 8} and another clique {8 3 5}. I place spaces between the nodes so that clique {3 5 8} is distinguished from clique {3 58}.

Updating the Prohibit Period

Method UpdateProhibitPeriod adaptively increases or decreases the prohibit time based on previously seen clique solutions. The method begins:

```
static int UpdateProhibitPeriod(MyGraph graph, List<int> clique,
  int bestSize, Hashtable history, int time, int prohibitPeriod,
  ref int timeProhibitChanged)
{
  int result = prohibitPeriod;
  CliqueInfo cliqueInfo = new CliqueInfo(clique, time);
. . .
```

The method will return a prohibit time that could possibly be the same as the current prohibit time. A CliqueInfo object holding the current clique and current time are instantiated, as shown here:

```
if (history.Contains(cliqueInfo.GetHashCode())) {
  CliqueInfo ci = (CliqueInfo)history[cliqueInfo.GetHashCode
  int intervalSinceLastVisit = time - ci.LastSeen;
  ci.LastSeen = time;
  if (intervalSinceLastVisit < 2 * graph.NumberNodes - 1) {
    timeProhibitChanged = time;
    if (prohibitPeriod + 1 < 2 * bestSize) return prohibitPeriod + 1;
    else return 2 * bestSize;
  }
}
else history.Add(cliqueInfo.GetHashCode(), cliqueInfo);
...
```

The code checks to see if the current clique solution, in the form of a CliqueInfo object, has been seen before—that is, is the clique in the history hash table? If the current clique has been seen before, the algorithm computes how long it has been since the clique was seen. If this interval is short enough, the prohibit time is incremented (subject to an upper limit). The idea is that because it hasn't been very long since the current clique was seen, the algorithm is generating duplicate solutions. If the prohibit time is increased, there will be more tabu nodes and therefore fewer allowed nodes, reducing the chances of generating duplicate clique solutions. If the current clique solution hasn't been seen before, it's added to the history hash table.

The "short enough" interval is twice the number of nodes in the graph, less one. This is used to determine when to increment the prohibit time. The best value to use here is another open question in maximum clique research. The "upper limit" for the prohibit time is twice the size of the current best known solution. The best upper-limit value is, as you can probably guess by now, another open research question.

At this point, either the current clique hasn't been seen before, or the clique has been seen before but the interval required to increment the prohibit time wasn't small enough. So the algorithm now checks to see if the prohibit period can be decremented, which will decrease the number of tabu nodes and increase the number of allowed nodes, which in turn gives the algorithm more nodes to add or drop:

```
...
  if (time - timeProhibitChanged > 10 * bestSize) {
    timeProhibitChanged = time;
    if (prohibitPeriod - 1 > 1)
      return prohibitPeriod - 1;
    else
      return 1;
  }
  else {
    return result; // no change
  }
} // UpdateProhibitTime
```

Rather than explicitly checking to see if it has been a "relatively long" time since the current clique was seen, the algorithm indirectly checks how long it has been since the current clique was seen by examining the time when the prohibit period was last changed. Once again, the best value for "relatively long" isn't clearly understood. I use a value 10 times the size of the current best solution. Notice that the prohibit time can't drop below 1.

More Research

Research results on the maximum clique problem suggest that a greedy approach with an adaptive tabu feature gives the best overall results when both performance and solution quality are taken into account. DIMACS is a research organization that created a set of difficult graph clique benchmark problems. I ran the code presented here against one particularly difficult DIMACS problem (C2000.9) and the code quickly (in less than two seconds) found a clique with size 76 that's within 1.5 percent of the size of the best-known solution of 77.

At several points in this column, I've mentioned research results on the maximum clique problem. If you're interested in this research, I recommend you search the Web for academic papers written by the following: R. Battiti et al., W. Pullan et al. and K. Katayama et al. Several papers by these three authors and their colleagues were my primary references.

A promising unexplored area for the maximum clique algorithm presented here is to incorporate some form of what is called *plateau search*. Recall that the maximum clique algorithm first attempts to add a non-tabu node to the current clique solution. Then, if that isn't possible, the algorithm drops a node from the current clique solution. The idea of plateau search is to find one node in the current clique solution that can be swapped with a node not in the clique. Although this doesn't increase the size of the current clique, it doesn't decrease the size of the clique, either. ■

DR. JAMES MCCAFFREY *works for Volt Information Sciences Inc., where he manages technical training for software engineers working at the Microsoft Redmond, Wash., campus. He's worked on several Microsoft products, including Internet Explorer and MSN Search. Dr. McCaffrey is the author of ".NET Test Automation Recipes" (Apress, 2006), and can be reached at jammc@microsoft.com.*

THANKS to the following technical experts for reviewing this article: *Paul Koch, Dan Liebling, Ann Loomis Thompson and Shane Williams*

TED NEWARD

Parser Combinators

With the conclusion of the multiparadigmatic series, it seemed time to venture out into new ground. As fate would have it, however, some client work recently left me with interesting material that bears discussion, relates to the design of software, acts as another example of the commonality/variability analysis at the core of the multiparadigmatic series, and … well, in the end, it's just really cool.

The Problem

The client I have is neck-deep in the neuro-optical scientific world and asked me to work with him on a new project designed to make conducting experiments on optical tissue easier. Specifically, I'm working on a software system to control a microscope rig that will drive various stimulus devices (LEDs, lights and so on) that will trigger responses from the optical tissue, and then capture the results measured by hardware watching the optical tissue.

If it all sounds vaguely Matrix-y to you, you're not entirely alone. When I first heard about this project, my reaction was simultaneously, "Oh, wow, that's cool!" and, "Oh, wait, I just threw up in my mouth a little."

At any rate, one of the key things about the rig is that it will have a fairly complex configuration associated with each experiment run, and that led us to contemplate how to specify that configuration. On the one hand, it seemed an obvious problem for an XML file. However, the people running the rig aren't going to be computer programmers, but rather scientists and lab assistants, so it seemed a little heavy-handed to expect them to write well-formed XML files (and get it right at every turn). The thought of producing some kind of GUI-based configuration system struck us as highly over-engineered, particularly as it would quickly turn into discussions of how best to capture open-ended kinds of data.

In the end, it seemed more appropriate to give them a custom configuration format, which meant tons of parsing text on my part. (To some, this would imply that I'm building a DSL; this is a debate best left to philosophers and others involved in the serious task of alcohol consumption.) Fortunately, solutions abound in this space.

Thoughts

A parser serves two interesting and useful purposes: converting text into some other, more meaningful form, and verifying/validating the text follows a certain structure (which typically is part of helping to convert it into a more meaningful form). So, for example, a phone number, which is at its heart just a sequence of numbers, still has a structure to it that requires verification. That format varies from continent to continent, but the numbers are still numbers. In fact, a phone number is a great example of a case where the "more meaningful form" isn't an integer value—the digits aren't an integer value, they're a symbolic representation that's usually better represented as a domain type. (Treating them as "just" a number makes it difficult to extract the country code or area code, for example.)

If a phone number is made up of digits, and so are numbers (salaries, employee IDs and so on), then there's going to be some duplication in code where we parse and verify digits, unless we somehow extend a parser. This implies, then, that we'd like whatever parser we build to be open-ended, allowing somebody using the parser/library to extend it in different ways (Canadian postal codes, for example) without having to modify the source itself. This is known as the "open-closed principle": Software entities should be open for extension, but closed for modification.

The client I have is neck-deep in the neuro-optical scientific world and asked me to work with him on a new project designed to make conducting experiments on optical tissue easier.

Solution: Generative Metaprogramming

One solution is the traditional "lex/yacc" approach, known more formally as a "parser generator." This entails specifying the syntax for the configuration file in an abstract format—usually some variation on the Backus-Naur form (BNF) syntax/grammar used to describe formal grammar, such as what most programming languages use—then running a tool to generate code to pick the string input apart and yield some kind of tree of structures or objects as a result. Generally, this involved process is split into two steps, "lexing" and "parsing," in which the lexer first transforms the string input into tokens, validating that the characters do in fact form legitimate tokens along the way. Then the parser takes the tokens and validates that the tokens are appearing in the appropriate order and contain

Does your Team do more than just track bugs?

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

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

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

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

Native Smart Card Login Support including Government and DOD

New in Team 2.11

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

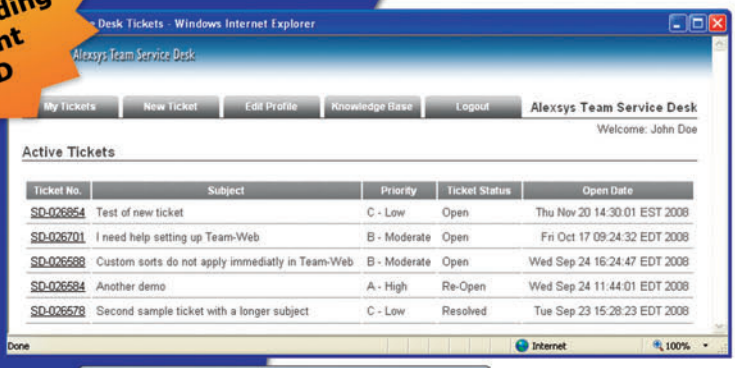
Service Desk Features

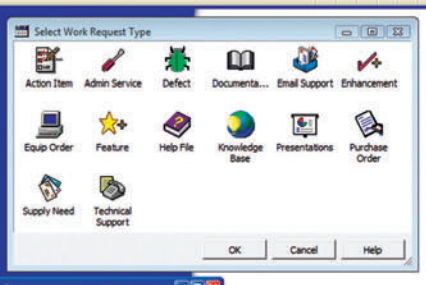
- Fully Secure
- Unlimited Users Self Registered or Active Directory
- Integrated into Your Web Site
- Fast/AJAX Dynamic Content
- Unlimited Service Desks
- Visual Service Desk Builder

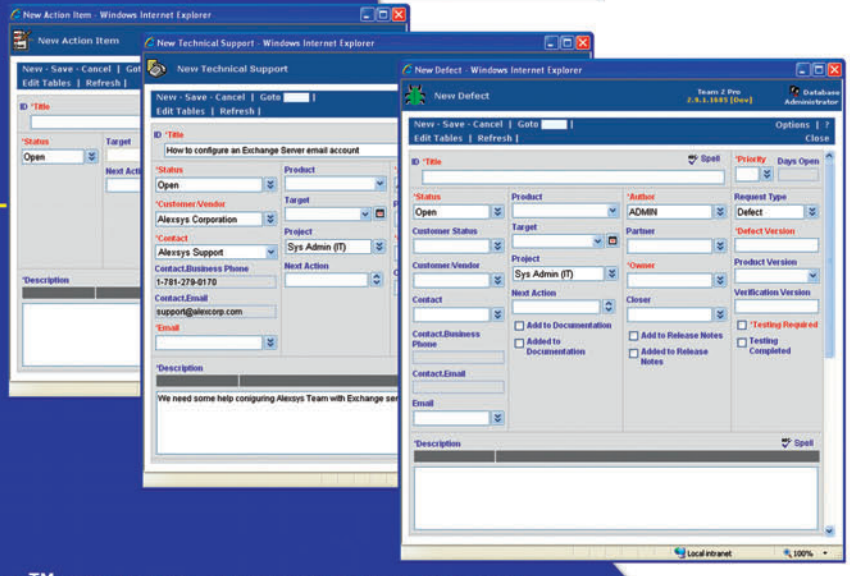
Team 2 Features

- Windows and Web Clients
- Multiple Work Request Forms
- Customizable Database
- Point and Click Workflows
- Role Based Security
- Clear Text Database
- Project Trees
- Time Recording
- Notifications and Escalations
- Outlook Integration

ZDNet Editors' Pick







Free Trial and Single User FreePack™ available at www.alexcorp.com.
FreePack™ includes a free single user Team Pro and Team-Web license.
Need more help? Give us a call at 1-888-880-ALEX (2539).

Team 2 works with its own standard database, while Team Pro works with Microsoft SQL, MySQL, and Oracle Servers.
Team 2 works with Windows 7/2008/2003/Vista/XP.
Team-Web works with Internet Explorer, Firefox, Netscape, Safari, and Chrome.

the appropriate values, and so on, usually transforming the tokens into some kind of abstract tree structure for further analysis.

The problems with parser generators are the same for any generative metaprogramming approach: The code generated will need to be regenerated in the event that the syntax changes. But more importantly for this kind of scenario, the code generated will be computer-generated, with all the wonderful variable naming that comes with computer-generated code (anybody ready to stand up for variables such as "integer431" and "string$$xyz"?), thus difficult to debug.

Solution: Functional

In a certain kind of light, parsing is fundamentally functional: It takes input, performs some kind of operation on it and produces output as a result. The critical insight, it turns out, is that a parser can be created out of lots of little parsers, each of which parses a tiny bit of the string input, then returns a token and another function to parse the next little bit of string input. These techniques, which I believe were introduced in Haskell, are formally known as *parser combinators*, and they turn out to be an elegant solution to "mid-size" parsing problems—parsers that aren't necessarily as complex as a programming language would require, but something beyond what String.Split (or a hacked-up series of regex scans) can do.

In the case of parser combinators, the open-for-extension requirement is achieved by creating small functions, then using functional techniques to "combine" them into larger functions (which is where we get the name "combinators"). Larger parsers can be composed by anyone with sufficient skill to understand function composition. This technique is a general one that bears exploration, but I'll save that for a future column.

As it turns out, there are several parser combinator libraries available for the Microsoft .NET Framework, many of them based on the Parsec module written in Haskell that sort of set the standard for parser combinatory libraries. Two such libraries are FParsec, written for F#, and Sprache, written for C#. Each is open source and relatively well-documented, such that they serve the dual purpose of being both useful out of the box and as a model from which to study design ideas. I'll also leave FParsec for a future column.

"Sprache Sie Parsing?"

Sprache, available at code.google.com/p/sprache, describes itself as a "simple, lightweight library for constructing parsers directly in C# code," which "doesn't compete with 'industrial strength' language workbenches. It fits somewhere in between regular expressions and a full-featured toolset such as ANTLR." (ANTLR is a parser generator, fitting into the Generative Metaprogramming category, like lex/yacc.)

Getting started with Sprache is straightforward: Download the code, build the project, then copy the Sprache.dll assembly into your project's dependency directory and add the reference to the project. From here, all the parser definition work is done by declaring Sprache.Parser instances and combining them in particular ways to create Sprache.Parser instances, which in turn may, if desired (and it usually is), return domain objects containing some or all of the parsed values.

Figure 1 **Parsing a Phone Number**

```
[TestMethod]
public void ParseJustThreeNumbers()
{
  string result = threeNumberParser.Parse("123");
  Assert.AreEqual("123", result);
}
[TestMethod]
public void ParseJustThreeNumbersOutOfMore()
{
  string result = threeNumberParser.Parse("12345678");
  Assert.AreEqual("123", result);
}
[TestMethod]
public void FailToParseAThreeDigitNumberBecauseItIsTooShort()
{
  var result = threeNumberParser.TryParse("10");
  Assert.IsTrue(result.ToString().StartsWith("Parsing failure"));
}
```

Sprache Simple

To begin, let's start with a parser that knows how to parse user-entered phone numbers into a PhoneNumber domain type. For simplicity, I'll stick with the U.S.-style format—(nnn) nnn-nnnn—but we want to specifically recognize the breakdown in area codes, prefix and line, and allow for letters in the place of digits (so somebody can enter their phone number as "(800) EAT-NUTS" if they desire). Ideally, the PhoneNumber domain type will convert between alpha and all-numeric forms on demand, but that functionality will be left as an exercise to the reader (meaning, essentially, that I don't want to bother with it).

> There are several parser combinator libraries available for the Microsoft .NET Framework.

(The pedant in me demands to point out that simply converting all the alphas to numbers isn't a fully compliant solution, by the way. In college, it was common in my circle of friends to try to come up with phone numbers that spelled out "cool" things—one ex-roommate is still waiting for 1-800-CTHULHU to become free, in fact, so he can win the game for all eternity.)

The easiest place to start is with the PhoneNumber domain type:

```
class PhoneNumber
{
  public string AreaCode { get; set; }
  public string Prefix { get; set; }
  public string Line { get; set; }
}
```

Were this a "real" domain type, AreaCode, Prefix and Line would have validation code in their property-set methods, but that would lead to a repetition of code between the parser and the domain class (which, by the way, we'll fix before this is all done).

Next, we need to know how to create a simple parser that knows how to parse *n* number of digits:

```
public static Parser<string> numberParser =
  Parse.Digit.AtLeastOnce().Text();
```

Defining the numberParser is straightforward. Begin with the primitive parser Digit (an instance of a Parser<T> defined on the

Figure 2 **Converting Input into a PhoneNumber Object**

```
public static Parser<string> fourNumberParser =
  Parse.Numeric.Then(first =>
    Parse.Numeric.Then(second =>
      Parse.Numeric.Then(third =>
        Parse.Numeric.Then(fourth =>
          Parse.Return("" + first.ToString() +
            second.ToString() + third.ToString() +
              fourth.ToString())))));
public static Parser<string> areaCodeParser =
  (from number in threeNumberParser
  select number).
  XOr(
  from lparens in Parse.Char('(')
  from number in threeNumberParser
  from rparens in Parse.Char(')')
  select number);
public static Parser<PhoneNumber> phoneParser =
  (from areaCode in areaCodeParser
  from _1 in Parse.WhiteSpace.Many().Text()
  from prefix in threeNumberParser
  from _2 in (Parse.WhiteSpace.Many().Text()).
  Or(Parse.Char('-').Many())
  from line in fourNumberParser
  select new PhoneNumber() { AreaCode=areaCode, Prefix=prefix, Line=line});
```

Sprache.Parse class), and describe that we want at least one digit in the input stream, implicitly consuming all digits until the input stream either runs dry or the parser encounters a non-digit character. The Text method converts the stream of parsed results into a single string for our consumption.

Testing this is pretty easy—feed it a string and let 'er rip:

```
[TestMethod]
public void ParseANumber()
{
  string result = numberParser.Parse("101");
  Assert.AreEqual("101", result);
}
[TestMethod]
public void FailToParseANumberBecauseItHasTextInIt()
{
  string result = numberParser.TryParse("abc").ToString();
  Assert.IsTrue(result.StartsWith("Parsing failure"));
}
```

When run, this stores "101" into result. If the Parse method is fed an input string of "abc," it will yield an exception. (If nonthrowing behavior is preferred, Sprache also has a TryParse method that returns a Result object that can be interrogated regarding success or failure.)

The phone-number parsing situation is a little bit more complicated, though; it needs to parse just three or four digits—no more, no less. Defining one such parser (the three-digit parser) is a bit trickier, but still doable:

```
public static Parser<string> threeNumberParser =
  Parse.Numeric.Then(first =>
    Parse.Numeric.Then(second =>
      Parse.Numeric.Then(third =>
        Parse.Return(first.ToString() +
          second.ToString() + third.ToString()))));
```

The Numeric parser takes a character and, if it's a digit, advances to the next character. The Then method takes a function (in the form of a lambda expression) to execute. The Return method collects each of these into a single string and, as its name implies, uses that as the return value (see **Figure 1**).

Success. So far. (Yes, the definition of threeNumberParser is awkward—surely there has to be a better way to define this! Fear not: there is, but to understand how to extend the parser, we have to dive deeper into how Sprache is constructed, and that's the subject of the next part in this series.)

Now, however, we need to handle the left-parens, the right-parens and the dash, and convert everything into a PhoneNumber object. It might seem a bit awkward with what we see so far, but watch what happens next, as shown in **Figure 2**.

Using the parser becomes pretty straightforward at this point:

```
[TestMethod]
public void ParseAFullPhoneNumberWithSomeWhitespace()
{
  var result = phoneParser.Parse("(425) 647-4526");
  Assert.AreEqual("425", result.AreaCode);
  Assert.AreEqual("647", result.Prefix);
  Assert.AreEqual("4526", result.Line);
}
```

Best of all, the parser is entirely extensible, because it, too, can be composed into a larger parser that transforms text input into an Address object or ContactInfo object or anything else imaginable.

The Combinatorics Concept

Historically, parsing text has been the province of "language researchers" and academia, largely due to the complicated and difficult edit-generate-compile-test-debug cycle inherent with generative metaprogramming solutions. Trying to walk through computer-generated code—particularly the finite-state-machine-based versions that many parser generators churn out—in a debugger is a challenge to even the most hard-bitten developer. For that reason, most developers don't think about solutions along parsing lines when presented with a text-based problem. And, in truth, most of the time, a parser- generator-based solution would be drastic overkill.

> Most developers don't think about solutions along parsing lines when presented with a text-based problem.

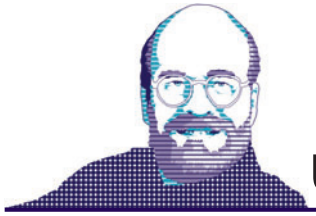
Parser combinators serve as a nice in-between solution: flexible enough and powerful enough to handle some nontrivial parsing, without requiring a Ph.D. in computer science to understand how to use them. Even more interestingly, the concept of combinatorics is a fascinating one, and leads to some other interesting ideas, some of which we'll explore later.

In the spirit in which this column was born, make sure to keep an "eye" out for my next column (sorry, couldn't resist), in which I'll extend Sprache just a touch, to reduce the ugliness of the three- and four-digit parsers defined here.

Happy coding! ■

TED NEWARD *is an architectural consultant with Neudesic LLC. He has written more than 100 articles and authored or coauthored a dozen books, including "Professional F# 2.0" (Wrox, 2010). He's a C# MVP and speaks at conferences around the world. He consults and mentors regularly—reach him at ted@tedneward.com or Ted.Neward@neudesic.com if you're interested in having him come work with your team, or read his blog at blogs.tedneward.com.*

THANKS to the following technical expert for reviewing this article: *Luke Hoban*



CHARLES PETZOLD

Video Feeds on Windows Phone 7

The modern smartphone is packed with electronic sensory organs through which it can obtain information about the outside world. These include the cell phone radio itself, Wi-Fi, GPS, a touchscreen, motion detectors and more.

To the application programmer, these sensory organs are only as useful as the APIs associated with them. If the APIs are deficient, the hardware features become much less valuable or even worthless.

From the perspective of an application programmer, one of the features missing from the initial release of Windows Phone was a good eyeball. Although the camera has always been in Windows Phone, the only API available in the original release was CameraCaptureTask. This class essentially spawns a child process that lets the user take a photo, and then returns that picture to the application. That's it. The application can't control any part of this process, nor can it obtain the live video feed coming through the lens.

That deficiency has now been corrected with two sets of programming interfaces.

One set of APIs concerns the Camera and PhotoCamera classes. These classes allow an application to assemble an entire photo-taking UI, including flash options; live preview video feed; shutter key presses and half-presses; and focus detection. I hope to discuss this interface in a future column.

The APIs I'll be discussing in this column were inherited from the Silverlight 4 webcam interface. They let an application obtain live video and audio feeds from the phone's camera and microphone. These feeds can be presented to the user, saved to a file or—and here it gets more interesting—manipulated or interpreted in some way.

Devices and Sources

The webcam interface of Silverlight 4 has been enhanced just a little for Windows Phone 7, and consists of about a dozen classes defined in the System.Windows.Media namespace. You'll always begin with the static CaptureDeviceConfiguration class. If the phone supports multiple cameras or microphones, these are available from the GetAvailableVideoCaptureDevices and GetAvailableAudioCaptureDevices methods. You might want to present these to the user in a list for selection. Alternatively, you can simply call the GetDefaultVideoCaptureDevice and GetDefaultAudioCaptureDevice methods.

Documentation mentions these methods might return null, probably indicating the phone doesn't contain a camera. This is unlikely, but it's a good idea to check for null anyway.

These CaptureDeviceConfiguration methods return instances of VideoCaptureDevice and AudioCaptureDevice or collections of instances of these two classes. These classes provide a friendly name for the device, a SupportedFormats collection and a DesiredFormat property. For video, the formats involve the pixel dimensions of each frame of video, the color format and frames per second. For audio, the format specifies the number of channels, the bits per sample and the wave format, which is always Pulse Code Modulation (PCM).

A Silverlight 4 application must call the CaptureDeviceConfiguration.RequestDeviceAccess method to obtain permission from the user to access the webcam. This call must be in response to user input, such as a button click. If the CaptureDeviceConfiguration.AllowedDeviceAccess property is true, however, then the user has already given permission for this access and the program needn't call RequestDeviceAccess again.

Obviously, the RequestDeviceAccess method serves to protect the privacy of the user. But the Web-based Silverlight and Silverlight for Windows Phone 7 seem to be a little different in this respect. The idea of a Web site surreptitiously accessing your webcam is decidedly creepy, but much less so for a phone program. It's my experience that for a Windows Phone application, AllowedDeviceAccess always returns true. Nevertheless, in all the programs described in this column, I've defined a UI to call RequestDeviceAccess.

Figure 1 **The MainPage.xaml File from StraightVideo**

```
<phone:PhoneApplicationPage
  x:Class="StraightVideo.MainPage"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:phone="clr-namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
  xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
  ...
  SupportedOrientations="Landscape" Orientation="LandscapeLeft"
  shell:SystemTray.IsVisible="True">

  <Grid x:Name="LayoutRoot" Background="Transparent">
    <Grid.RowDefinitions>
      <RowDefinition Height="Auto"/>
      <RowDefinition Height="*"/>
    </Grid.RowDefinitions>

    <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
      <TextBlock x:Name="ApplicationTitle" Text="STRAIGHT VIDEO"
        Style="{StaticResource PhoneTextNormalStyle}"/>
    </StackPanel>

    <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
      <Grid.Background>
        <VideoBrush x:Name="videoBrush" />
      </Grid.Background>
      <Button Name="startButton"
        Content="start"
        HorizontalAlignment="Center"
        VerticalAlignment="Center"
        Click="OnStartButtonClick" />
    </Grid>
  </Grid>
</phone:PhoneApplicationPage>
```

Code download available at code.msdn.microsoft.com/mag201112UIFrontiers.

Figure 2 **The MainPage.xaml.cs File from StraightVideo**

```
public partial class MainPage : PhoneApplicationPage
{
  CaptureSource captureSource;

  public MainPage()
  {
    InitializeComponent();

    captureSource = new CaptureSource
    {
      VideoCaptureDevice =
        CaptureDeviceConfiguration.GetDefaultVideoCaptureDevice()
    };
  }

  protected override void OnNavigatedTo(NavigationEventArgs args)
  {
    if (captureSource !=
      null && CaptureDeviceConfiguration.AllowedDeviceAccess)
    {
      videoBrush.SetSource(captureSource);
      captureSource.Start();
      startButton.Visibility = Visibility.Collapsed;
    }
    base.OnNavigatedTo(args);
  }

  protected override void OnNavigatedFrom(NavigationEventArgs args)
  {
    if (captureSource != null && captureSource.State == CaptureState.Started)
    {
      captureSource.Stop();
      startButton.Visibility = Visibility.Visible;
    }
    base.OnNavigatedFrom(args);
  }

  void OnStartButtonClick(object sender, RoutedEventArgs args)
  {
    if (captureSource != null &&
        (CaptureDeviceConfiguration.AllowedDeviceAccess ||
        CaptureDeviceConfiguration.RequestDeviceAccess())
    {
      videoBrush.SetSource(captureSource);
      captureSource.Start();
      startButton.Visibility = Visibility.Collapsed;
    }
  }
}
```

The application must also create a CaptureSource object, which combines a video device and an audio device into a single stream of live video and audio. CaptureSource has two properties, named VideoCaptureDevice and AudioCaptureDevice, that you set to instances of VideoCaptureDevice and AudioCaptureDevice obtained from CaptureDeviceConfiguration. You needn't set both properties if you're interested in only video or only audio. In the sample programs in this column, I've focused entirely on video.

After creating a CaptureSource object, you can call the object's Start and Stop methods. In a program dedicated to obtaining video or audio feeds, you'll probably want to call Start in the OnNavigatedTo override and Stop in the OnNavigatedFrom override.

In addition, you can use the CaptureImageAsync method of CaptureSource to obtain individual video frames in the form of WriteableBitmap objects. I won't be demonstrating that feature.

Once you have a CaptureSource object, you can go in one of two directions: You can create a VideoBrush to display the live video feed, or you can connect CaptureSource to a "sink" object to get access to raw data or to save to a file in isolated storage.

The VideoBrush

Definitely the easiest CaptureSource option is the VideoBrush. Silverlight 3 introduced the VideoBrush with a MediaElement source, and Silverlight 4 added the CaptureSource alternative for VideoBrush. As with any brush, you can use it to color element backgrounds or foregrounds.

In the downloadable code for this column is a program called StraightVideo that uses VideoCaptureDevice, CaptureSource and VideoBrush to display the live video feed coming through the default camera lens. **Figure 1** shows a good chunk of the MainPage.xaml file. Notice the use of landscape mode (which you'll want for video feeds), the definition of the VideoBrush on the Background property of the content Grid and the Button for obtaining user permission to access the camera.

Figure 2 shows much of the codebehind file. The CaptureSource object is created in the page's constructor, but it's started and stopped in the navigation overrides. I also found it necessary to call SetSource on the VideoBrush in OnNavigatedTo; otherwise the image was lost after a previous Stop call.

You can run this program on the Windows Phone Emulator, but it's much more interesting on a real device. You'll notice that the rendering of the video feed is very responsive. Evidently the video feed is going directly to the video hardware. (More evidence for this supposition is that my customary method of obtaining screenshots from the phone by rendering the PhoneApplicationFrame object to a WriteableBitmap didn't work with this program.) You'll also notice that because the video is rendered via a brush, the brush is stretched to the dimensions of the content Grid and the image is distorted.

Figure 3 **Sharing VideoBrush Objects in FlipXYVideo**

```
void CreateRowsAndColumns()
{
  videoPanel.Children.Clear();
  videoPanel.RowDefinitions.Clear();
  videoPanel.ColumnDefinitions.Clear();

  for (int row = 0; row < numRowsCols; row++)
    videoPanel.RowDefinitions.Add(new RowDefinition
    {
      Height = new GridLength(1, GridUnitType.Star)
    });

  for (int col = 0; col < numRowsCols; col++)
    videoPanel.ColumnDefinitions.Add(new ColumnDefinition
  {
    Width = new GridLength(1, GridUnitType.Star)
  });

  for (int row = 0; row < numRowsCols; row++)
    for (int col = 0; col < numRowsCols; col++)
    {
      Rectangle rect = new Rectangle
      {
        Fill = videoBrush,
        RenderTransformOrigin = new Point(0.5, 0.5),
        RenderTransform = new ScaleTransform
        {
          ScaleX = 1 - 2 * (col % 2),
          ScaleY = 1 - 2 * (row % 2),
        },
      };
      Grid.SetRow(rect, row);
      Grid.SetColumn(rect, col);
      videoPanel.Children.Add(rect);
    }
  fewerButton.IsEnabled = numRowsCols > 1;
}
```

One of the nice features of brushes is that they can be shared among multiple elements. That's the idea behind FlipXYVideo. This program dynamically creates a bunch of tiled Rectangle objects in a Grid. The same VideoBrush is used for each, except every other Rectangle is flipped vertically or horizontally or both, as shown in **Figure 3**. You can increase or decrease the number of rows and columns from ApplicationBar buttons.

This program is fun to play with, but not as much fun as the kaleidoscope program, which I'll discuss shortly.

Source and Sink

The alternative to using a VideoBrush is connecting a CaptureSource object to an AudioSink, VideoSink or FileSink object. The use of the word "sink" in these class names is in the sense of "receptacle" and is similar to the word's use in electronics or network theory. (Or think of a "heat source" and a "heat sink.")

The FileSink class is the preferred method for saving video or audio streams to your application's isolated storage without any intervention on your part. If you need access to the actual video or audio bits in real time, you'll use VideoSink and AudioSink. These two classes are abstract. You derive a class from one or both of these abstract classes and override the OnCaptureStarted, OnCaptureStopped, OnFormatChange and OnSample methods.

The class that you derive from VideoSink or AudioSink will always get a call to OnFormatChange before the first call to OnSample. The information supplied with OnFormatChange indicates how the sample data is to be interpreted. For both VideoSink and AudioSink, the OnSample call provides timing information and an array of bytes. For AudioSink, these bytes represent PCM data. For VideoSink, these bytes are rows and columns of pixels for each frame of video. This data is always raw and uncompressed.

Both OnFormatChange and OnSample are called in secondary threads of execution, so you'll need to use a Dispatcher object for tasks within these methods that must be performed in the UI thread.

The StraightVideoSink program is similar to StraightVideo except the video data comes through a class derived from VideoSink. This derived class (shown in **Figure 4**) is named SimpleVideoSink because it merely takes the OnSample byte array and transfers it to a WriteableBitmap.

MainPage uses that WriteableBitmap with an Image element to display the resultant video feed. (Alternatively, it could create an ImageBrush and set that to the background or foreground of an element.)

Here's the catch: That WriteableBitmap can't be created until the OnFormatChange method is called in the VideoSink derivative, because that call indicates the size of the video frame. (It's usually 640x480 pixels on my phone but conceivably might be something else.) Although the VideoSink derivative creates the WriteableBitmap, MainPage needs to access it. That's why I defined a constructor for SimpleVideoSink that contains an Action argument to call when the WriteableBitmap is created.

Notice that the WriteableBitmap must be created in the program's UI thread, so SimpleVideoSink uses a Dispatcher object to queue up the creation for the UI thread. This means that the WriteableBitmap might not be created before the first OnSample call. Watch out for that! Although the OnSample method can access the Pixels array of the WriteableBitmap in a secondary thread, the call to Invalidate the bitmap must occur in the UI thread because that call ultimately affects the display of the bitmap by the Image element.

The MainPage class of StraightVideoSink includes an ApplicationBar button to toggle between color and gray-shaded video feeds. These are the only two options, and you can switch to one or the other by setting the DesiredFormat property of the VideoCaptureDevice object. A color feed has 4 bytes per pixel in the order green, blue, red and alpha (which will always be 255). A gray-shade feed has only 1 byte per pixel. In either case, a WriteableBitmap always has 4 bytes per pixel where every pixel is represented by a 32-bit integer with the highest 8 bits for the alpha channel, followed by

Figure 4 **The SimpleVideoSink Class Used in StraightVideoSink**

```
public class SimpleVideoSink : VideoSink
{
  VideoFormat videoFormat;
  WriteableBitmap writeableBitmap;
  Action<WriteableBitmap> action;

  public SimpleVideoSink(Action<WriteableBitmap> action)
  {
    this.action = action;
  }

  protected override void OnCaptureStarted() { }

  protected override void OnCaptureStopped() { }

  protected override void OnFormatChange(VideoFormat videoFormat)
  {
    this.videoFormat = videoFormat;

    System.Windows.Deployment.Current.Dispatcher.BeginInvoke(() =>
    {
      writeableBitmap = new WriteableBitmap(videoFormat.PixelWidth,
        videoFormat.PixelHeight);
      action(writeableBitmap);
    });
  }

  protected override void OnSample(long sampleTimeInHundredNanoseconds,
    long frameDurationInHundredNanoseconds, byte[] sampleData)
  {
    if (writeableBitmap == null)
      return;

    int baseIndex = 0;

    for (int row = 0; row < writeableBitmap.PixelHeight; row++)
    {
      for (int col = 0; col < writeableBitmap.PixelWidth; col++)
      {
        int pixel = 0;

        if (videoFormat.PixelFormat == PixelFormatType.Format8bppGrayscale)
        {
          byte grayShade = sampleData[baseIndex + col];
          pixel = (int)grayShade | (grayShade << 8) |
            (grayShade << 16) | (0xFF << 24);
        }
        else
        {
          int index = baseIndex + 4 * col;
          pixel = (int)sampleData[index + 0] | (sampleData[index + 1] << 8) |
            (sampleData[index + 2] << 16) | (sampleData[index + 3] << 24);
        }
        writeableBitmap.Pixels[row * writeableBitmap.PixelWidth + col] = pixel;
      }
      baseIndex += videoFormat.Stride;
    }
    writeableBitmap.Dispatcher.BeginInvoke(() =>
      {
        writeableBitmap.Invalidate();
      });
  }
}
```

Figure 5 **The Source Triangle for a Kaleidoscope**

Figure 6 **The Destination Bitmap for a Kaleidoscope**

Figure 7 **The Destination Bitmap Showing the Source Triangles**

red, green and blue. The CaptureSource object must be stopped and restarted when switching formats.

Although StraightVideo and StraightVideoSink both display the live video feed, you'll probably notice that StraightVideoSink is considerably more sluggish as a result of the work the program is doing to transfer the video frame to the WriteableBitmap.

Making a Kaleidoscope

If you just need a real-time video feed with occasional frame captures, you can use the CaptureImageAsync method of CaptureSource. Because of the performance overhead, you'll probably restrict the use of VideoSink to more specialized applications involving the manipulation of the pixel bits.

Let's write such a "specialized" program that arranges the video feed into a kaleidoscopic pattern. Conceptually, this is fairly straightforward: The VideoSink derivative gets a video feed where each frame is probably 640x480 pixels in size or perhaps something else. You want to reference an equilateral triangle of image data from that frame as shown in **Figure 5**.

I decided on a triangle with its base on the top and its apex at the bottom to better capture faces.

The image in that triangle is then duplicated on a WriteableBitmap multiple times with some rotation and flipping so the images are tiled and grouped into hexagons without any discontinuities, as shown in **Figure 6**. I know the hexagons look like pretty flowers, but they're really just many images of my face (perhaps *too* many images of my face).

The pattern of repetition becomes more apparent when the individual triangles are delimited, as shown in **Figure 7**. When rendered on the phone, the height of the target WriteableBitmap will be the same as the phone's smaller dimension, or 480 pixels. Each equilateral triangle thus has a side of 120 pixels. This means that the height of the triangle is 120 times the square root of 0.75, or about 104 pixels. In the program, I use 104 for the math but 105 for sizing the bitmap to make the loops simpler. The entire resultant image is 630 pixels wide.

I found it most convenient to treat the total image as three identical vertical bands 210 pixels wide. Each of those vertical bands has reflection symmetry around the vertical midpoint, so I reduced the image to a single 105x480-pixel bitmap repeated six times, half of those with reflection. That band consists of just seven full triangles and two partial triangles.

Even so, I was quite nervous about the calculations necessary to assemble this image. Then I realized that these calculations wouldn't have to be performed at the video refresh rate of 30 times per second. They need only be performed once, when the size of the video image becomes available in the OnFormatChange override.

The resultant program is called Kaleidovideo. (That name would be considered an etymological abomination by traditionalists because "kaleido" comes from a Greek root meaning "beautiful form," but "video" has a Latin root, and when coining new words you're not supposed to mix the two.)

The KaleidoscopeVideoSink class overrides VideoSink. The OnFormatChange method is responsible for computing the members of an array called indexMap. That array has as many members as the number of pixels in the WriteableBitmap (105 multiplied by 480, or 50,400) and stores an index into the rectangular area of the video image. Using this indexMap array, the transfer of pixels from the video image to the WriteableBitmap in the OnSample method is pretty much as fast as conceivably possible.

This program is lots of fun. Everything in the world looks rather more beautiful in Kaleidovideo. **Figure 8** shows one of my bookcases, for example. You can even watch TV through it.

Just in case you see something on the screen you'd like to save, I added a "capture" button to the ApplicationBar. You'll find the images in the Saved Pictures folder of the phone's photo library. ■

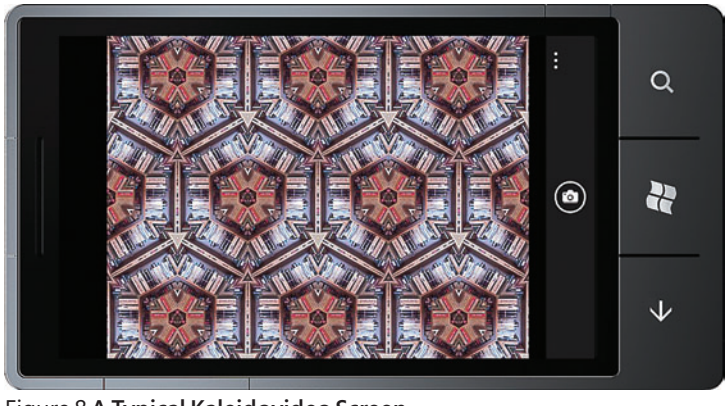


Figure 8 **A Typical Kaleidovideo Screen**

CHARLES PETZOLD *is a longtime contributing editor to* MSDN Magazine. *His recent book, Programming Windows Phone 7 (Microsoft Press, 2010), is available as a free download at bit.ly/cpebookpdf.*

THANKS to the following technical experts for reviewing this article: *Mark Hopkins and Matt Stroshane*

YOUR MAP TO THE .NET DEVELOPMENT PLATFORM

Visual Studio Live! Las Vegas

March 26-30, 2012

Mirage Resort and Casino, Las Vegas, NV

Registration now open!
vslive.com/lasvegas

Visual Studio Live! New York

May 14-17, 2012

New York Marriott at the Brooklyn Bridge

Visual Studio Live! Redmond

August 6-10, 2012

Microsoft Headquarters in Redmond, WA

Visual Studio Live! Orlando *TBD—Fall 2012*

DAVID PLATT

Jobs and Ritchie: Entangled Photons

Steve Jobs died on Oct. 5, and the tributes resounded through the universe. Far less noted was the passing of Dennis Ritchie just three days later. The two men were the yin and yang of modern computing, completely different, yet linked over time—like entangled photons acting upon each other at a distance.

You know all about Jobs the iconoclast, dropping out of college after just one term, the maverick, always upsetting things. Contrast him with Ritchie the Harvard Ph.D., working in the ultimate establishment at Bell Labs for 30 years. The news of Jobs' death flashed instantly through every channel, but I didn't notice Ritchie's passing until a week after he died.

> Civilians have never heard of Dennis Ritchie, but everyone in the geek business today works directly with something of his, or with something built on his innovations.

Jobs' accomplishments are familiar to all. To my mind, his greatest was the iPod, Apple's first foray into mass-market consumer products and the company's first gadget, if you will, back in 2001. How long ago that seems, how different the world was then. The Microsoft .NET Framework hadn't shipped yet, the now-ubiquitous smartphone didn't exist, and my older daughter (now an eye-rolling pre-teen) was taking her first baby steps chasing Simba, the family cat. If you wanted music, you had to carry not only your Discman (creation of Sony's Akio Morita, another iconoclast), but also your stack of CDs. You had to manually swap them in and out, dropping them and scratching them and wiping fingerprints off them, playing tracks in the order in which they were burned.

Jobs changed all that, moving music from backpack to pocket. The lessons he learned in the process enabled Apple's success in the smartphone and tablet markets. He realized that not only did the gadget itself need to be great, but it also required an ongoing business infrastructure; hence the iTunes store. Pretty much single-handedly, Jobs brought down the music industry establishment, much as Craigslist brought down the newspaper industry.

Civilians have never heard of Dennis Ritchie, but everyone in the geek business today works directly with something of his, or with something built on his innovations. Ritchie and Ken Thompson developed the Unix OS, for which they shared the prestigious Turing Award. He developed the C language (so named because it followed one called B) in the early 1970s.

C was my first industrial language. I used it for controlling a chaff decoy launcher on warships, then an ion implantation machine in a semiconductor fab, then a financial trader's dealing system. At my last company, we said that to get a job there, a programmer had to have a beard (I qualified) and know how to spell C (I struggled, but eventually mastered it).

Many successful languages are built on C. Bjarne Stroustrup designed C++, the first industrial object-oriented language. C# and Java hark back to C as well; from semicolons and curly braces to the "main" entry point. You can think of the sharp sign as two ++ operators stacked on top of each other, and leaning forward as if in motion. Objective C, a Smalltalk-influenced derivative, has become the primary language for the Apple world.

Jobs was a pioneer of marketing and design, of figuring out what customers really wanted before they knew it themselves. But without Ritchie's work, Jobs wouldn't have been able to develop the software that made his devices hum. Now Jobs and Ritchie forever share the sad confluence of their passing, and the sense that each got short-changed by at least a decade.

So long then, and thank you, to a pair of guys who did great things, one of them known to the whole world, and one known only to us geeks. Lots of people mourn Jobs. Let's you and I hoist one for Ritchie, because we're the only ones who will. You get to buy. ■

MSDN Magazine *lost a great columnist on Aug. 29, with the passing of Simba ("The Cat Butt Factor,"* msdn.microsoft.com/magazine/gg983482*). She crossed over peacefully, sleeping in a sunbeam in her favorite garden, aged 20 years and 3 months (roughly 120 human years). May you and I be as lucky, dear reader. I thought I'd be getting a lot more work done now without her jumping on my keyboard, but somehow it hasn't worked out that way.*

DAVID S. PLATT *teaches Programming .NET at Harvard University Extension School and at companies all over the world. He's the author of 11 programming books, including "Why Software Sucks" (Addison-Wesley Professional, 2006) and "Introducing Microsoft .NET" (Microsoft Press, 2002). Microsoft named him a Software Legend in 2002. He wonders whether he should tape down two of his daughter's fingers so she learns how to count in octal. You can contact him at rollthunder.com.*

360° Application Transformation

From start…

To finish

Syncfusion
Deliver innovation with ease